A Flexible High-Density Sensor Network

by

Behram Farrokh Thomas Mistree

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2008

Author .....
Department of Electrical Engineering and Computer Science
Aug. 3, 2008

Certified by .....
Joseph A. Paradiso
Associate Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

A Flexible High-Density Sensor Network

by

Behram Farrokh Thomas Mistree

Submitted to the Department of Electrical Engineering and Computer Science
on Aug. 3, 2008, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

This thesis explores building and deploying a scalable electronic sensate skin that was designed as a dense sensor network. Our skin is built from small (1" x 1") rigid circuit boards attached to their neighbors with flexible interconnects. Each boardcontained an embedded processor together with a suite of thirteen sensors, providing dense, multimodal capture of proximate and contact phenomena. In addition to the design of the physical system, this thesis develops protocols for internode communication (both neighbor-neighbor and global), and power-efficient wake-on-phenomena operation. The system was rigorously tested with an array of up to 4x3 nodes subject to a variety of sensor stimuli. Although there were some robustness issues in the final design (particularly in the wired interconnects, which were not the focus of this thesis work), the skin that we developed showed good flexibility for a prototype, ran quickly and efficiently, and could detect and respond to a variety of stimuli.

Thesis Supervisor: Joseph A. Paradiso
Title: Associate Professor

Acknowledgments

I owe a deep thank you to **Prof. Paradiso** for giving me a chance to work on this project. I learned a lot and matured over the past year. It motivated me to continue my study of electrical engineering, and introduced me to the importance of research.

I would also like to thank my brother, **Dinsha Mistree**, primarily for being a great brother. He helped me write my thesis, but, most importantly, encouraged me throughout the last year. Similarly, my parents, **Janet K. Allen** and **Farrokh Mistree** were a constant source of support.

In addition, I would like to thank several current members of the Responsive Environments Group:

- **Mark Feldmeier**: For explaining basic circuitry and helping me use oscilloscopes.
- **Nan-wei Gong**: For being incredibly enthusiastic and great to be around; and also for teaching me as she learned.
- **Mat Laibowitz**: For explaining MSP430 function to me, being an inspiration with a soldering iron, and discussing great and not so great movies every so often.
- **Jason LaPenta**: For looking over much of my board layout and coaching me on low-noise PCB techniques.
- **Michael Lapinski**: For teaching me basic life lessons.
- **Lisa Lieberson**: For putting up with multiple questions and helping me procure parts.
- **Manas Mittal**: For being around late at night, playing loud music, and walking to 7-eleven together.

I would also like to thank a former member of the Responsive Environments Group, **Gerardo Perez**, for providing an excellent starting point for my thesis.

Finally, I would like to thank the National Science Foundation along with the Things That Think Consortium and other sponsors of the MIT Media Laboratory for funding me while working on this project.

Contents

1 Flexible High-Density Sensor Networks: The Starting Point **17**

1.1 Physical Objectives 18

1.1.1 Package Size 18

1.1.2 Bendable 19

1.1.3 Electrical/Computational/Low Power 20

1.1.4 Sensing Modalities 20

1.2 Related Work 21

2 Hardware **25**

2.1 Microcontroller 26

2.2 Mux 26

2.3 Light Sensor 27

2.4 Pressure Sensors 29

2.5 Microphone and Whisker 30

2.5.1 Microphone 30

2.5.2 Whisker Sensor 34

2.6 Temperature Sensor 35

2.7 Hall Effect Sensor 36

2.8 Connectors 37

2.9 Master Node 40

2.10 Power Consumption Summary 40

3 Software **43**

3.1 Firmware . . . 44

3.1.1 Slave Node Firmware . . . 44

3.1.2 Master Node Firmware . . . 49

3.2 Visualization . . . 50

3.2.1 Overview and Motivation . . . 50

3.2.2 Visualization Goals . . . 51

3.2.3 Graphics Toolbox Choice . . . 52

3.2.4 Sensor-Visualization Mapping . . . 52

4 Communication **59**

4.1 Motivation . . . 59

4.2 I2C Bus . . . 61

4.2.1 Overview . . . 61

4.2.2 Advantages of I2C Bus . . . 62

4.2.3 Disadvantages of the I2C Bus . . . 63

4.3 Asynchronous Peer-to-peer Communication . . . 66

4.3.1 Motivation . . . 66

4.3.2 Implementation . . . 67

5 Results **75**

5.1 Basic Results . . . 75

5.1.1 Size . . . 75

5.1.2 Flexibility . . . 76

5.1.3 Robustness . . . 77

5.2 Communication Results . . . 83

5.2.1 Peer-to-peer . . . 84

5.2.2 I2C . . . 89

5.3 Basic Sensor Results . . . 91

5.3.1 Light Results . . . 91

5.3.2 Microphone Results . . . 92

5.3.3 Whisker Results . . . 96
5.3.4 Pressure Results . . . 98
5.3.5 Temperature Sensor . . . 100
5.4 Extended Results . . . 108
5.4.1 Proximity Events . . . 108
5.4.2 Magnetic Bend Sensor . . . 111
5.4.3 Hand Press . . . 115
5.5 Power Exploration . . . 121

6 Discussion **129**
6.1 Communication . . . 129
6.1.1 Peer-to-peer Protocol . . . 129
6.2 Light . . . 132
6.3 Temperature . . . 133
6.4 Whisker . . . 134
6.5 Microphone . . . 136
6.6 Pressure . . . 136
6.7 Magnetic Bend Sensors . . . 137

7 Conclusion and Future Work **141**
7.1 Errata to be Fixed . . . 141
7.1.1 Temperature Sensor . . . 141
7.1.2 Connectors . . . 142
7.1.3 Hall Effect Sensors . . . 142
7.2 Potential Extensions . . . 143
7.2.1 Visualization . . . 143
7.2.2 Enlarging our Skin . . . 143
7.2.3 Distributed Control . . . 144
7.2.4 Power . . . 146
7.3 Summary . . . 147

A Simulation Code **149**

A.1 Skin Simulation 149

A.2 Magnet Simulation 149

B Additional Data **153**

B.1 Table Pound 153

B.2 Stomp on floor 153

B.3 Skin Yank 153

B.4 Whisker Blow 157

C PCB Layout **159**

D Circuit Schematics **165**

List of Figures

1-1 Perez's S.N.A.K.E system 24

2-1 Picture of mux on node board 27
2-2 Picture of TPS851 light sensor mounted on our board. 28
2-3 QTC pressure versus resistance 29
2-4 Picture of FSR sensor 30
2-5 Diagram of pressure sensor and standoff 31
2-6 The OPA347's transfer function 32
2-7 Picture of SPM0102NE-3 microphone mounted on our node 33
2-8 Conditioning circuitry for microphone 34
2-9 Conditioning circuitry for whisker sensor 35
2-10 LM20CIM temperature sensor on node board 36
2-11 Picture of Hall effect sensors and magnets mounted on node board . . 38
2-12 Diagram showing internode connections 39
2-13 Picture of wired connectors selected for linking nodes 39

3-1 Light sensor visualization 54
3-2 Temperature sensor visualization 54
3-3 Pressure sensor visualization 55
3-4 Microphone visualization 56
3-5 Bend sensor visualization 56

4-1 A simplified diagram of the I2C's electrical connections. (Image from [11]) 62

4-2 Pin connections between nodes for peer-to-peer communication. . . . 68
4-3 Flow chart of peer-to-peer communication 71

5-1 Four nodes. 76
5-2 Four nodes bent around a camera case 77
5-3 Four nodes bent into a strange topology 78
5-4 Thirteen nodes bent into an unusual topology 79
5-5 Thirteen nodes hanging off edge of a table 80
5-6 Picture of two connectors side by side 82
5-7 Block diagram depicting the experimental procedure used to measure the accuracy of peer-to-peer transmission 85
5-8 Block diagram depicting the experimental procedure used to measure the baud rate of peer-to-peer transmission 86
5-9 Block diagram depicting the experimental procedure used to demonstrate the effects of allowing a node to set different peer-to-peer communication rates 88
5-10 Response of twelve nodes to flashlight stimulus 93
5-11 Response of nine nodes to hand shadows 94
5-12 Windowed microphone response to five claps of varying intensity . . . 94
5-13 Un-windowed microphone response to five claps of varying intensity . 95
5-14 Windowed microphone responses to five claps of twelve nodes connected in a grid. 96
5-15 Un-windowed microphone responses to five claps of twelve nodes connected in a grid. 97
5-16 Whisker responses of a grid of nine nodes as a hand passes over the tips of the whisker bristles three times 98
5-17 FSR responses of a grid of nine nodes as each node is sequentially pressed upon 99
5-18 Comparison of FSR sensor against QTC sensor 100
5-19 Temperature sensor response to heat gun 101

5-20 Temperature sensor response to thumb's being applied to it 102
5-21 Filtered response of temperature sensor to heat gun 103
5-22 Filtered response of temperature sensor to thumb's being applied to it 104
5-23 Filtered temperature response of twelve nodes after an incandescent light bulb is turned on 105
5-24 Filtered temperature response of twelve nodes after an incandescent light bulb is turned off 106
5-25 Spatial temperature gradient across a grid of twelve nodes 107
5-26 Eleven nodes' light sensor response to shadow 109
5-27 Whisker sensor waveforms of grid 110
5-28 Whisker data from four nodes as a hand passes over each sequentially 111
5-29 Responses of six Hall effect sensors 113
5-30 West North Hall effect sensor results as skin is bent to an angle of 60^o three times 114
5-31 Light sensor data from a twelve node grid as a hand descends, presses on, and leaves grid 116
5-32 Whisker sensor data from a twelve node grid as a hand descends, presses on, and leaves grid 117
5-33 Filtered temperature sensor data from a twelve node grid as a hand descends, presses on, and leaves grid 118
5-34 Microphone data from a twelve node grid as a hand descends, presses on, and leaves grid 119
5-35 Pressure sensor data from a twelve node grid as a hand descends, presses on, and leaves grid 120
5-36 Light sensor data from a twelve node grid as a hand rolls from the left side of the skin onto the right side of the skin and back. 121
5-37 Whisker sensor data from a twelve node grid as a hand rolls from the left side of the skin onto the right side of the skin and back 122
5-38 Temperature sensor data from a twelve node grid as a hand rolls from the left side of the skin onto the right side of the skin and back 123

5-39 Microphone data from a twelve node grid as a hand rolls from the left side of the skin onto the right side of the skin and back 124
5-40 Pressure sensor data from a twelve node grid as a hand rolls from the left side of the skin onto the right side of the skin and back 125
5-41 Node configuration for low power mode exploration 126

6-1 North West Hall effect sensor readings 139

A-1 Visual output of skin simulation program 150
A-2 Visual output of magnet simulation 151

B-1 Whisker sensor's response to a fist's pounding the table on which our skin is placed 154
B-2 Microphone's response to a fist's pounding the table on which our skin is placed 155
B-3 Whisker sensor's response to a foot's stomping the floor near our skin twice 156
B-4 Responses of whisker sensors to a sudden tug on the base of our skin 157
B-5 Responses of whisker sensors to someone's blowing across skin 158

C-1 PCB layout of entire node board 160
C-2 PCB layout of top layer of node board 161
C-3 PCB layout of interior layer of node board: VCC plane 162
C-4 PCB layout of interior layer of node board: GND plane 163
C-5 PCB layout of bottom layer of node board 164

D-1 Power, LED, temperature sensor, and light sensor circuitry for node . 166
D-2 Mux control circuitry for node 167
D-3 Microphone and whisker conditioning circuitry 168
D-4 Microcontroller and connector circuitry 169

List of Tables

2.1 Summary of current consumption for the components of each node . 41

5.1 Measurements of accuracy for peer-to-peer protocol 85

5.2 Measurements of baud rate for peer-to-peer protocol. 87

5.3 Ready-delay's effects on number of 8-bit characters received from neighbor via peer-to-peer protocol and processing time required to perform 10,000 square root operations. 90

5.4 Accuracy results of I2C bus 90

5.5 Hand speeds calculated from whisker sensors 112

5.6 Current consumption from skin patch wakeup routine 127

Chapter 1

Flexible High-Density Sensor Networks: The Starting Point

This thesis describes the function of a hardware and software platform built to replicate some of the sensory and mechanical aspects of skin. Throughout the rest of this document, for simplicity, we refer to this platform as "skin". Although such a designation is convenient, it is imperfect.

Biological skin is a complex organ capable of much more functionality than our project imitates. In addition to its sensory and mechanical characteristics, biological skin can regulate heat, protect the body from pathogens, and signal emotional state (for instance, a blush) [33]. While building a system capable of biological skin's extended functionality would be exciting and profound, as will be demonstrated in later chapters, our system provides a real and powerful starting point for a variety of explorations.

This thesis is divided into several chapters.

- Chapter 2 details our network's physical hardware, explaining both our choices of sensors as well as all additional circuitry necessary for our work's operation.
- Chapter 3 explains the firmware written for our skin as well as the software that we developed for real-time representation of our network's state.
- Chapter 4 describes our network's communication system.

- Chapter 5 presents results from our work including basic communication benchmarks and data collected from the network in response to a variety of stimuli.
- Chapter 6 discusses and analyzes the relevance of our results as well as highlighting some of their more interesting features.
- Chapter 7 concludes our work and posits additional avenues for future exploration.

The remainder of this chapter describes our project's goals and provides context for our system by presenting a review of relevant literature.

1.1 Physical Objectives

The previous section introduced our work, explaining our basic skin metaphor. While such an overview is useful at a high level, it leaves many design questions unspecified. Therefore, this section motivates and catalogs a specific list of our work's design and functional goals.

1.1.1 Package Size

Our skin is composed of a number of discrete nodes. Such a design provides researchers the flexibility to use our system not only to analyze stimuli, but also to perform more rigorous studies, such as examining the dynamic between local and global processing of stimuli. However, a poor implementation may be susceptible to information loss. Consider, for instance, two types of circular nodes each with a single, centrally located temperature sensor. Type 1 nodes are each the size of a CD while Type 2 nodes are each the size of a quarter. If one were to tile a sheet of paper with each type of node, one would be lucky to fit four or five Type 1 nodes; on the other hand, one could fit dozens and dozens of Type 2 nodes on the same sheet of paper. Such a discrepancy implies that Type 2 nodes would offer more temperature sensors per area than Type 1 nodes, making spatial interpolation of temperature values more reliable.

For modalities that tend to vary gradually, such as temperature in the example described above, one might argue that the increased data provided by Type 2 sensors adds little information to the overall system. However, for stimuli, such as pressure, that can have very distinct edges, increased spatial sampling could have a profound effect.

From this example, we see the importance of either shrinking nodes or increasing the number of sensors on each node. We choose the first option for two reasons. The first reason is mechanical: each node is connected to its neighbors through flexible interconnects. Therefore, decreasing the size of each node makes the overall network more generally flexible.

The second reason is slightly more complicated and touches upon robustness. With the Type 2 nodes from the example above, if a microprocessor fails, depending on the network architecture, the overall skin develops a small, but unsightly, quarter-sized blind spot. In contrast, a failure in a system composed of Type 1 nodes would create such a large sensor-less region that the system's overall functionality would be dramatically reduced.

In addition, smaller nodes imply an increase in node density, meaning that for a given area of skin there are more potential paths for information to be routed from one node to another. A network of smaller nodes might be able to route around a single communication failure more easily than a network of larger nodes. Therefore, smaller nodes might enhance the overall integrity of the skin's communication system.

1.1.2 Bendable

Practically, requiring our network to be bendable dramatically increases its application space. A bendable system could be deployed for situations in which topology is dynamic or unknown. In addition, bendable systems would be more interesting to interact with. Philosophically, one could read this requirement as a statement that our society is past the point where people and their spaces are forced to adapt to technology; instead, technology should mold itself to people and their environments.

1.1.3 Electrical/Computational/Low Power

Future applications for our skin might be in a mobile system, where it would be powered by a battery (for example, on a mobile robot). In such a context, the overall usefulness of our skin would greatly depend on how much power it consumed - a system that consumed less power would operate longer. As explained later in this work, for simplicity, the current version of our skin is wired directly to a static power source. This decision was made consciously - requiring the system to be battery powered would demand resources that were better spent in the development of a prototype. However, it is recognized that there might be great interest in eventually weaning our network away from a wall socket, and as such, our initial prototype is designed to be economical with its power usage.

There are, of course, other benefits to a low power system. In particular, if one created a large version of our skin, power inefficiencies might prove to be an unacceptably expensive energy waste.

1.1.4 Sensing Modalities

Our network is loosely modeled after skin - a remarkable multi-sensory organ capable of detecting temperature, pressure, proximity (hair), and light changes (in some species). As such, the modalities that our network is designed to sense are informed heavily, but not exclusively, by those modalities that biological skin is capable of sensing. Below is a list of stimuli that we designed each node to detect:

- Pressure: Our skin was designed for gentle human interaction. Therefore, the range of detection for pressure is between a light poke and a moderately heavy hand press.

- Sound: Typical skin does not evince the ability to distinguish sound. However, as explained later in our work audio amplitude measurements augmented tactile sensing (certain pressure stimuli are strong enough to also create an audio signature). In addition, sound allows us to attempt localization of stimuli not in direct contact with the network.

- Bend/Strain: The relative bend between nodes is an important value to measure - it gives information about the general topology of the network.

- Proximity/Airflow: Biological skin has the ability to distinguish simple proximity and airflow events in its environment using hair. It would be interesting if our system were able to track these events as well.

- Light: Light detectors marginally promote our skin metaphor: the skins of certain animals, such as cuttlefish, are capable of distinguishing small changes in light. More importantly however, light sensors give us a smorgasbord of information: sudden changes in a light sensor's state may indicate the shadow of an approaching object, while the signature of the light sensor's waveform may indicate a location (indoor lights have a characteristic 60 Hz hum), etc.

- Temperature: A sensitive temperature measurement could indicate the proximity or touch of a human or animal or signify the presence of a nearby (potentially dangerous) heat source.

A thorough description of our skin nodes' performance in detecting each of these modalities is presented in Chapter 5.

1.2 Related Work

Work on dense sensor systems is hardly unique. There are an overwhelming number of projects in the literature, some of which bear relevance to skins. A brief sampling:

- Rossignac, et al. outline work being done to embed pressure sensors within morphable materials in their paper, "Finger sculpting with Digital Clay." Although the project is still incomplete, the intention is to create a material whose 3d structure can be tracked by a computer as users shape and form it [35].

- Rekimoto describes a "SmartSkin" capacitative surface. The sensors of the surface feed information back to a single controlling PC which calculates hand position and shape from the aggregated data [34].

- Hakozaki uses inductive coupling to implement skin for a robot. Although their system is clearly impressive, it is solely restricted to pressure sensing and glosses over algorithmic questions of sensor data processing [16].

- Maximilian outlines a capacitive pressure array that is flexible enough to be used as a skin in his paper "A textile based capacitive pressure sensor" [30].

- Stiehl built a companion robot named "Huggable" for deployment in nursing homes and hospital settings [39]. In the form of a teddy bear, Huggable is designed to promote and detect human contact and interaction. To these ends, the developers of Huggable custom-built a synthetic skin for their platform capable of pressure and temperature sensing. Using a neural-network model to process sensor data, Huggable was capable of accurately classifying stimuli that ranged from petting to rubbing to squeezing [40]. Although, in deploying a flexible sensor system Stiehl's Huggable work shares several similarities with our system, there are important distinctions as well. Most notably, the Huggable system does not incorporate any distributed processing. Whereas each node in our array is capable of processing all of its sensory inputs, each Huggable platform only has one, central processor which samples and manages the entirety of the system's skin. As such, Huggable is unable to analyze the networking, communications, scalability, and control questions that our system is specifically built to explore. In addition, Huggable's sensor suite differs from ours. While both Huggable and our system have pressure and temperature sensors, Huggable has a capacitive sensor which we chose not to include for space considerations; and our system has light, whisker, sound, and magnetic sensors which are all absent on Huggable.

Such a list demonstrates the rich academic interest in the broad topic of creating dense tactile systems. However, these papers neither propose nor describe systems with the distributed processing capabilities we demonstrate in our work. Our skin is built not only with a high density of sensors, but also with a high density of processing power: as will be described later, approximately each square inch of our skin contains

a processor capable of running millions of instructions per second. In this way, we can understand our skin as a step towards Speckled Computing, a concept discussed in [4], [44], [5]. Speckled Computing explores the potential effects and functions that future, ultra-small electronics will permit, focusing on electrical and algorithmic questions of power, communication, and manufacturing.

As a hybrid between Speckled Computing and dense sensor networks, our skin traces its ancestry to a series of work initiated by Lumelsky et al.'s call for the development of such a system [27] and continued on through research at the MIT Media Laboratory's Responsive Environments Group. In particular, our skin extends Responsive Environments' research trajectory defined by Tribble, Pushpin Computing and S.N.A.K.E. Each of these systems is described separately below.

- Tribble: Developed by Lifton [31], the Tactile Reactive Interface Built By Linked Elements is a multi-modal and peer-to-peer sensor network. Similar to our skin's design, Tribble is made up of separate processing patches. However, Tribble differs in a few key respects from the system instantiated in this thesis. Specifically, Tribble's nodes are large, reducing its overall sensor resolution. In addition, Tribble's nodes are configured to form a rigid and inflexible sphere, and each node contains a speaker for producing distributed audio. Despite these differences, Tribble provides an excellent point of approach for this paper's work.

- Pushpin Computing: Pushpin Computing, also developed by Lifton, was a testbed that allowed researchers to place arbitrary numbers of distinct sensor nodes on a rigid substrate, and then investigate how to get those nodes to coordinate, aggregate, and manage the data that each collected [8], [7]. The nodes in the Pushpin network were not directly physically linked to each other. As such, our skin evinces distinct mechanical and electrical issues apart from those Lifton had to deal with. However, the algorithmic framework and general network principles that Lifton studied proved very useful in working with our skin.

- S.N.A.K.E.: In S.N.A.K.E., Perez proposed a system very similar to our skin

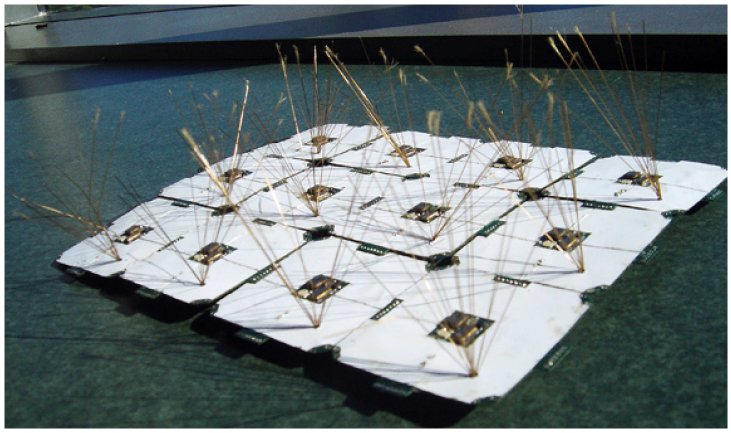

Figure 1-1: Photo of Perez's S.N.A.K.E system [33].

network, including flexible nodes with processing power and multi-modal sensor input. Pictured in Figure 1-1, Perez's overall system only achieved mixed results, largely due to mechanical issues that compromised robustness and flexibility[33]. Our project expands from Perez's basic objectives, and addresses several of the difficulties he faced with a fundamental re-design of the system - instead of each node's being fully flexible (which did not work well with the large IC packages needed on each node) and instead of using Perez's distinct springy multi-layer flex circuit board, our skin employs rigid nodes with flexible interconnects, taking some inspiration from midieval chainmail. In addition, compared to Perez's network, our improved skin reduces the area of each node, increasing sensor density and deploys several different, more effective sensors.

In summary, combining a high-density sensor network with embedded processing, our work is an amalgam of two relatively distinct fields: high-density sensor networks and speckled computing. This positions our work along a unique tract, only sparsely populated by projects such as Pushpin Computing, Tribble, and S.N.A.K.E. We differentiate our skin from Pushpin Computing and Tribble by instantiating our work on a flexible substrate and adding sensing modalities contained in neither of these projects, and we differentiate our work from S.N.A.K.E. by extending S.N.A.K.E.'s functionality and mechanical robustness.

Chapter 2

Hardware

The previous chapter explained our design decisions, including functional objectives, and basic motivation. In contrast, this chapter focuses on several of our most important implementation decisions. Specifically, we explain component selection and the basic circuitry onboard each of our nodes. For thoroughness, we provide our PCB layout and complete circuit schematics in Appendices C and D, respectively.

At this point in our work, it is necessary to distinguish between two types of nodes on our skin: the master node and the slave node. Our skin is composed of a single master node connected to multiple slave nodes. Slave nodes are primarily designed for sensing and inter-node communication. In contrast, a master node performs relatively few sensing tasks and instead primarily serves as a bridge between our skin and a PC, which, as described in Section 3.2, records and presents our nodes' states. We used almost the same hardware for both master and slave nodes: a master node uses the same PCB as a slave node and a master node's components are a superset of a slave node's components. All but Section 2.9 describe components common to both master and slave nodes; Section 2.9 explains the minor hardware differences between master and slave nodes. Chapter 5 shows the relative effectiveness of each of our sensor choices, while Chapters 6 and 7 thoroughly analyze these results and evaluates the decisions presented in this chapter respectively. For convenience, we conclude this chapter with a table summarizing the current consumption of each major component of our skin.

2.1 Microcontroller

We took great care in selecting our microcontroller. We required a small, low-power device capable of running at high clock speeds and with enough memory to support complex programs. We selected an MSP430 microcontroller for this purpose. Known for its low power consumption and agile wake up [28], Texas Instruments' MSP430 family of microcontrollers is used in a wide variety of sensor network research (networks for health monitoring [25], environmental monitoring [43], etc.). Our specific microcontroller, an MSP430F1611, had several additional enticing features. Most notably, it:

- Supported multiple clocks and had multiple power modes. As described in Section 5.5, this feature allowed us to experiment with various power algorithms and schemes.
- Provided an adequate number of input and output pins. As described in Section 4.3, we require 20 input and output pins just for our peer-to-peer connections. All told, our design uses 49 of our MSP430F1611's 64 pins.
- Had numerous built-in peripherals including and analog-to-digital converter and communication modules.

2.2 Mux

While performing some basic accounting during our project's design phase, we encountered a problem: our sensor count (13) exceeds the number of channels that our ADC supports (8). Therefore, we add a mux to each node which effectively expands our ADC. Because space is at such a premium on our nodes, our primary criteria for mux selection is size. We settled on the ISL84781IRZ (pictured in Figure 2-1) which comes in an ultra-small 16 QFN package. Aside from its small size, the ISL84781IRZ is also attractive for its low power consumption (less than 2 μW) and short switching time of 13 nano-seconds (a figure that ensures our MSP430F1611 running at its fastest

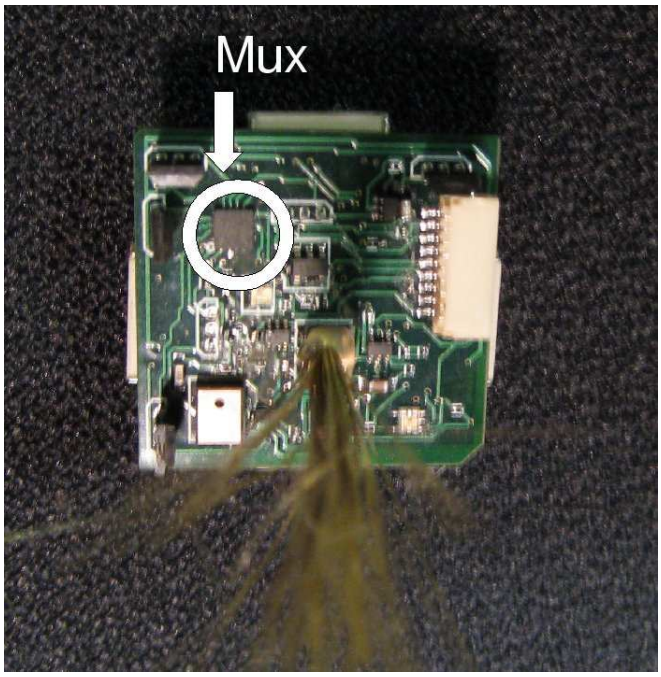

Figure 2-1: Picture of mux on node board.

possible clock rate will be able to read data immediately after setting the mux's control pins) [20]. The output from our mux is passed through a voltage divider, which reduces the mux's output signal by half before it reaches our MSP430F1611's ADC. As will be explained later, the purpose of this divider is to ensure that signals from our Hall effect sensors do not exceed the voltage threshold of our ADC.

2.3 Light Sensor

We used a Toshiba TPS851 photosensor as our skin's light detector. The TPS851, pictured in Figure 2-2, couples a photodiode in a single package with a current multiplier and has several features that specifically recommend it for this project:

- The TPS851 is available in an ultra-small package: Figure 2-2 depicts the sensor on the corner of one of our nodes. Notice how relatively small this sensor is compared to the bulky programming connector above it.
- The sensor comes packaged with both a photodiode and a current multiplier. Therefore, we only require one additional resistor for signal conditioning, thereby

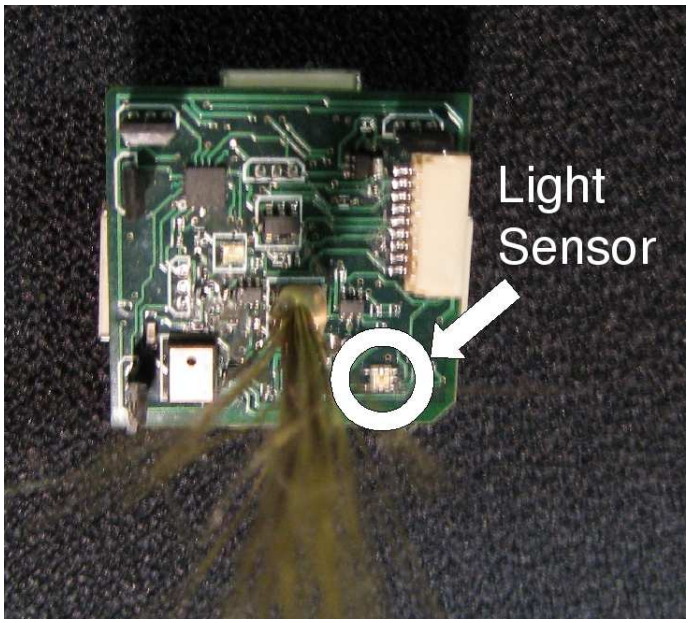

Figure 2-2: Picture of TPS851 light sensor mounted on our board.

reducing component count and, in turn, required board area. In addition, coupling both an amplifier and a photodiode reduces noise that might be introduced from lengthy the potentially lengthy traces that would otherwise connect these two parts.

- The TPS851 comes specifically attuned to detect light that excites the human eye [41]. Therefore, it will perform best in the environments our skin will be presented with.

The output current of our TPS851 light sensor passes through a resistor, creating a voltage which we sample directly from a pin on our microcontroller's analog-to-digital converter. For a thorough treatment of the TPS851's performance on our node, please see Section 5.3.1 which displays our light sensor's responses to different stimuli and Section 6.2 which analyzes those results and discusses the TPS851's overall effectiveness for our project.

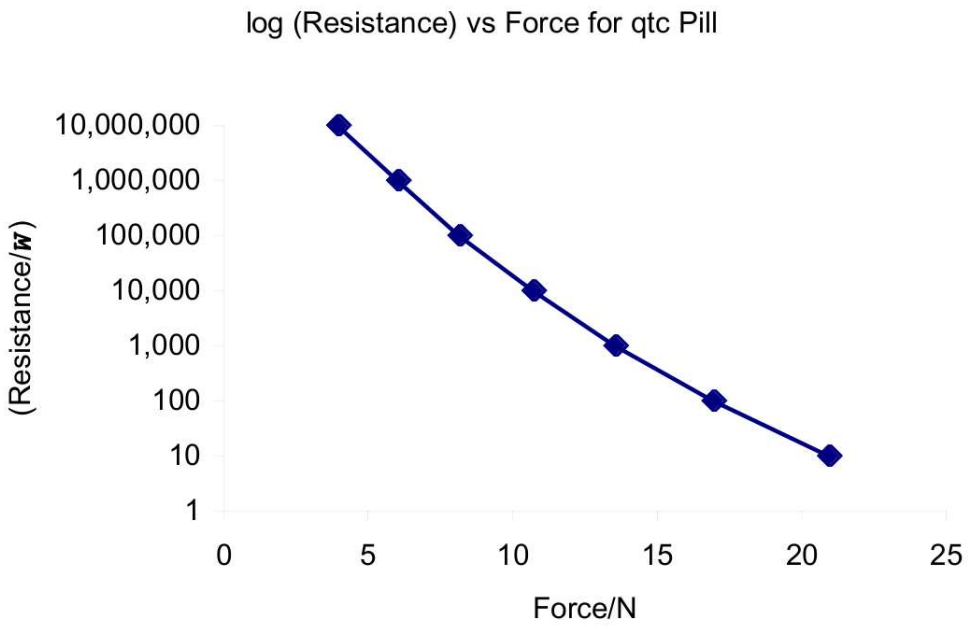

Figure 2-3: QTC pressure versus resistance. Figure from [26]

2.4 Pressure Sensors

Perez used a quantum tunnelling composite (QTC) material for his pressure sensors [33]. Following his lead, we also attempted to use QTC material in the form of a 3.6 mm x 3.6 mm "pill" for pressure sensing. Provided by Peratech, the material's manufacturer, Figure 2-3 (from [26]) plots a QTC pill's resistance against the amount of force applied to it. As can be seen in the figure, QTC material changes its resistance non-linearly as force is applied to it.

We used our QTC sensor in a voltage divider. Connecting the midpoint of our voltage divider to a pin of our MSP430F1611's ADC, we hoped to determine voltage across each pill, and in turn each pill's resistance and therefore the pressure applied. However, we experienced mixed results using QTC pills. Although as shown in Section 5.3.4, we were able to detect some direct pressure events on our skin, in general, the pills were not adequately sensitive to the stimuli of interest. In addition, because they would not bond to solder [26], we resorted to individually gluing pills to each board by hand - a sloppy practice complicated by the glue's interfering with each pill's electrical contact with our node boards. The imprecision of this method made it impossible to

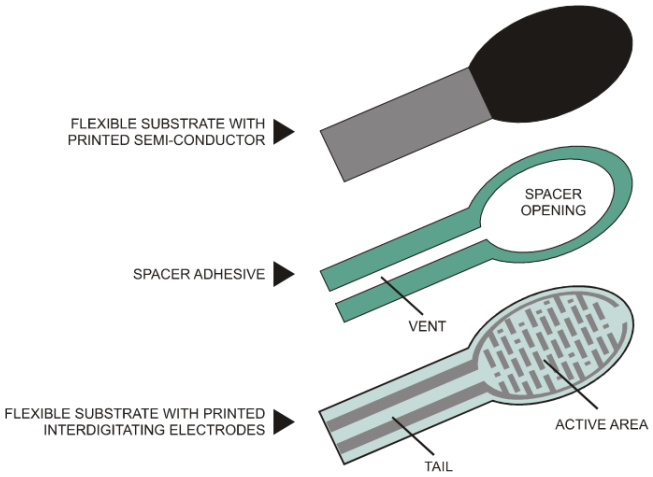

Figure 2-4: Picture of FSR sensor. Figure from [12].

reliably calibrate these QTC sensors, and motivated us to use a pre-packaged force sensitive resistor (FSR) instead.

Similar to QTC material, an FSR changes its resistance as force is applied to it. Therefore, we used the same voltage divider technique described above to track its resistance changes. The primary advantage of using FSRs instead of QTC pills is FSRs' packaging. Pictured in Figure 2-4 (from [12]), our FSR comes with two exposed tin leads. Compared to the ugly operation of gluing QTC pills to our boards, these solderable sensors were easier to mount and therefore more reliable.

To optimally direct pressure stimuli to the most sensitive head of the FSR sensor, we built a standoff system. This system is diagrammed in Figure 2-5, and greatly improved the node's pressure sensing, which is demonstrated in Section 5.3.4 and analyzed in Section 6.6.

2.5 Microphone and Whisker

2.5.1 Microphone

Sound sensitivity is rarely associated with skin, however, we chose to include a microphone on our skin because of:

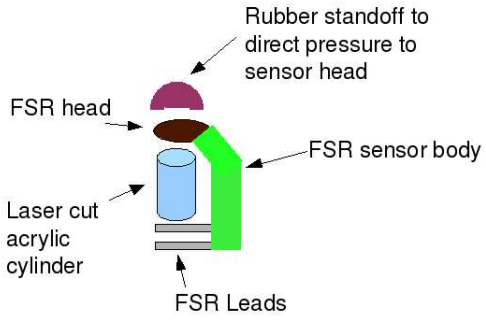

Figure 2-5: Diagram of pressure sensor and standoff mounting system.

1. The application power it gives us: as demonstrated by Lifton, audio data can be used to perform network initialization and locate stimuli [32]. In addition, other authors, such as [22], have demonstrated the usefulness of audio data in categorizing stimuli.

2. The fact that it allows us to push the performance requirements of our system: only the most exotic applications require us to sample our non-sound sensors at the rate necessary for our microphone to record low-quality audio. Therefore, applictions requiring sound sensing and/or processing forces us to push our real-time performance to its limits.

We settled on an SPM0102NE-3 piezo microphone manufactured by Knowles Acoustic. Used in portable devices and cell phones [1], the SPM0102NE-3 is designed to be small and to require little power - both highly appealing characteristics for our project. Despite the care we took in selecting one of the smallest microphones we could find, incorporating sound sensing into our skin was still relatively expensive from a board area perspective: as can be seen in Figure 2-7, the SPM0102NE-3 is by far the largest sensor on our boards, and requires the most sophisticated and largest signal conditioning circuitry of any of our other sensors. Figure 2-8 presents the circuitry used to condition the SPM0102NE-3's output.

The first stage in our audio conditioning, highlighted by the red box in Figure 2-8, is an inverting amplifier modified with an additional capacitor in the feedback path to increase phase margin. We chose an OPA347 operational amplifier manufac-

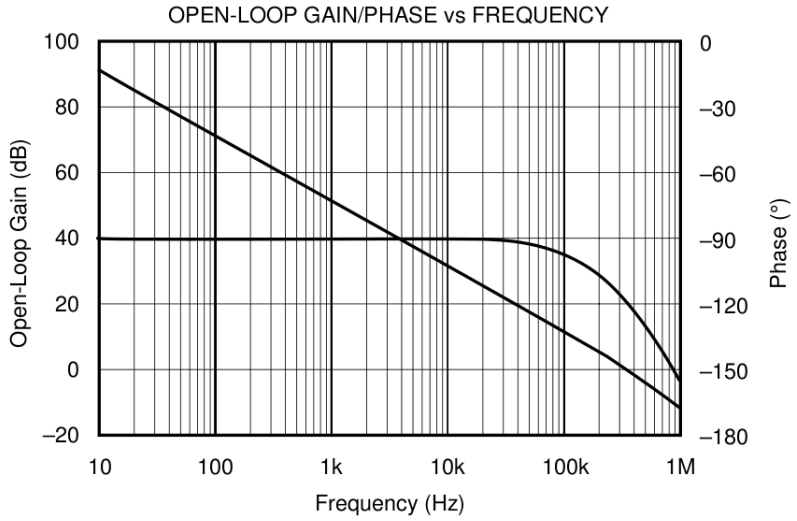

Figure 2-6: The OPA347's transfer function.

tured by Texas Instruments for our circuit because of its small package size, power efficiency, and high degree of linearity. As can be seen from our operational amplifier's transfer function presented in Figure 2-6, the OPA347 does have a noticeable disadvantage though: because its phase decreases even for low frequencies with a slope of approximately 20 dB/decade, there is a limit to the amount of gain we can apply while still expecting stable performance. Our initial circuitry, taken from [33] had a much lower gain. However, this lower gain was not adequately sensitive for our purposes. Therefore, we naively increased the gain of our circuit. Unfortunately, our new configuration had a relatively low 3 dB cutoff frequency of 100 Hz.

The next section of our microphone conditioning circuitry, designated by the middle dashed box in Figure 2-8, is a simple low-pass filter with a cutoff frequency of 4 KHz. Originally incorporated to deal with potential high-frequency noise picked up by the microphone, oscilloscope tests of our microphone's response later showed that there was no noise on the line and such a filter was unnecessary. As the direction-of-arrival sensing benefits from high-frequency information, we disabled this filter. As can be seen from the schematic in Figure 2-8, the output of this filter is fed into the cathode of a diode. This diode functions as a negative peak detector for our sensor - because diodes do not conduct until appropriately biased, only signals greater than

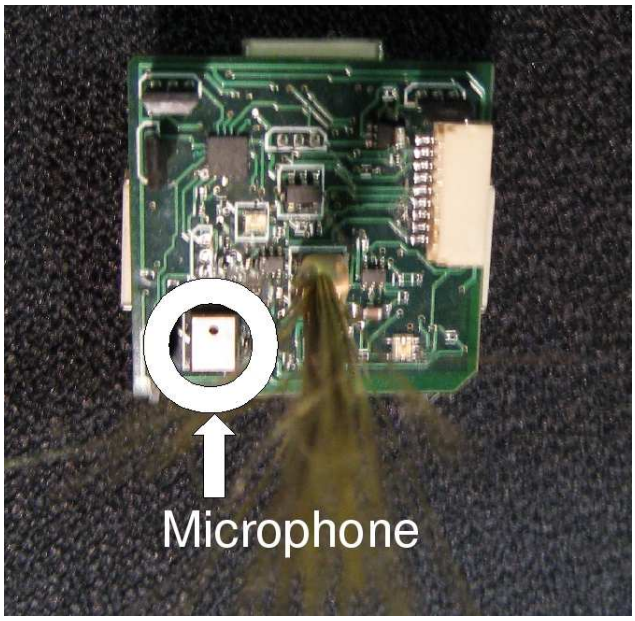

Figure 2-7: Picture of SPM0102NE-3 microphone mounted on our node.

our diode's .37 V forward voltage will be passed to our conditioning circuitry's final stage. The final stage of our conditioning path, designated by the green box in Figure 2-8, is a simple RC envelope follower which extends the length of the audio signal, thereby permitting us to sample below the Nyquist rate and still detect audio events.

Realizing that future uses of our skin may require our MSP430F1611 to record signals both before and after our peak detector, we routed lines for both signals to our ADC. The pre-peak detector signal gets passed through a mux and a voltage divider (for reasons explained in Section 2.7) on its way to our ADC, while the post-peak detector goes directly to our MSP430F1611's ADC. Unfortunately, due to an oversight, we neglected to consider the impedance of our analog-to-digital converter in the design of our circuitry. Therefore, it is difficult to say with exact certainty what the time scale of decay is. For a thorough treatment of the SPM0102NE-3's performance on our node, please see Section 5.3.2 which displays our microphone's response to different stimuli and Section 6.5 which analyzes those results and discusses the SPM0102NE-3's overall effectiveness for our project.

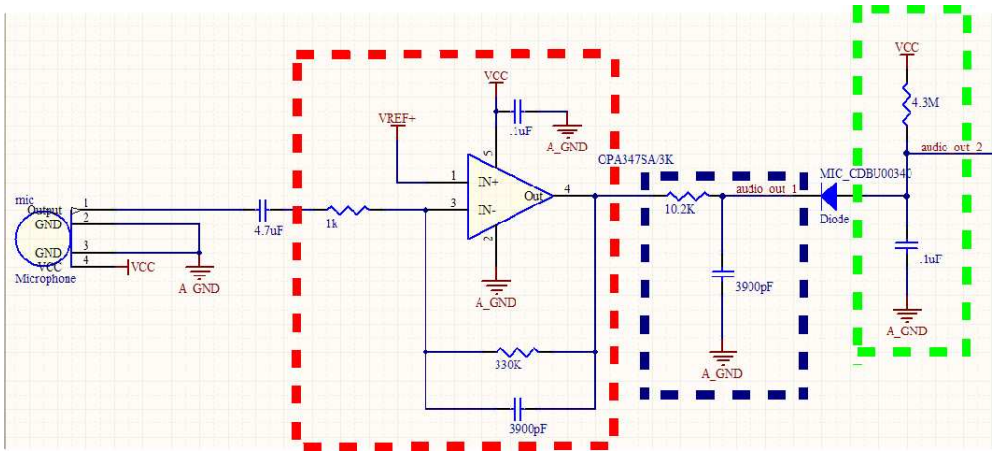

Figure 2-8: Conditioning circuitry for microphone. The left-most dashed box is a basic inverting amplifier with an additional feedback capacitor for stability. The center dashed box is a low-pass filter with a 4 KHz cutoff frequency. The right-most dashed box is an RC envelope follower that extends audio signals.

2.5.2 Whisker Sensor

Identifying the importance of whiskers to some rodents and other species, we made a concerted effort to develop and deploy a whisker sensor on each of our nodes. To this end, we tested and evaluated several different implementations of hair sensors before settling on our current configuration. Most notably, we considered deploying a piezo-film based sensor similar to that described by Lifton [31], but dismissed it because of the amount of board area such a system would necessitate. We additionally examined another piezo based solution, this time using a PKGS shock sensor manufactured by Murata. Gluing bristles to this part and amplifying its output, we were able to detect heavy and direct stimuli. However, this shock sensor-based approach could not detect softer stimuli, and we were once again forced to reconsider. Our final whisker sensor consists of paint brush bristles glued to the surface of an SPM0102NE-3 microphone (the same type of microphone as described in the previous audio sensing section). We experimented with several different glues and epoxies for affixing bristles to the sensor, but settled on hot glue for its ease of application and because it completely blocked audio signals from exciting the SPM0102NE-3's element.

The whisker's signal conditioning circuitry, shown in Figure 2-9, is very similar to the microphone conditioning circuitry described in Section 2-8, with identical peak

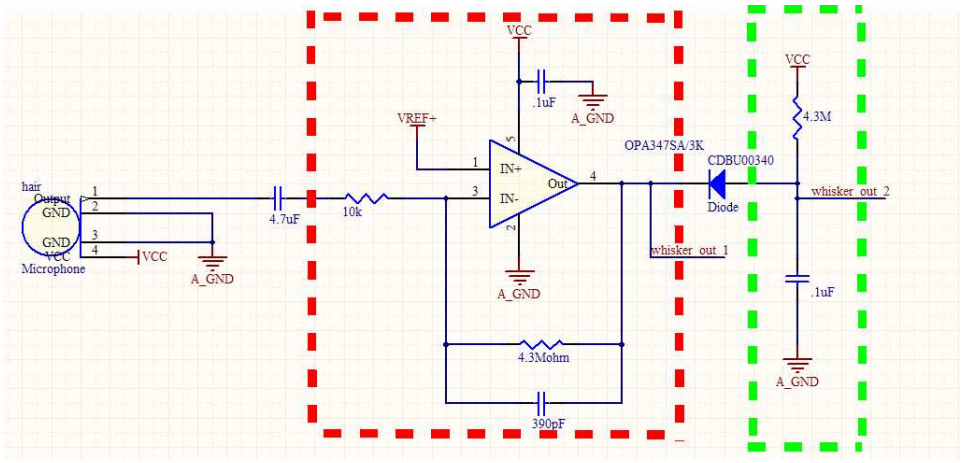

Figure 2-9: Conditioning circuitry for whisker. The dashed red box is a basic inverting amplifier with an additional feedback capacitor for stability. The dashed green box is an RC envelope follower that extends time of whisker signals.

detector and envelope follower stages. There are only three minor differences between the circuits: we perform no low-pass filtering on our whisker sensor's output; our whisker sensor's pre-peak detector signal is routed to a digital, interrupt enableable pin on our MSP430F1611 instead of through a mux to the MSP430F1611's ADC; and our whisker sensor has a much higher gain. A simple Nyquist plot indicated that this higher gain was not enough to push our sensor into instability - a finding confirmed in Section 5.3.3, which presents the response of our whisker sensor to multiple stimuli.

2.6 Temperature Sensor

We used an LMC20CIM for temperature sensing. We were attracted to the LM20CIM for its low power consumption of 7 μA when active and its economic footprint - as seen in Figure 2-10, our temperature sensor takes up very little board space.

Equally attractive, however, was the sensor's simplicity. The LM20CIM requires no additional signal conditioning circuitry, and feeds a voltage value directly into our analog-to-digital converter. According to the part's datasheet, this voltage maps non-linearly to an absolute temperature with +/- 5 ^{o}C accuracy [36]. As will be explained later in Section 6.3, the LM20CIM sensor unfortunately left much to be desired, and we never were able to record temperature with the accuracy that the

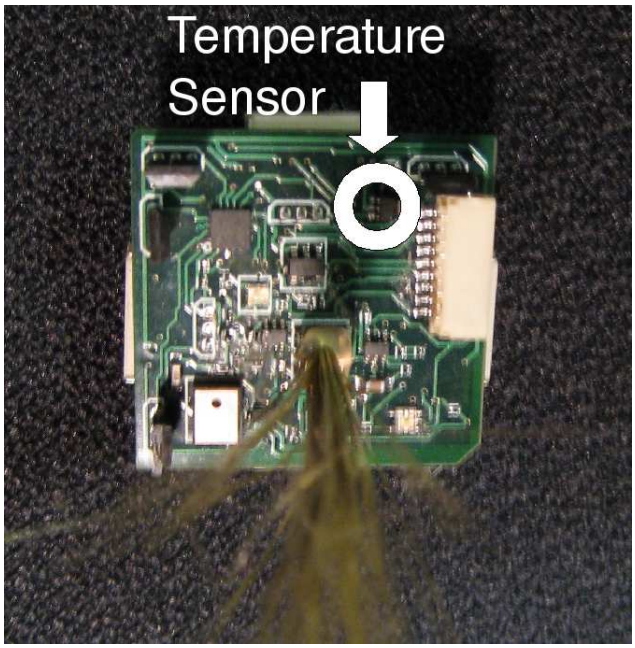

Figure 2-10: LM20CIM temperature sensor on node board.

sensor advertised.

2.7 Hall Effect Sensor

In an effort to deduce the angular positions of nodes relative to each other, we included six linear Hall effect sensors on each node as well as two magnets. As can be seen from Figure 2-11, the West and North sides of each board are populated with the Hall effect sensors while strong, Neodymium magnets are mounted on the South and East sides of each board. The Hall effect sensors on each side of the boards are non-collinear, which theoretically allows differential measurements between the sensors to infer the relative angles between nodes, similar to the work presented in [14]. (Throughout the rest of this work, we designate the Hall effect sensor in the bottom left corner of our node as the West South sensor, the Hall effect sensor in the middle of the left side of our board as the West sensor, the Hall effect sensor on the top left corner of our node facing left as the West North sensor, the Hall effect sensor on the top left corner of our node facing upwards as the North West sensor, the Hall effect sensor in the middle of the top side of our node as the North sensor, and the Hall effect sensor on

the top right corner of our board as the North East sensor.)

The specific theoretical relationship between magnet strength, m; the magnetic latitude, λ; the distance a magnet is placed from a detector, r; and the strength of the magnetic field recorded by a detector, B is given by:

$$B = \frac{\mu_0}{4\pi}\frac{m}{r^3}\sqrt{1 + 3sin^2(\lambda)} \tag{2.1}$$

where μ_0 is the permeability of free space [42]. This equation has no closed-form solution. As such, we wrote a simulation program, detailed in Appendix A, which tested different methods of converting data from our Hall effect sensors into an estimate of a node's neighbor's position. We found that for small angles, θ, the approximation:

$$\theta = \frac{Hall_1 - Hall_2}{Hall_1 + Hall_2} \tag{2.2}$$

As explained in Section 5.4.2, our Hall effect sensors did not perform very well in localization. Some of this may have been due to our sensor selection. We settled on the A1323LUA-T sensor manufactured by Allegro. Unfortunately, this sensor, as well as all other available vertically mounted Hall effect sensors, required a minimum supply voltage of 4.5 V. This level was substantially past the operating points of our microcontroller and all our other sensors. Therefore, we created a separate supply voltage line for these sensors that ran at a higher voltage from the rest of the components on our boards. The outputs of all six of our Hall effect sensors were routed through our mux. To ensure that our Hall effect sensors' outputs did not exceed the input voltage requirements of our ADC, we put a simple voltage divider on the output of our mux.

2.8 Connectors

Our skin is formed from an array of rigid nodes. In order to achieve our goal of imitating the mechanical flexibility of biological skin, we therefore require bendable

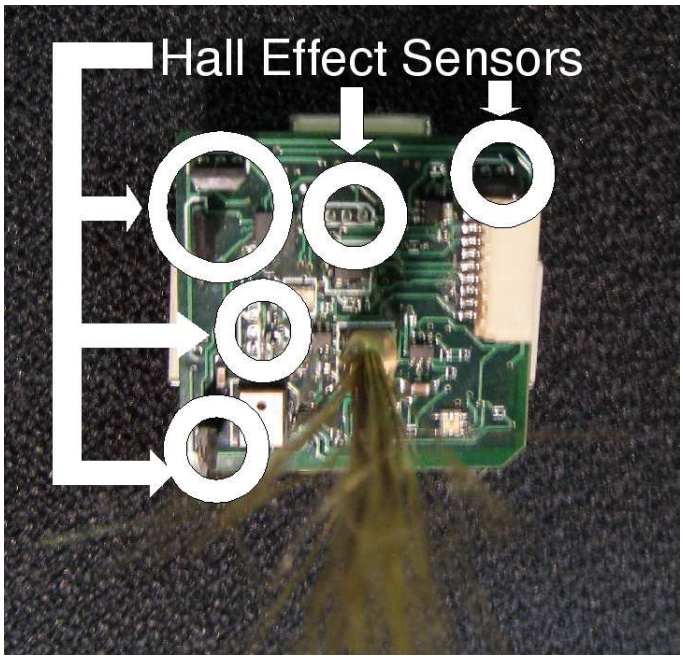

Figure 2-11: Picture of Hall effect sensors and magnets mounted on node board.

connections between our nodes. This poses an interesting challenge. Our connections must satisfy both mechanical and electrical considerations: not only must the interconnects physically hold nodes in place while allowing flexibility, but they must also be conductive and insulated (to prevent catastrophic short circuits). In addition, we must allow for a relatively large number of connections (9), pictured in Figure 2-12. (Later portions of our report, particularly Chapter 4, explain the function of each of these lines.)

We considered numerous solutions to this problem, researching a variety of specialized flectures from the mechanical engineering literature, various conductive polymers, and conductive fabric based approaches similar to [9], while specifically experimenting with springs, flat flex cable, wired interconnects, and conductive thread. However, each of these approaches had specific disadvantages: fabric-based and conductive thread-based interconnects were difficult to fix to our nodes and required a skill set that we did not have; flat flex cable connectors proved much too delicate for any real bending or strain; the steel springs that were available for our project were not insulated and conventional core solder would not bond to the metal, making

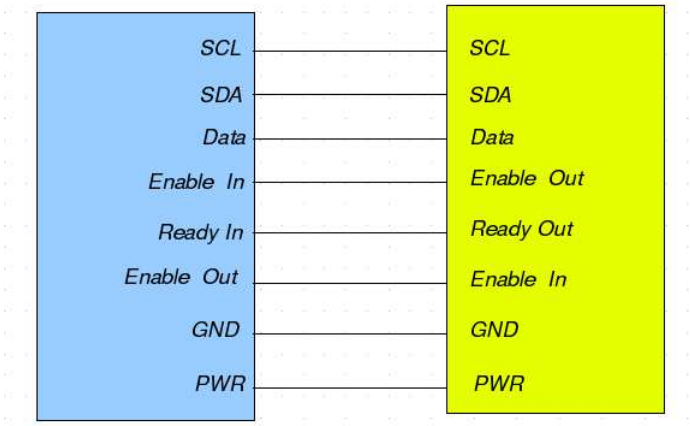

Figure 2-12: Diagram showing internode connections. Chapter 4 explains the function of each of these lines.

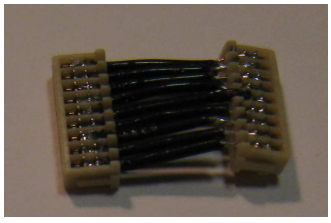

Figure 2-13: Picture of wired connectors selected for linking nodes.

attaching the springs difficult; and wire connectors often snapped and broke when subjected to tension. A thorough exploration of connection types may have provided us with a more flexible prototype, however, it would have significantly distracted from the principle focus of our work. Therefore, we settled upon the solution that most acceptably balanced functionality with ease of prototypability, and settled upon the wired connectors pictured in Figure 2-13. As described in Section 5.1.2, these connectors exhibited mixed results - they permitted reasonable flexibility, but often broke. Section 7.1.2 thoroughly treats alternate solutions future iterations of our skin might exhibit; our nodes are amenable to any kind of interconnect if the connectors are not soldered in, and the new connection scheme (for example flex circuitry, fabric, etc.) is bonded directly to the connectors' PCB holes.

2.9 Master Node

Our skin contains at least one master node, which looks very similar to our slave nodes. In order to support flexibility during experimentation with our skin, master nodes come equipped with all the sensors of a slave node (described in this chapter's previous sections). However, they primarily function as a communication bridge connecting our skin to a PC. To perform this task, our basic slave node requires some minor hardware modification. Chapter 4 presents a thorough and detailed analysis of our skin's communication system. Most important to our current description of hardware is our choice of an I2C backbone. As noted in Chapter 4, the I2C protocol relies on two pull-up resistors, one for a data line and one for a clock line. These resistors are soldered to our master node's PCB.

Additionally, as explained in Section 3.1.2, our master node connects to a PC via an RS-232 line. RS-232 relies on relatively precise timing to synchronize data transmission and reception. Because there is quite a lot of variation in the onboard oscillator of our MSP430F1611 [19], we soldered a precise 8 MHz oscillator to our master node's microcontroller to provide for this precision. Finally, we routed two wires from our microcontroller to a separate bread-boarded circuit containing a MAX 233 chip. This circuit performed voltage level conversion, translating our MSP430F1611's 0 V to 3.3 V signals to our RS-232's +/- 5 V thresholds.

2.10 Power Consumption Summary

Recalling Section 1.1, one of the principle goals in building our skin was to be thrifty with our power usage. Although we tested an algorithmic approach to power consumption (mentioned in Section 5.5), our greatest power savings occur from the selection of efficient components. Table 2.1 summarizes the current consumption of each component we selected. Our LED overwhelmingly requires the most power (although it can be duty-cycled and used sparingly), but as can be seen from the table, most other components are extremely low power.

Name	Part Number	Number on Node	Maximum Current	Minimum or Quiescent Current
Hall Effect	A1322LUA-T	6	8 mA	5 mA
Temperature	LM20CIM7/NOPB	1	7 μA	7 μA
Pressure (QTC)	QTC	Up to 3	3.23 mA	$<$ 1 μA
Pressure (FSR)	FSR	Up to 3	99.4 μA	$<$ 3.3 μA
Light	TPS851	1	620 μA	620 μA
Microcontroller	MSP430F1611	1	2.4 mA	.5 μA
Microphone	SPM0102NE-3	1	250 μA	100 μA
Whisker	SPM0102NE-3	1	250 μA	100 μA
Mux	ISL84781IRZ-ND	1	70 nA	70 nA
RGB LED	598-8710-307F	1	60 mA	0 mA

Table 2.1: Summary of current consumption for the components of each node.

Chapter 3

Software

We designed the electrical system of our nodes to support a wide range of functionality. Our circuitry powers and conditions sensors and connects neighbor nodes and sensor outputs to our microcontroller. This infrastructure, however, brings us only half-way to our goal of building a functioning skin: not only do we need a suitable electrical foundation for our work, but we also require well-written code to control and validate our skin's functionality. The software written for our project can be divided into three primary categories:

1. Software written for design and simulation purposes.
2. Code written to visualize and record our skin's state and data values.
3. Firmware written for control and communication of our skin.

We move our discussion of code written for design and simulation purposes to Appendix A, and focus on our visualization and firmware programs in this chapter (excepting firmware written for communication, which we save save for Chapter 4 because of how strongly informed such work is by electrical considerations). In general, the principles we followed in writing software aligned with our general goals in producing our skin (which are listed in Section 1.1). Specifically, our software was designed for efficiency, speed, flexibility, and robustness. As will be mentioned in later sections, occasionally these goals conflicted and we were forced to sacrifice one desired trait for another.

3.1 Firmware

Firmware is code run by our microcontroller, and therefore governs the most basic functionality of our nodes and skin. In general, our skin was composed of two types of nodes that performed very different tasks: a single master node and numerous slave nodes. We treat each of these types of nodes separately in the following two subsections.

3.1.1 Slave Node Firmware

The program flow for slave nodes was simple and straightforward. Slave nodes perform an initialization, which consists of:

- Setting a clock rate.
- Choosing a power mode.
- Setting and enabling interrupts - both for communication and analog-to-digital conversion of sensor outputs.
- Determining quiescent values for sensors.

We performed several experiments detailed in Section 5.5, in which we modulated clock rates and power modes to analyze how such choices impacted our skin's power consumption. However, for data acquisition, we generally ran our microcontrollers at their fastest clock rates (between 6 MHz and 8 MHz [19]) and in their highest power modes. Although more expensive from a power standpoint, the increased clock and sampling rates afforded by these choices provided much greater sensor resolution.

After initialization, node actions are event-driven: our microcontroller sustains an infinite loop until an event generates an interrupt, which breaks normal program flow. Interrupts on slave nodes took five primary forms:

1. **An interrupt triggered by an expired watchdog timer.** A watchdog timer is built into the architecture of our MSP430F1611 microcontroller. Watchdog

timers increment a register with each clock tick. A built-in system call allows a programmer to reset the watchdog's register to zero. If the watchdog's register ever overflows, the entire node is reset, and program execution is set to restart at the first instruction located in memory. Watchdog timers add to system robustness by reducing unanticipated infinite loops and unusual microcontroller states. We performed several experiments with watchdog timers, incorporating them into our skin during their normal operations. Later iterations of our work excluded watchdogs, as node performance was found to be stable without them, and enabling watchdogs occasionally resulted in correctly functioning nodes' being incorrectly reset.

2. **An interrupt triggered by a communication query from a neighbor.** As will be described in Chapter 4, we designed and implemented a peer-to-peer messaging protocol. Messages passed between nodes may be used to control a wide variety of behavior ranging from trivial (blinking an LED to acknowledge receipt of a message), to more complicated (changing a neighbor's clock and power mode upon receipt of a message, as discussed in Section 5.5). Realizing our firmware needed to be able to support such a wide range, we wrote our code modularly. Rather than modifying our program's entire structure, all that is needed to remove the way a node responds to a message is to delete a single line of code. Similarly, all a programmer must do to radically change the manner in which a node responds to its peer-to-peer messages is to write a modular function specifying new behavior and change a single line of code to direct program flow towards the new function.

3. **An interrupt triggered by a communication query from the master node.** As will be explained in Chapter 4, in addition to the peer-to-peer message protocol briefly described above, we also implemented an I2C bus communication link, which connected every single node on our skin. We relied heavily on our bus, using it as the primary communication line during data acquisition. Interrupts on our I2C bus were generated from the skin's master node request-

ing sensor information from an individual node. Individual slave nodes would then respond to the master's query with a sensor's latest state.

4. **An interrupt triggered by our ADC.** Our MSP430F1611 microcontroller comes equipped with a separate analog-to-digital conversion module. Analog-to-digital conversion is not instantaneous: after an ADC begins conversion, it can take several hundred clock cycles before a digital representation of the analog signal is available. Our ADC triggers an interrupt when conversion of an analog signal has terminated. Therefore, instead of repeatedly querying our ADC to determine whether conversion has terminated - an approach that would be computationally wasteful - program flow is automatically redirected when a converted value is ready. During development, we experimented with various different methods of processing an ADC interrupt: depending on the converted value and which sensor's output it represented, we changed our LED's color, delayed program flow, sent messages to neighbor nodes, etc. After performing minor compression on the converted value (which will be described later in this chapter), the converted sensor values were written into an array. Each position in this array represented the latest reading of a particular sensor in our skin. For instance, the third position in the array represented the output of one of our pressure sensors, the sixth position in the array represented the output of our temperature sensor, etc. The entire array is sequentially written back to the master node when the slave node is queried (as described in the previous paragraph). We then specify which next sensor's analog output should be converted to a digital value by our ADC (including correctly setting mux state if necessary). If the temperature sensor's output has just been converted, then we specify that the light sensor's output should be converted next. If our Northeast Hall effect sensor's output has just been converted, then we specify the Southwest Hall effect sensor's output should be converted next, changing our mux to route our Southwest Hall effect sensor's output to our ADC. We end an ADC interrupt by setting a timer whose function is described in the next

paragraph.

5. **An interrupt triggered by an expired timer.** As described in the previous paragraph, timers are set at the end of ADC interrupts. A timer starts an external clock which counts up from zero to some specified value. When the clock reaches that value, program flow breaks and an interrupt is triggered. This interrupt simply requests our ADC to begin converting an analog value to a digital value. Briefly, we experimented with making this call directly at the end of our ADC interrupt, however, this approach constricted the function of our nodes: while servicing an interrupt, an MSP430 cannot service any other interrupts. Therefore, directly requesting our ADC to begin conversion of another sensor's output immediately after termination of a conversion trapped our program flow: nodes spent the overwhelming majority of their time servicing ADC interrupts, which blocked them from dealing with other critical interrupts that were happening on the nodes. Putting in a timer increased the number of non-ADC interrupts on our skin. We carefully selected the length of our timer to allow our sensors at a minimum to completely refresh their values between queries from a master node.

Generally, interrupts were programmed in at compile time, although, our skin does support dynamically changing interrupts at run time (for instance, a node could turn on or off its ability to receive messages from its neighbors or stop sampling its ADC to go into a low power mode). We did review literature describing different methods for network self-assembly, which described different techniques that would have allowed our nodes to determine their neighbors and therefore how many of their peer-to-peer communication interrupts to set. We wrote code that mimicked some of this self-assembly. However, this work did not allign with the primary thrusts of our project, and due to time considerations, we were not able to thoroughly debug it and test it.

After setting and enabling its interrupts, our nodes go through a sensor initialization phase, which determines the quiescent values of all of our sensors. To find

the quiescent values, we average 64 samples of each sensor while no stimuli occur. In general, division is not a fast operation on microcontrollers. As demonstrated in Section 5.2.1, floating point operations require a substantial amount of time. However, we use a trick to speed our averaging. Noting that in a binary number system, a right bit-shift operation is equivalent to division by two, we simply bit-shift a sum of 64 values right 7 times. Bit-shift operations are very fast and greatly reduce the run time of our initialization phase.

One might wonder why we choose to compute sensors' unstimulated states. Our reasons are two-fold:

1. As exemplified in Section 5.5, occasionally, we want our skin to respond to extraordinary events. Such a goal begs the question of what an "extraordinary event" is. Measuring our sensors' values while our environment is at rest gives us a baseline for defining and distinguishing extraordinary events from only minor perturbances[1]. In addition, as mentioned in Section 5.3.3, we used this baseline to suppress noise on our whisker sensor. Specifically, our node only reported whisker events that significantly exceeded the whisker sensor's quiescent value.

2. Quiescent sensor values also allow us to perform minor data compression. Our analog-to-digital converter provides 12-bit resolution for voltages between 0 V and 2.5 V or voltages between 0 V and 3.3 V. Several of our sensors, however, physically cannot have higher or lower values than those in these ranges. For instance, our whisker and microphone sensors cannot output more than 1.5 V because of their inverted amplifier conditioning circuitry and because they are biased at 1.5 V. Therefore, for several of our sensors, we can save bits while transmitting sensor values by logging a sensor's output relative to its undisturbed state rather than sending its absolute value.

Computing quiescent values of our sensors' outputs concludes our slave node initialization. After initialization occurs, our node enters an infinite loop, relying on the

[1]Indeed, not determining the quiescent values of our sensors, may have contributed to poor results when we attempted to localize stimuli based on acute audio events.

interrupts described above to drive our skin's communicative and sensitive functionality. To summarize, slave nodes on our skin: set clock rates and power modes; set and enable interrupts that perform our primary tasks of recording and transmitting sensor data; sample all of their sensors in the sensors' undisturbed states; and, finally, wait for interrupts that drive our sensing and communication systems.

3.1.2 Master Node Firmware

Our skin only had one master node. Master nodes performed no sensing, but instead were responsible for querying the skin's slave nodes for their sensor values and for transmitting these values via an RS-232 connection to a listening PC. As explained in Section 2.9, because RS-232 requires specific timing conventions, we soldered an 8 MHz oscillator directly to our MSP430F1611. With this oscillator in place, our master node's firmware begins by dividing our clock to allow for a 115,200 baud rate over our RS-232 line. After initializing our clock, we set our master node as a master on our I2C bus (for a thorough treatment of what this means and details on our overall communication systems, please see Chapter 4).

Our master node then queries a single node, A, for its first sensor value. Our master node immediately pipes A's response to our listening PC through the RS-232 circuitry described in Section 2.9. Our master node continues querying A until A sends a pre-defined communication termination character, indicating that our master node has read a full set of sensor values from A. Our master node then proceeds to sample another node on our skin, B, until our master node receives the same pre-defined character that indicates B has completed sending its sensor values. This process continues forever sampling the specified number of nodes, with the master node periodically wrapping its queries around from the final node in the skin back to A.

Instead of using a pre-defined character to indicate the end of a data stream, we could have implemented a design in which our master node sampled each node a specific number of times. Our current design is much more flexible than this alternate version: Individual slave nodes can control the length of their transmissions to our

PC. For instance, if Node *A* only wanted to transmit pressure sensor information to our listening PC, it could selectively send only its pressure sensors' values and then issue a pre-defined termination character which causes the master node to proceed to Node *B*, etc.

Because our skin typically contains only one master node, our design unfortunately evinces a single point of failure. To reduce the potential negative side-effects of such a design, we performed some preliminary experiments, finding that communication occasionally slowed or altogether broke down when our master node would query an unresponsive slave or a slave that was busy servicing an alternate interrupt. To prevent such failures from bringing down our entire skin system, we implemented a safety timer function. If a particular slave node, *A*, does not respond to a master's query after a set period of time, the master node automatically resets, and queries Node *B*.

3.2 Visualization

3.2.1 Overview and Motivation

The previous section described microcontroller code written to enable a master node to transmit sensor values from our skin's slave nodes to a listening PC. This section explains PC code written to receive, process, save, and display these data (a program to which we refer throughout the rest of our work as the "visualization"), focusing most specifically on our visual mappings of sensor values. The importance of such a system is clear, it:

- **Provides a form of validation.** Thus far in our thesis, we have described at length a method for building a system capable of mimicking biological skin's sensing and mechanical properties. However, we have yet to propose a method for evaluating and validating our work. Our visualization provides precisely such a tool. By recording and displaying each node's sensor data across time, our visualization and its logs allow us to assess our skin's functionality and

accuracy.

- **Eases debugging.** During initial phases of development, we could only debug by using an oscilloscope or running a debugger from our IDE on a single node. Providing an additional level of feedback, our visualization system greatly assisted in our debugging efforts.

- **Increases usability.** Although our work currently does not have a well-defined application space, as described in Section 1.1 our skin has been built to be extended. As our project moves away from its initial phases and adapts, it will be useful to provide a layer of abstraction for testing additional functionalities. We anticipate that our visualization will increase our system's overall usability.

3.2.2 Visualization Goals

As described in the previous section, our visualization program forms an important part of our overall project. As such, we treat the design and implementation of our visualization seriously, and specifically lay out a series of goals that should enhance usability. Our visualization should:

1. Be clean and clear.
2. Follow user intuition.
3. Be able to record data.
4. Be fast enough for real-time visualization.
5. Scale elegantly as additional nodes are attached to the skin.

While meeting the strict performance goals of recording data and writing fast, efficient code was non-trivial, we were challenged most by the more aesthetic requirements that our visualization be clear and intuitive. Therefore, in later sections, we focus most of our discussion on these topics, only briefly describing more technical decisions - specifically our choice of graphics toolbox - in Section 3.2.3.

3.2.3 Graphics Toolbox Choice

We experimented with several graphics toolboxes, including OpenCV, Java's Swing API, OpenGL, and Matlab's built-in plotting function. In the end, we chose OpenGL. OpenGL provides a highly flexible, fast, and mature structure for writing our visualization software. OpenGL supports rendering well beyond the bounds of basic drawing and sketching; permits coloring of different hues and intensities; and even supports multiple light sources and shadows. Additionally, OpenGL is compatible with C++, an attractive feature given our comfort with writing code in C++ and the language's speed.

3.2.4 Sensor-Visualization Mapping

In many ways, our visualization assumed the role of transducer - for example, by definition, a temperature sensor senses a modality that has no visual representation: one does not *see* temperature, one feels it. Perhaps there exists some set of Jungian archetypes that map each of our sensed modalities to a unique visual representation. However, a cursory review of the literature however suggests that such a prospect is naive. As such, we wrote an extensible central core of code, building it in such a way that we could easily add, remove, and change sensor representations. This approach permitted rapid prototyping, and, in turn, a type of iterated design, allowing us to receive and respond quickly to feedback from helpful colleagues. As such, our initial visualization differed greatly from our initial efforts. Whereas our first visualization attempts relied heavily upon a strange and non-intuitive series of polygons rotating at different rates in different directions with different numbers of vertices, our later attempts, described in the remainder of this section, were more sophisticated and were interpretted more naturally. This work was informed by a simpler set of animated sensor mappings used in the Responsive Environment Group's Tricorder project, which developed a mobile, visual augmented reality browser for sensor networks [23].

To review, each of our nodes contains up to 13 sensors, which we aggregate to give us data on:

1. The intensity of light hitting the node.
2. The temperature of the node's surrounding environment.
3. The amount of pressure placed on our node.
4. The sound produced from stimuli.
5. The position of a node's northern neighbor.
6. The position of a node's western neighbor.
7. The amount a stimulus disturbs our whisker/hair sensor.

We treat each in order, beginning with the intensity of light detected by our TPS851 sensor. We map the intensity of light to a series of yellow lines painted radially outwards from the center of each visualized node. The length of these lines indicate the relative amount of light each node is receiving: in the dark, lines are very short; in direct sunlight, lines are very long. Figure 3-1 shows an example of a single node in low light conditions and high light conditions. Notice that in high light conditions, the yellow lines are much longer than in low light conditions.

We visualize temperature using a horizontal bar (highlighted with an arrow in Figure 3-2). High temperatures increase the size of the bar, and turn the bar an intense red. Low temperatures shorten the length of the bar, and turn it blue. Figure 3-2 shows an example of a single node in a low temperature environment and in a high temperature environment. Notice that in the high temperature environment, the horizontal bar is much wider and redder than in the low temperature environment.

Pressure data from our FSR are visualized by a single pentagon near the center of each node (indicated with an arrow in Figure 3-3). With no pressure applied, the pentagon is static and dark blue. As pressure increases, the pentagon goes from blue to red and begins rotating at a rate proportional to the intensity of the pressure applied to our FSR. We suppressed pressure data from our two QTC pressure sensors: as will be noted in Section 3-3, our QTC sensors did not perform as reliably as we would have liked. Therefore, we suppressed information on each node's QTC sensors,

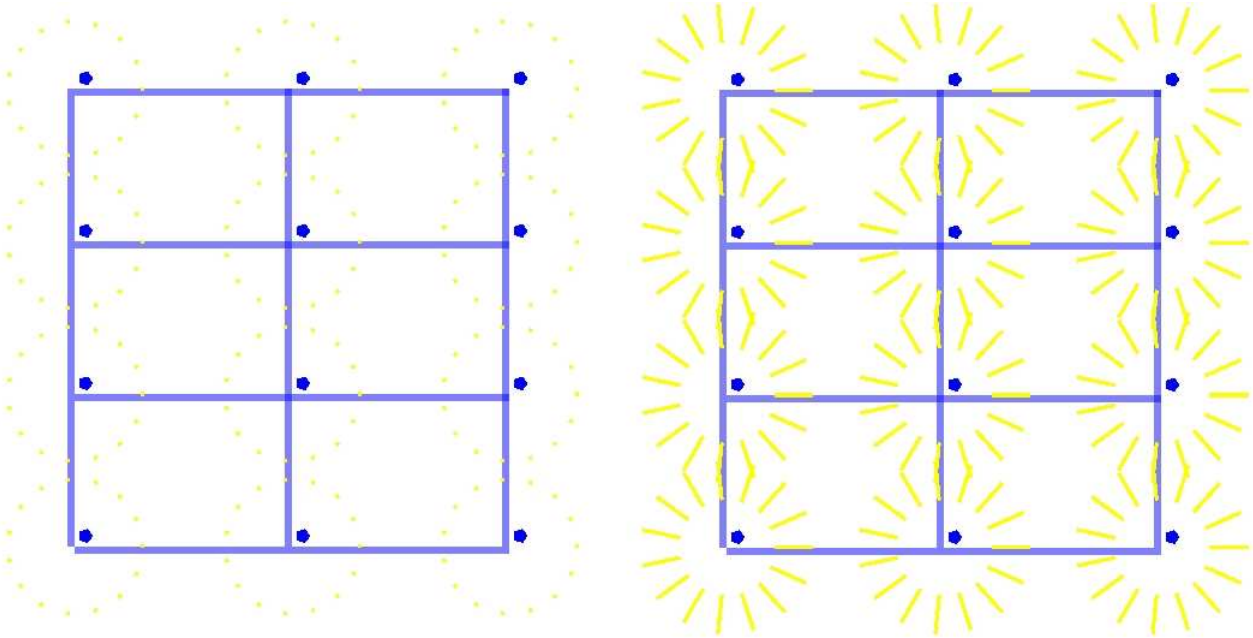

Figure 3-1: Light sensor visualization. Left grid shows visualization for nodes in a dark environment. Right grid shows nodes exposed to a major light stimulus. Notice the change in length of the lines emanating from each node on the grid.

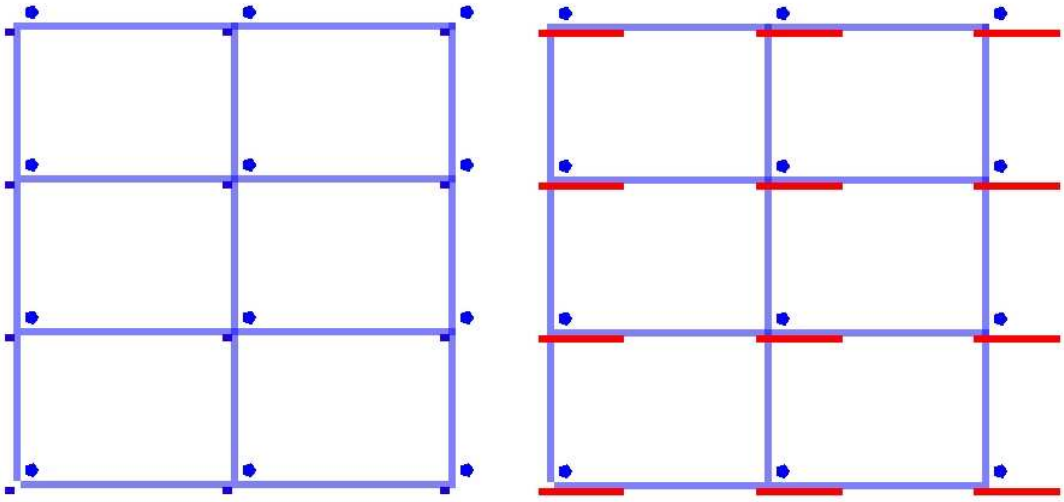

Figure 3-2: Temperature sensor visualization. Left grid shows visualization for nodes in a cold environment. Right grid shows nodes in a very warm environment. Notice the change in length of the horizontal bar for each node.

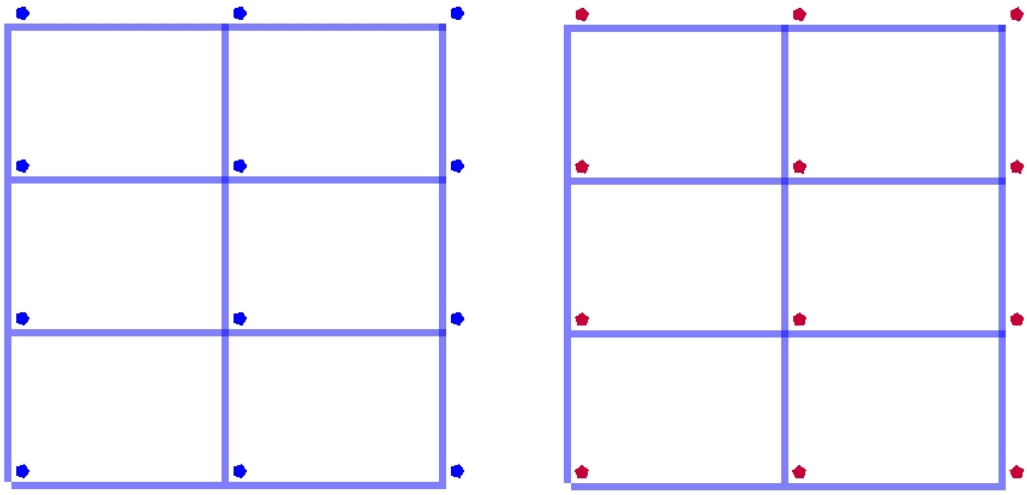

Figure 3-3: Pressure sensor visualization. Left grid shows visualization for nodes minor pressure stimulus. Right grid shows nodes with major pressure stimulus. Pressure is visualized by small pentagons in the figure. Small pentagons change color from blue to red as pressure increases. In addition, pentagons rotate faster as pressure increases.

deciding that the minor additional information provided by our QTC sensors did not justify the clutter they created in our visualization.

We change both the color and number of concentric circles surrounding our nodes to display sound data. As can be seen in Figure 3-4, intense sounds increase the redness of and the number of concentric circles surrounding our node; soft sounds decrease the number of concentric circles surrounding our node and make them less visible.

We assume that our skin is connected in a grid pattern. As neighbor nodes move, we bend our visualization's grid to follow them. Figure 3-5 provides an example of such bending.

Responses recorded on our whisker sensor proved difficult to represent. After several false starts, we settled upon "trembling" each node when our whisker sensor is excited. Trembling consists of rapidly perturbing a node from its normal position. The amplitude of the perturbation is governed by the intensity of the whisker event. An intense whisker event causes a node to move far away from its grid position; a light whisker event barely moves our node from its grid position. To avoid confusing

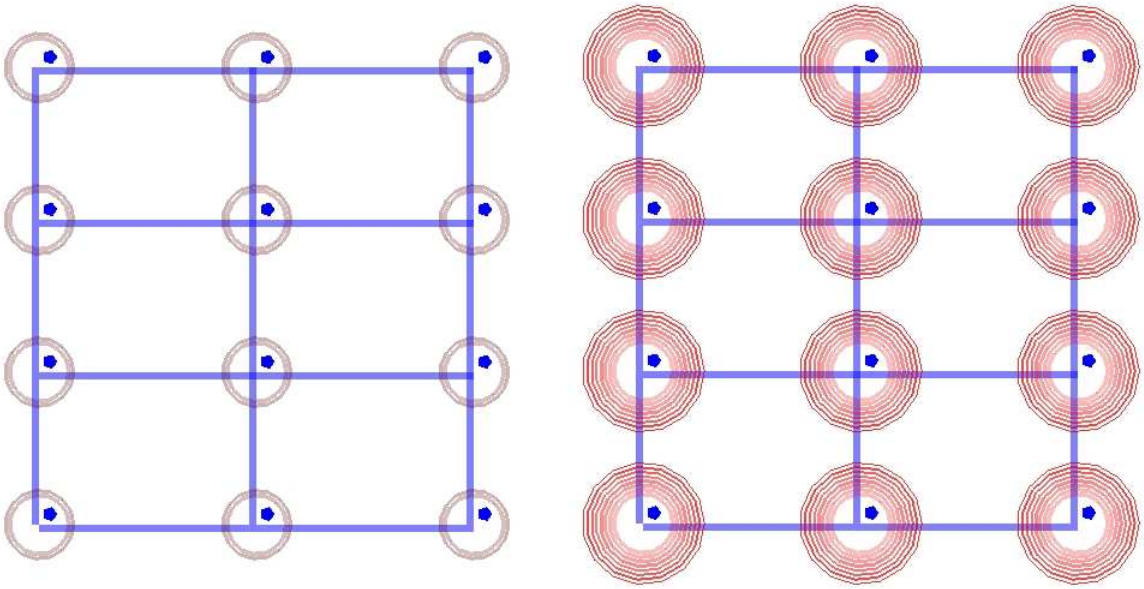

Figure 3-4: Microphone visualization. Left grid shows visualization for nodes with minor audio stimulus. Right grid shows skin undergoing major audio stimulus. Notice the change in the number of concentric circles emanating from each node.

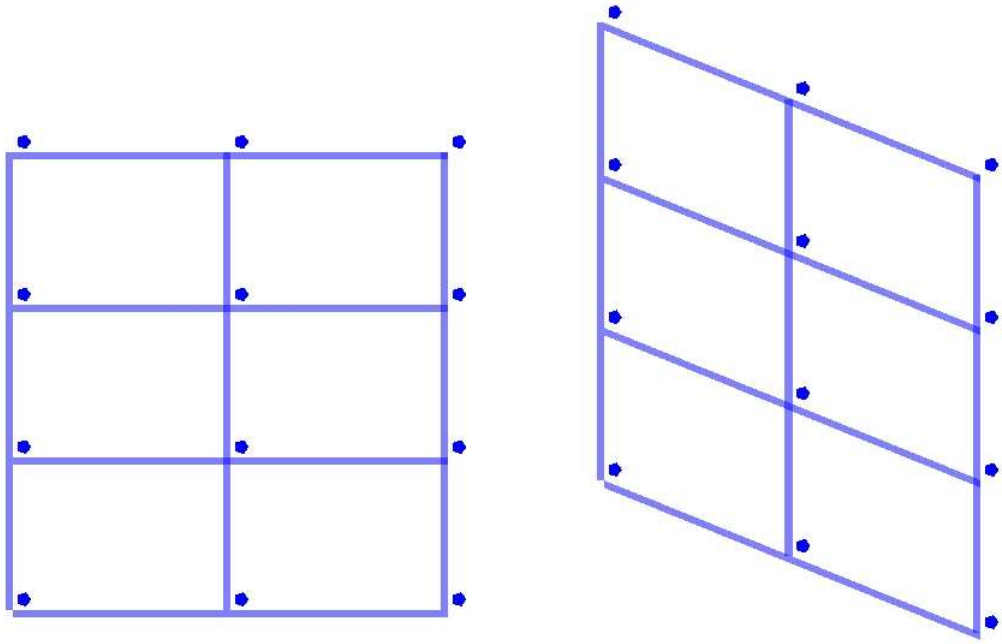

Figure 3-5: Bend sensor visualization. Grid on left shows no bend in skin. Grid on left shows skin as it is uniformly deformed.

our whisker sensor's tremble with the bending from neighboring node positions, we set our maximum trembling to a relatively small value.

Chapter 4

Communication

4.1 Motivation

Our experiments showed the MSP430F1611 microcontroller that we chose is able to perform 12-bit analog-to-digital conversion at a rate of roughly 32 KHz. To put these numbers into context, consider Leo Tolstoy's *War and Peace*. Famous as some of the finest literature ever written, *War and Peace* is also infamous as one of the longest novels ever written: the primary listing of the English translation of the work on Amazon.com contains over 1200 pages and a little over 3.1 million characters. At the sampling rate described above, every ten seconds, each node in our skin has the potential to generate more information than is contained in the entirety of Tolstoy's work. In short, we are not dealing with a stream of information - we are dealing with a flood.

While earlier Chapter 2 of our thesis described infrastructure developed to enable such blazingly-fast data collection, thus far we have not dealt directly with what to do with all the information.

In general, we have three options:

1. Store the data.
2. Quickly process and delete data.
3. Transmit the data to a unit that capable of storing, presenting, or using it.

Examining the first option of storing the data, we immediately find a problem: each node's microcontroller only has 48 KB of flash memory available and physical space on each PCB is at too much of a premium to mount additional storage. Therefore, unless we sample a slow enough stimulus to allow us to significantly throttle the sensor sampling rate, this strategy proves impractical.

Depending on the application, the second option of performing local, real-time computations and discarding excess data can be quite powerful and quite complicated. As an illustrative example of such a strategy, consider configuring a node as a burglar alarm. As an alarm, a node is solely interested in sudden changes in its sensors' states: instead of storing all generated data, a node compares every new sensor reading against the sensor's undisturbed state, and discards all other information as irrelevant. There are enormous benefits to local, online extraction of relevance from stimuli in a distributed network - such a strategy greatly reduces the need for on-board storage and communication. Although the time scale of this project did not allow us to examine this problem in any substantial way, the overall system has been designed to support further inquiries in this direction, as will be discussed later in Section 7.

The third option of quickly transmitting collected sensor readings is the focus of this current chapter. As noted throughout the literature, communication schemes form a considerable part of the sensor network cannon [2]. As noted by various authors, our choice of communication protocols, greatly impacts important network features such as architecture, physical topology, and general operation[18]. To demonstrate functionality, extend usability, and enable exploration, we designed our skin to support two separate modes of communication: an I2C bus and a custom-built peer-to-peer communication scheme. The remainder of this chapter provides an overview of each of these protocols' basic function and implementation as well as their advantages and disadvantages.

4.2 I2C Bus

4.2.1 Overview

Initially devised by Philips Electronics in 1992, the I2C bus protocol has undergone two major revisions since its initial release date. The current standard, set in January of 2000, thoroughly outlines a robust, multi-master, multi-slave communication protocol. As opposed to other protocols, such as SPI and RS-232, I2C requires only two connecting wires between communicants - a clock line (SCL) and a data line (SDA)[1].

Communicants each have a unique address and and can be set in either a master mode or a slave mode. As implied by the existence of the clock line, the protocol is synchronous. Masters are responsible for setting the clock on the SCL line and initiate all read and write operations between communicants by querying slaves. Sent along the SDA line, a query has three parts: an initial start bit, the seven-bit or ten-bit address of the slave a master is querying, and a final bit representing whether the master is writing ('0') or reading ('1') from the slave [37][2].

Slaves respond to the master query by sending an acknowledge bit across the SDA line. If the master has sent a read query, the slave follows its acknowledge bit by sending a byte of data to the master which responds with a single acknowledge bit after receipt of the entire byte. In contrast, if the master has sent a write query, the master transmits a byte of information to the slave, and the slave sends back a single-bit acknowledge response after receiving the entire byte [37].

As seen in Figure 4-1, both the SDA and SCL lines are set by pullup resistors. A master or a slave, sets a logical '0' on the SDA line by sinking current and a logical '1' on the SDA line by setting its input to have an effectively infinite impedance.

[1] In addition, because the protocol needs pullup resistors, as will be described later, a common ground and perhaps a common source voltage are also recommended.

[2] In addition to specifically querying one slave, a master can also send out a "general call", which transmits simultaneously to every slave. When a master performs a general call, the query described here takes a different form, containing a "general call address" instead of a specific slave's address [37].

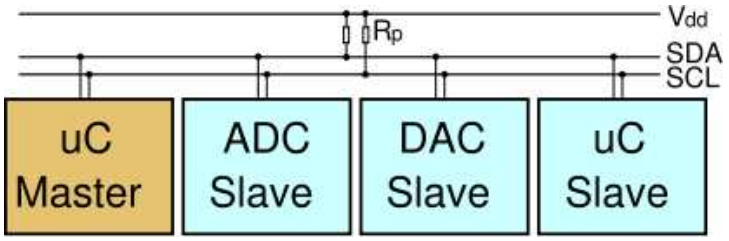

Figure 4-1: A simplified diagram of the I2C's electrical connections. (Image from [11])

4.2.2 Advantages of I2C Bus

Two-wire Setup

As mentioned in Section 4.2.1, I2C's two connecting wires between communicants are fewer than other communication schemes such as SPI (3 minimum) and RS-232 (3 minimum). This small number of connections makes the I2C bus an attractive option for our skin: recalling that each node has four neighbors, each additional wire necessary for communication between boards adds four additional points of connection per board. The increased complexity in routing additional connections, coupled with the added physical space these connections require, has a detrimental impact on our project goal of reducing each node's package size. Less intuitive, but equally immediate is the value the I2C protocol's minimalistic connection count has on another of our design goals: flexibility. While additional connections have no impact on shear movements between nodes, they greatly curtail torsional freedom between the nodes.

Dynamic Multi-master Multi-slave

Real-world applications of sensors and sensor networks such as health monitoring [6] or meshes created by portable devices [17] often do not have the benefit of assuming a static network. Nodes physically enter and leave the network, fail, re-orient, and change modes in such a way that an adaptable (rather than fixed) communication infrastructure can often greatly increase efficiency and function [13].

The I2C communication protocol allows easy transition from a node's being in a slave state to a node's being in a master state, thereby enabling exploration of

self-configuring networks and ad hoc algorithms. In addition, such a feature greatly adds to the robustness of the overall skin. While our network will not face many of the conditions that require adaptable communication schemes, it is certainly possible for a connector to break or a single node to go offline. If the network detects that a master has left the network (through a timeout, a master's transmitting an exit message, etc.) any slave node in the network can easily and quickly replace it.

Built-in on Chip

The MSP430F1611 microcontroller that we selected has a dedicated, on-board I2C module. Relying on this module reduces component count and, in turn, area. Additionally, choosing a communication protocol specifically embedded into our microcontroller greatly simplifies network configuration and maintenance: setup, reads, and writes are all well-defined and well-documented register operations. Further, the MSP430F1611 dedicates a specific interrupt vector to I2C communications. This interrupt vector allows us to still listen for and awaken to I2C events, even while the microcontroller is in its lowest power modes.

Other: Transfer Rate and Precision

The MSP430F1611 microcontroller that we chose supports "fast" I2C mode [19], allowing transfer rates of up to 400 KBits per second [37]. Our skin nodes have the ability to collect much more information than 400 KBits of data per second; however, given the lengthy time scale of sensor stimuli of interest, this rate should still allow a skin composed of over a dozen nodes to capture important events.

4.2.3 Disadvantages of the I2C Bus

While the I2C backbone provides a powerful transmission line, it does have at least two severe limitations that stem from the bus' being shared by every node in the network:

- The I2C bus is not elegantly scalable.

- The I2C bus exhibits a single point of failure.

Robustness - Single Point of Failure

I2C technology, in some ways, should be thought of as non-robust: it exhibits a single point of failure. As described in Section 4.2.1, *all* bits in the network are transmitted across a single data line. As such, a single malfunctioning node can corrupt or disable all I2C communications. Simple tests with the bus while debugging identified this problem. When two nodes were accidentally set to the same address, instead of receiving a character 'a', each received an incorrect character.

Scalability

Electrical

Scalability concerns are both electrical and algorithmic. As described, the I2C protocol's electrical limitations may not be readily apparent, however a more careful examination of the protocol shows where problems may arise. Recall from Figure 4-1 that low-to-high transitions on both the clock and data lines' values are set by pullup resistors. In a platonic realization of this circuit, this transition is instantaneous; in real-life, parasitic capacitances will delay and distort this transition, obeying a first-order linear differential equation with time constant, τ, equal to total resistance, $R_{total} \times C_{parasitic}$.

Therefore, the degree to which parasitic capacitance will affect communication varies depending on:

1. The thresholding values used by the protocol to define logical '0'-s and '1'-s.

2. The rate at which the bus is run. If τ is very small compared to the period of the clock, parasitic capacitance should have negligible effects on communication. However, if τ is large, long settling times may corrupt information transfer.

3. The amount of parasitic capacitance in the circuit. The time constant defined above is directly proportional to $C_{parasitic}$. Increasing the parasitic capacitance

of the circuit should therefore lengthen our time constant, potentially impacting communication accuracy.

Defined strictly by our MSP430F1611's hardware, we have no control over the values our I2C protocol uses to designate logical '0'-s and '1'-s.

Capacitances add in parallel. Therefore, each additional node we add to our skin increases $C_{parasitic}$ and, in turn, τ. This delay penalty leaves us with the unpleasant necessity of having to throttle our I2C protocol as we add nodes to our skin[3].

Scalability

Electrical concerns are not the only scalability problem we encounter in deploying an I2C backbone. Consider four, distinct nodes, A, B, C, and D. A has a message to transmit to B and C has a message to transmit to D. If all four nodes are on the same bus, C will have to wait for A to transmit its message to B before being able to transmit to D. At first blush, such a delay seems trivial, however, a simple toy example indicates how substantially the singleness of the I2C communication channel can effect overall network performance for certain skin configurations.

Assume every other node on a grid is instructed to send an 8-bit character message to its right-hand neighbor. If we use the I2C bus exclusively, even running at 400 Kbits per second fast mode, we soon encounter a problem. Because nodes cannot simultaneously transmit, we get the constraining equation:

$$\frac{\text{Num}_{nodes}}{2} \times \frac{\text{Num chars each transmitting node sends}}{\text{second}} \times \frac{\text{8-bits}}{\text{char}} = 400 \text{ Kbits} \tag{4.1}$$

[3]One might wonder why we solely focus on stray capacitances while ignoring R_{total}. The answer is that the MSP430F1611's microcontroller is only able to source and sink a prescribed amount of current [19]. Reducing R_{total} significantly would therefore overwhelm our microcontroller.

Equation 4.2.3 illustrates a fundamental tradeoff between the size of the network and its performance and motivates the need for an additional communication scheme.

4.3 Asynchronous Peer-to-peer Communication

4.3.1 Motivation

Peer-to-peer communication addresses the two primary problems with the I2C bus mentioned in Section 4.2.3. A single node's failure cannot disable communication across the entire network. And, peer-to-peer communication helps to preserve scalability: for the example described in Section 4.2.3, a peer-to-peer protocol allows us to scale our network with no degradation in performance[4]. Further, peer-to-peer communication permits us to focus on local stimuli, allowing small patches of our skin to locally process data without loading the I2C backbone.

In addition to the above selling points, peer-to-peer communication permits greater experimental flexibility. A master-slave protocol, such as the I2C bus implemented in our skin promotes only specific and hierarchical architectures and control mechanisms. Including a peer-to-peer protocol allows us the flexibility to research topics such as information diffusion, agent-based control, and networking algorithms that would be unapproachable using the I2C bus format.

Finally, our peer-to-peer protocol increases our skin's overall robustness. It serves as a redundant channel for communication in case the I2C line is broken or overly loaded. In addition, it allows us to message nodes with bad I2C connections, or shut them down/circumvent them if they are degrading our I2C line.

[4]In general, this is a simple example. More sophisticated questions that relax assumptions about which nodes need to communicate and the fixed topology of the network, pose very complicated routing problems.

4.3.2 Implementation

Design Goals

Electrical peer-to-peer communication is a well-worn problem in electrical engineering. Existing at least since the days of the telegraph in the early 1800s [21], numerous protocols have been devised and implemented throughout the years to suit various purposes. To select among these schemes, we followed several design principles:

1. **Fast.** Because we do not have a dedicated peer-to-peer communication module into our microcontroller, it may be naive to hope for a data transmission rate approaching the 400 KBits per second achieved by our I2C line. However, we would like our peer-to-peer protocol to run as quickly as possible.

2. **Asynchronous.** Subjectively, an asynchronous protocol is both more algorithmically challenging and aesthetically pleasing than a synchronous line: it reduces our required assumptions and puts fewer restrictions on the network's future applications.

3. **Efficient.** The MSP430F1611 that we chose is powerful for what it is - a cost-effective microcontroller. However, it is by no means a super-computer. Each instruction is precious and takes time[5]. We must take care that our protocol does not overwhelm other operations on the processor.

4. **Adaptable.** Our skin is intended to support a wide variety of functionality and experimentation. Therefore, it is essential that our communication protocol be flexible enough to be effective in a wide-range of tests and situations.

5. **Flexible.** As noted in Section 4.2.2, each additional wire connecting nodes reduces mechanical flexibility. Therefore, we attempt to keep connection count to a minimum.

[5]Cursory tests showed that it took our microcontroller running at its fastest rate over 14.4 seconds to perform 10,000 square root operations - orders of magnitude longer than a modern laptop.

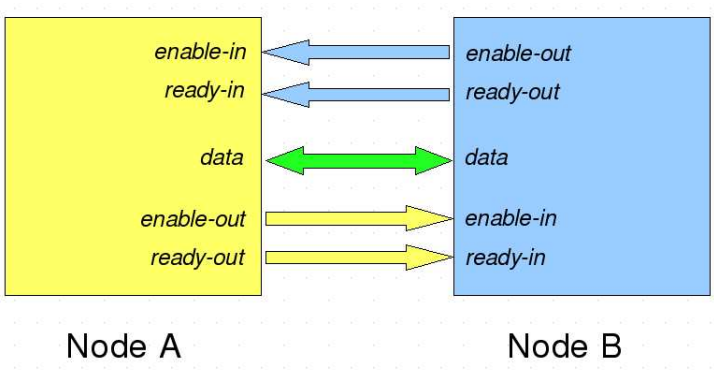

Figure 4-2: Pin connections between nodes for peer-to-peer communication.

Realization

For our peer-to-peer protocol, we employed a five-wire ready-enable scheme. Each node has a "ready-out", "enable-out", "ready-in", "enable-in", and "data" line for each neighbor. Consider two nodes, A and B. As their names imply, the ready-out and enable-out lines are outputs for a single node, A, and are connected as inputs to the ready-in and enable-in lines of its neighbor, B. Similarly, the ready-out and enable-out lines of B are connected to the ready-in and enable-in lines of A. A's and B's data lines are connected directly to each other and serve as either inputs or outputs depending on the direction of communication. These connections are detailed in Figure 4-2.

After initialization, each node's ready-out line is high, its enable-out line is low, and its data line is set as an input. In addition, A and B place an interrupt condition on each of their ready-in and enable-in lines.

If A wants to communicate to B, A first checks if B is ready to receive information by checking its ready-in line (B's ready-out line). If A's ready-in line is low, B is **not ready** to receive information, and the message to be transmitted is put in a queue of messages to be transmitted when B is **ready** to receive. If A's ready-in line is high, B is ready to receive information from A, and A begins transmission. Transmission begins by A's:

1. Changing its ready-out line from high to low.

2. Setting its data line as an output, putting on the data line either a logical '1' (data line goes high) or a logical '0' (data line goes low).

3. Changing its enable-out line from low to high.

4. Sets an interrupt on its ready-in line to detect changes from low-to-high.

B detects as an interrupt a transition on its enable-in line, indicating that A has begun a transmission. B then:

1. Changes its ready-in state from high to low.

2. Sets a global variable on its microcontroller that blocks B from initiating a transfer with A. (All messages B attempts to send while A is transmitting are put on a queue to be dealt with after A's transmission has terminated.)

3. Reads and stores the information on the data line. (The enable-in line of each node should be construed as a validity check: if enable-in is high, information on the data line is ready for reading; if enable-in is low, information on the data line should not be read.)

4. Increments a count of the number of bits received by A.

    - If the count is equal to the pre-designated message length, B knows that it has received the full message from A, and B:
        - Unblocks the variable preventing communication from it to A.
        - Resets its counter of bits received from A to zero.
        - Resets the interrupt generated by its enable-in line's transition in order to be ready to read further interrupts on the line.
        - Checks whether the queue of messages B had attempted to send while receiving from A has any messages. If the queue does contain messages, B begins to transmit them to A. If the queue is empty, B sets its ready-in line to high, indicating that it is ready to receive any new messages.
        - Calls a specified service-routine to react to the message A transmits.

- If the count is less than the pre-designated message length, B knows that it still has more bits to receive from A and:
  - Resets the interrupt generated by its enable-in line's transition in order to be ready to read further interrupts on the line.
  - Sets its ready-in line to high, indicating that it is ready to receive the next bit of the message from A.

The low-to-high transition on B's ready-in-line engenders an interrupt on A. If A has transmitted the entirety of its message, A:

1. Sets its enable-out line low.
2. Changes the data line from an output line to an input line.
3. Sets its ready-out line high.
4. Checks to see if A has any additional messages to send. If A does have additional messages to send, it begins sending them.

If A has not transmitted the entirety of its message, then A:

1. Sets its enable-out line low.
2. Loads the next bit of data onto the data line.
3. Sets the enable-out line high.

Disadvantages of Design

Our peer-to-peer system supports most, but not all, of the design goals laid out in Section 4.3.2. While Section 5.2 demonstrates our protocol's speed and efficiency, requiring a full five wires does degrade mechanical flexibility. We did attempt to reduce the number of connecting wires between nodes by making the system half-duplex, meaning that two connected nodes can communicate to each other, but not simultaneously.

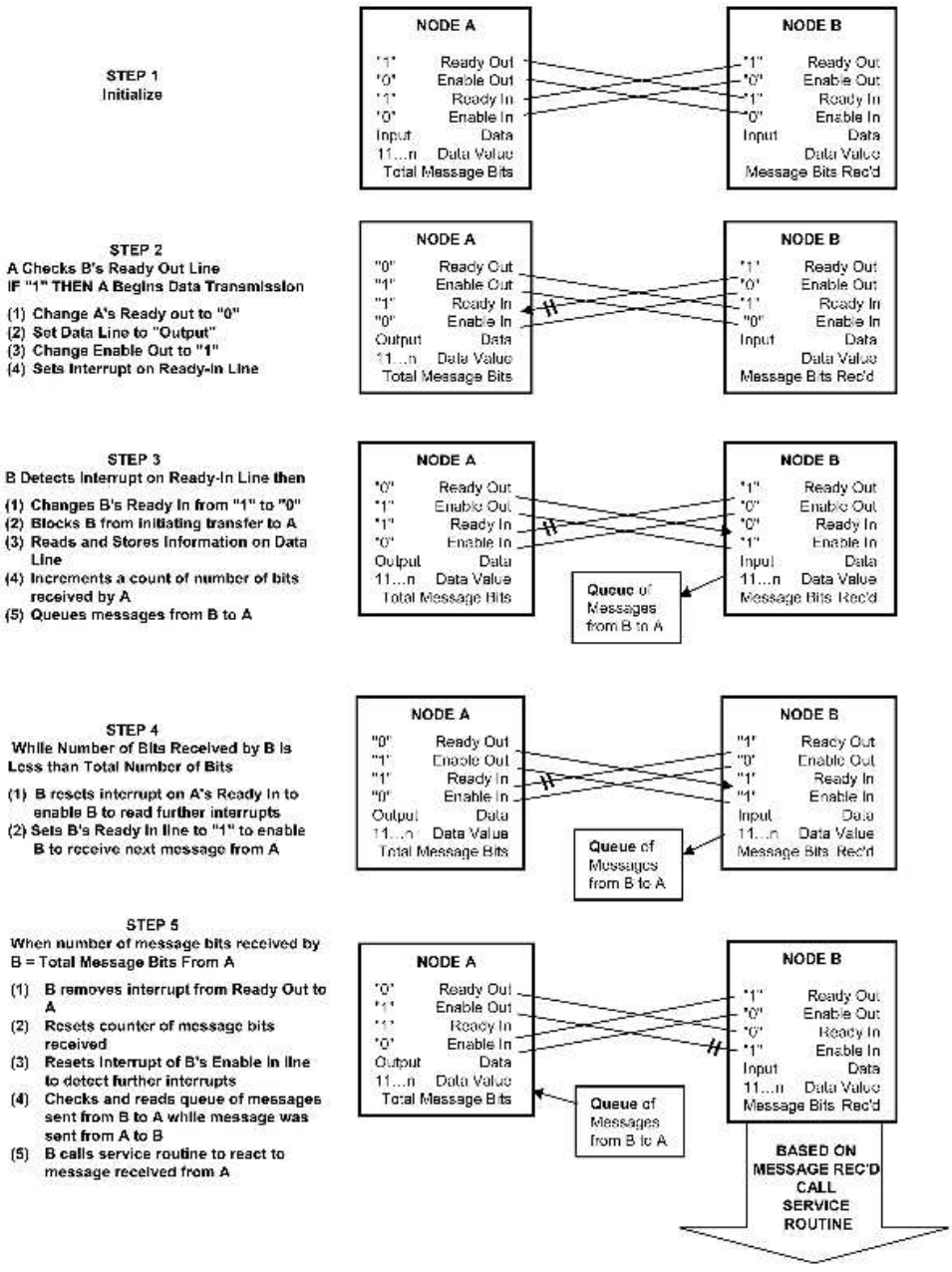

Figure 4-3: Flow chart of peer-to-peer communication.

Advantages of Design

Efficient

Our design avoids approaches based on continuously polling individual microprocessor pins in favor of a design that exploits event-based interrupts. These interrupts, fundamentally built into the MSP430F1611 architecture, do not affect processing rate or command flow until triggered thereby allowing normal operation.

Adaptable

Variable Message Length

Bits are precious. All communication between nodes need not be of a prescribed length: for a variety of reasons, nodes may need to send messages much smaller or larger than the 8-bit unit used by most systems. For example, two nodes, A and B, may only need to exchange a limited, discrete set of information. In such a case, A selects from a library of 16 messages to send to B. Similarly, B selects from a library of 4 messages to send to A. Instead of each node's transmitting a full 8-bits to its neighbor, A need only transmit 4 bits to B and B need only transmit two bits to A, making transmission from A to B twice as efficient as the 8-bit standard, and transmission from B to A four-times as efficient as the 8-bit standard. The design of the system allows message lengths to be changed either at compile time or dynamically.

Different Clock Rates

According to the MSP430F1611's datasheet, each additional Mega-Hertz our microcontroller is run at increases its current consumption by roughly 500 μA [19]. This difference in power consumption provides a real and substantial interest in running nodes below their peak points of operation. However, we must balance the power consumption benefit of this under-clocking strategy with our desire not to miss relevant information on the skin. One potential way of achieving both objectives is to realize the skin need not be in a heterogeneous clocking state. Some nodes can run with

clock settings faster than other nodes'. While synchronous communication schemes might fail in such a scenario, our asynchronous implementation completely supports it.

Symmetric Rate Control

At a high-level, each node performs three tasks:

1. Sampling sensors.
2. Processing data.
3. Communicating with other nodes.

Depending on the skin's desired function, there may be instances where one node needs to prioritize one of these tasks over the others. For instance, during a particularly noisy period, it may be more important for a node to sample its microphone than it is for the node to communicate its status to its neighbors.

Scaling down sensor sampling or processing rates to allow the node to weigh other tasks more heavily presents no real difficulty - one need only adjust timer ratios either dynamically or at compile time for each node. However, because communication rates are a function of not only how many communication requests a node makes (which a node can easily control) but also how many communication requests a node receives (which a node cannot easily control), scaling communication rates presents more of a problem. One might suggest a superficial solution in which a node sends control messages to its neighbors, requesting the neighbors reduce the number of messages they transmit.

However, this approach proves cumbersome: it assumes that neighbors will correctly interpret and respect rate selection messages. In addition, if rates need to be updated quickly, for instance, 50 times a second, such a scheme would require an extensive numbers of transmissions between nodes.

Our peer-to-peer protocol permits an alternate approach to using control messages. A side-effect of building a protocol capable of supporting mixed clock rates is

that we can use timers on either node to artificially throttle the rate of communication. A receiver can reduce the rate of communication by delaying switching its ready-out line from low to high after receiving a bit of information from the transmitter. Reciprocally, a transmitter can delay switching its enable-out bit from low-to-high, also throttling the the communication rate. These delays, if implemented by the timer interrupts built into the MSP430F1611 architecture, permit both the transmitter and the receiver to perform their regular processing tasks while waiting for the delayed communication to continue regardless of which node throttles the stream. In such a scheme, both the transmitter and receiver have symmetric control over the communication channel. Further, communication rates can be changed either at compile time or dynamically in response to particular stimuli.

Chapter 5

Results

This chapter presents raw data gathered from our skin. We begin by analyzing the skin's overall size, flexibility, and robustness in Section 5.1, and proceed to describe fully the speed and accuracy of the skin's communication lines in Section 5.2. We then change focus to the skin's sensors, demonstrating individual sensors' functionality in Section 5.3 and presenting aggregate sensor data for particular stimuli in Section 5.4. Finally, we conclude the chapter with an exploration of the skin's power consumption and a simple example program which shows the energy savings of the MSP430F1611's low power modes in Section 5.5. Each of this chapter's sections has a sister section in Chapter 6 which supplements the presented data with an analysis of their significance and implications. In general, we collected more data than practical to present in a single chapter. As such, we show additional data in Appendix B.

5.1 Basic Results

5.1.1 Size

Aside from a slight notch cut into the bottom right corner for orientation, each node is approximately square with dimensions of 1.04 in. x 1.02 in. To make these numbers more immediate, a United States quarter would roughly inscribe each of our boards; Figure 5-1 presents a visual comparison of our node's board size to United States

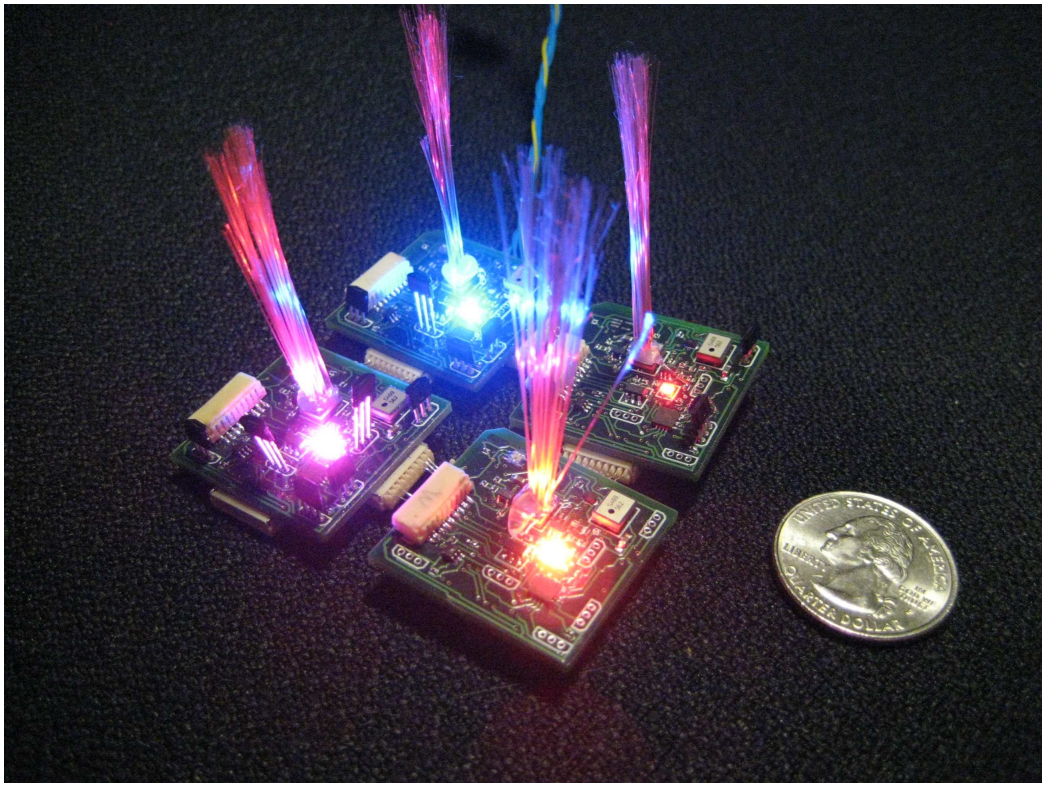

Figure 5-1: Notice that each node is small enough for a quarter to approximately inscribe it.

quarter.

The dimensions of our board compare favorably with our predecessor, Perez's (pictured in Figure 1-1). Perez's nodes were each 1.865 in x 1.865 in [33]. Therefore, our reworked design occupies only 30% the area of the previously developed S.N.A.K.E. system[1].

5.1.2 Flexibility

As mentioned in Section 2.8, interconnects between nodes could have been built from materials far more exotic than our current connectors, such as flex circuitry or cloth. While these systems may have had their advantages, to enable rapid prototyping, we chose connectors made from wire. As demonstrated in Figure 5-2, even with our

[1]In fairness, Perez's system did not require additional area for connectors: his nodes connected directly to each other. Our nodes, however, use wired connectors to link to each other. To assure reasonable flexibility, the length of these wired connectors is generally not shorter than .5 cm.

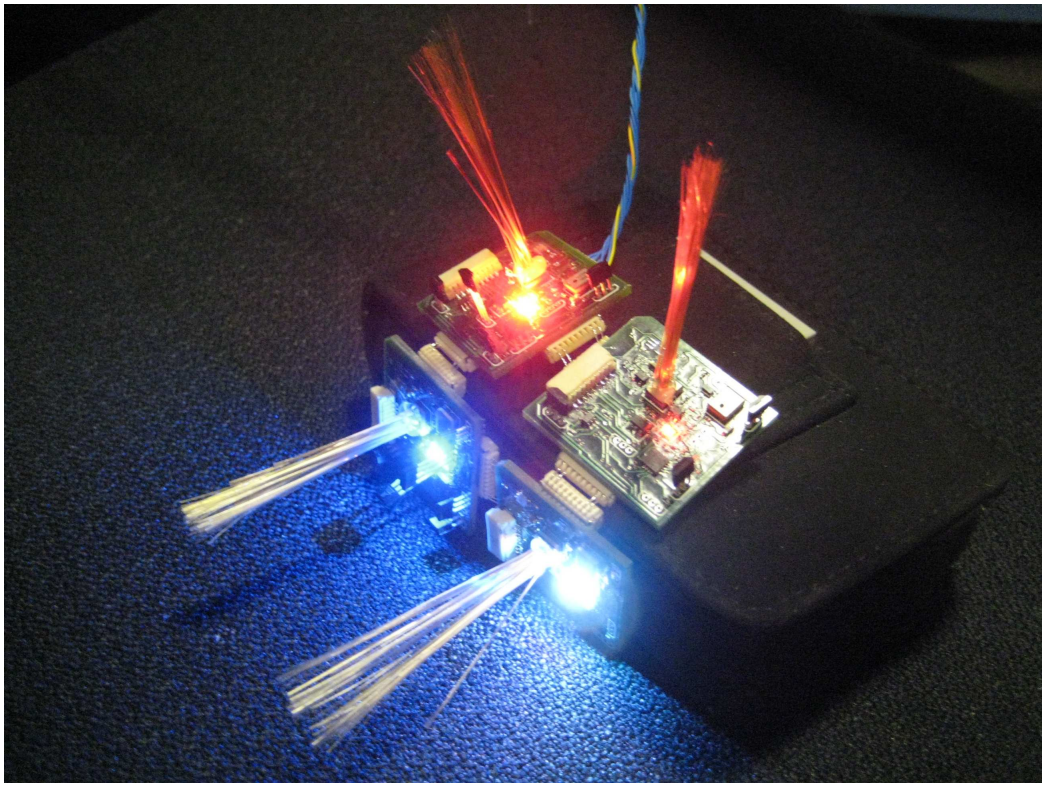

Figure 5-2: Four nodes bent around a camera case. Notice how the skin tolerates bends of 90^o.

wire connectors, our skin can easily bend at angles of over 90^o if the bend is required perpendicular to node connectors.

In general, flexibility along alternate axes was affected by the length and number of connectors linking nodes. A node linked by a single 1 cm long connector to its neighbor could not rotate more than 20^o in either direction. A node linked by a single connector approximately 2 in long could rotate a full 360^o in either direction.

Figures 5-3, 5-4, and 5-5 highlight the skin's overall flexibility and show the wide variety of topologies the network supports.

5.1.3 Robustness

Electrical, mechanical, and algorithmic robustness were early identified as some of the most important design goals of our work. Although robustness is difficult to quantify, we can present several observations made in the course of collecting data and working

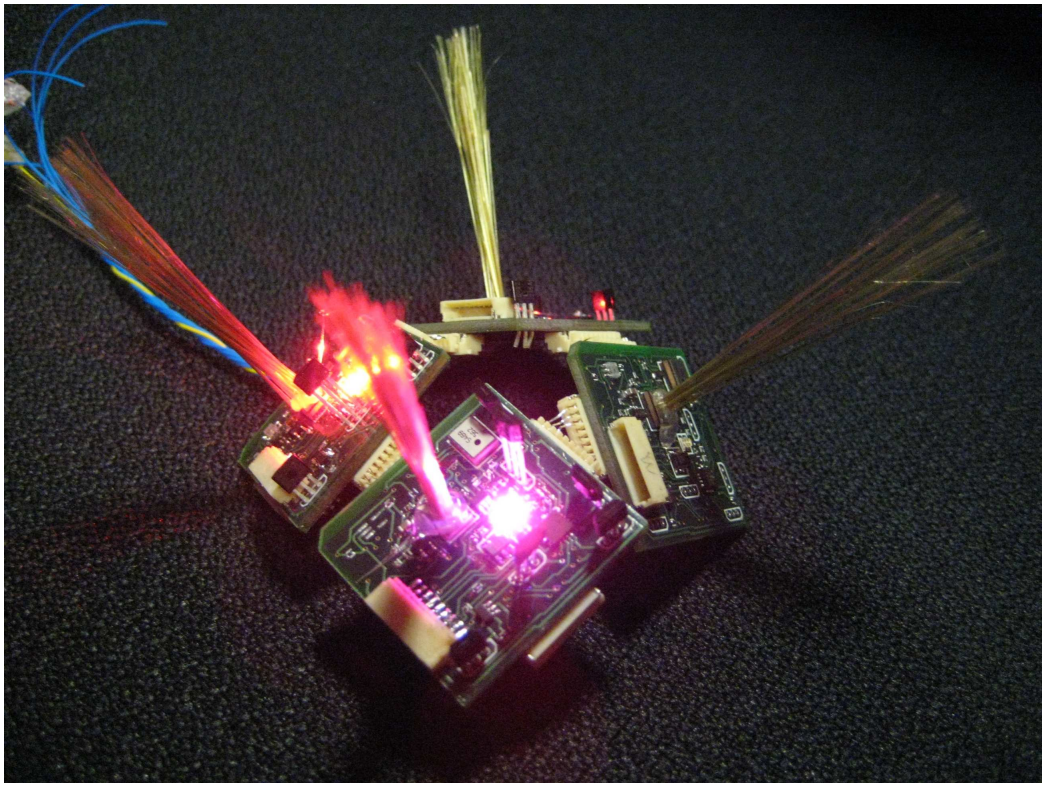

Figure 5-3: Four nodes bent into a strange topology. The figure highlights the flexibility of the overall skin.

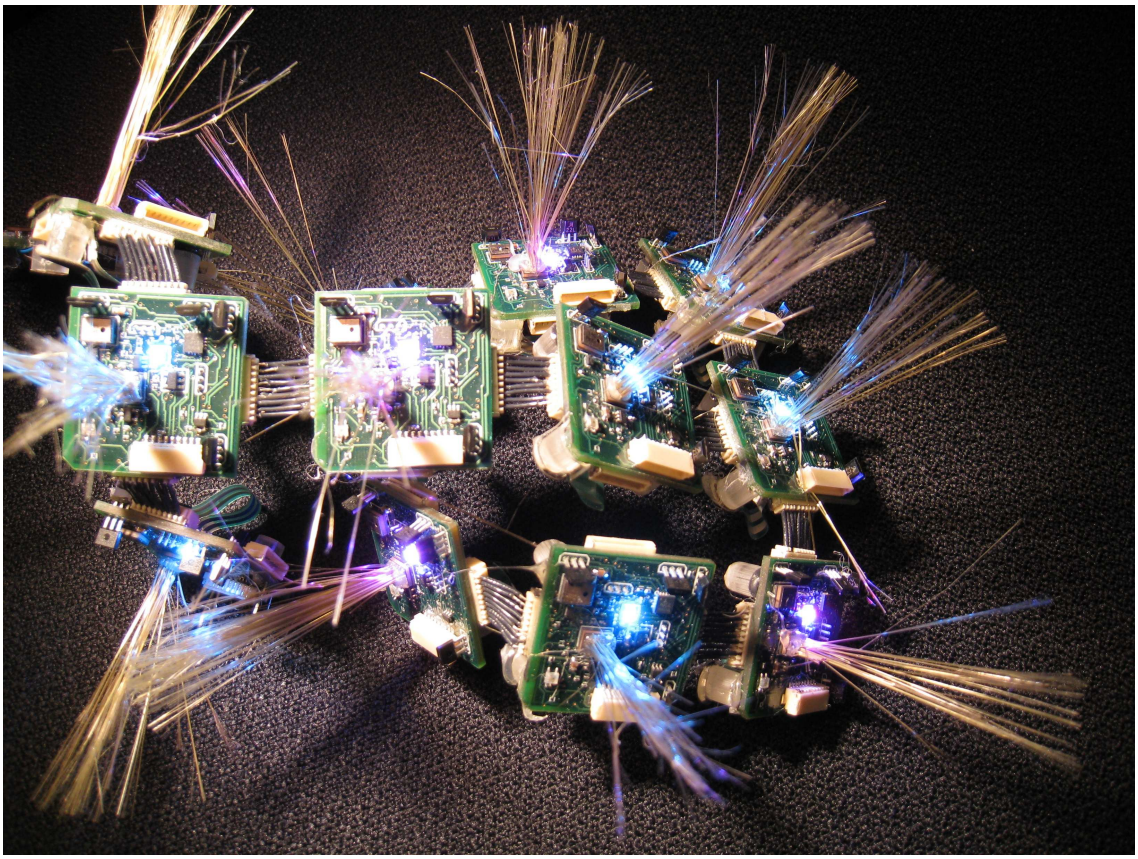

Figure 5-4: Thirteen nodes bent into a strange topology. The figure demonstrates the flexibility of the overall skin.

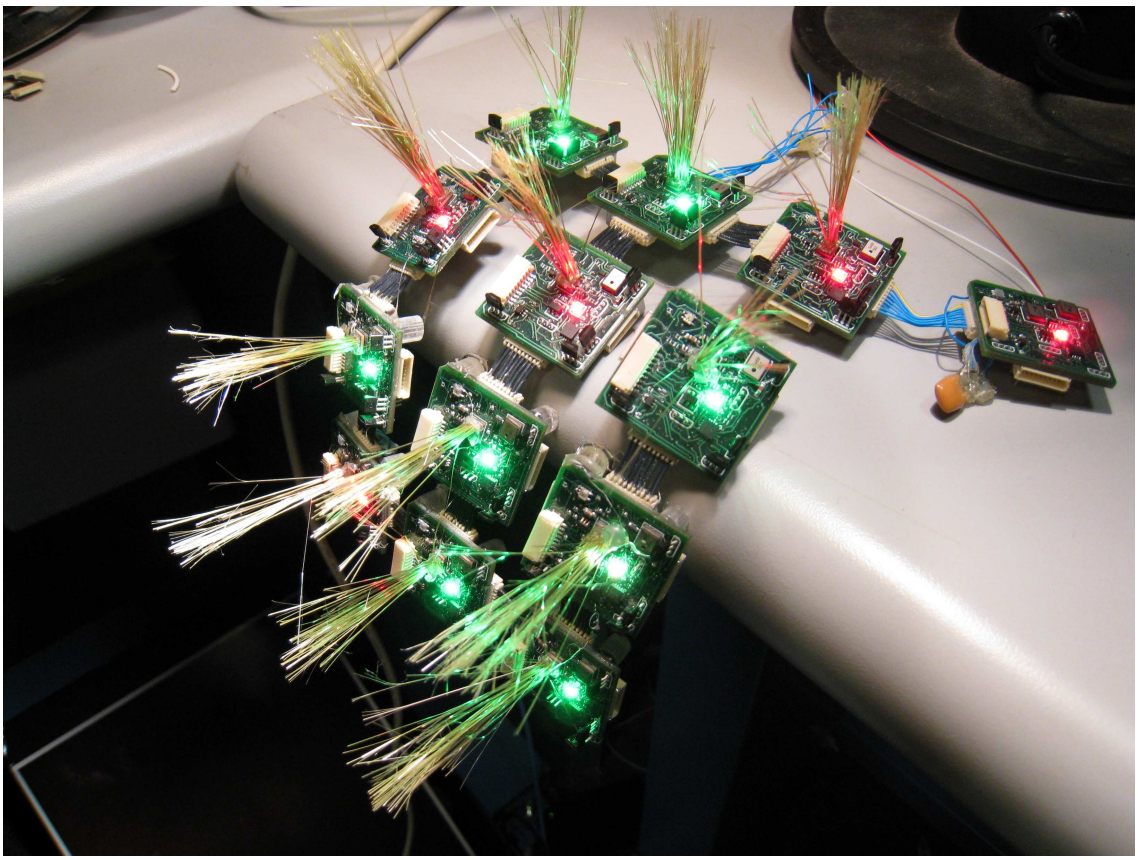

Figure 5-5: Thirteen nodes hanging off edge of a table. Demonstrates the flexibility of the overall skin.

with the skin.

Overall Skin and Visualization

We set a 13 node skin in data collection mode with:

- One master node running at 8 MHz querying the other 12 nodes on an I2C line, piping received information through an RS-232 line to a laptop computer running our visualization.

- The other 12 nodes running at the fastest rate their onboard clocks provide, sampling all their sensors and responding to I2C queries with their sensors' values.

Configured in this manner, the skin maintained functionality for over an hour with no immediate signs of degradation in transfer rate or sensor accuracy. The output of the visualization showed reasonable sensor values and responsiveness on each node. Further, the visualization produced no segmentation faults in the hour-long time it was running, and ran quickly - indicating a lack of memory leaks. The only constraint in running the skin and visualization for such a long period of time appeared to be the amount of data generated. Recalling Section 3.2, our visualization is built to automatically save data it receives and processes from the skin. Running our 13 node skin in the data acquisition mode described above for an hour generated a saved data file that was over 52 MB, a figure that corresponds to over 10.4 million sensor recordings. Although this sized file is easily manageable on today's computers, it should be noted that, eventually, our recorded data file would grow beyond our means to store it: in a little under 40 days, our skin and visualization program would overwhelm the 40 GB of free space on our recording laptop's hard drive. Such a use of course differs from the skin's intended functionality. It is more likely that a deployed system will only sparsely sample its sensors or quickly process and destroy collected data rather than storing it indefinitely. These numbers are only presented for thoroughness and to provide a full account of all of our work's limitations.

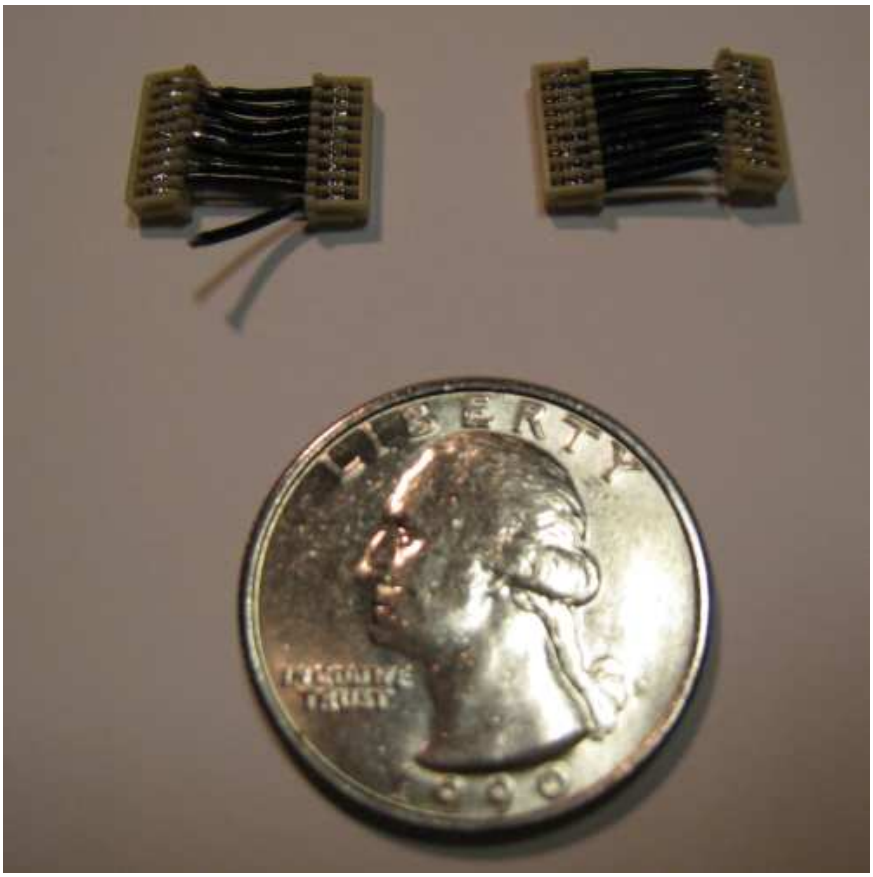

Figure 5-6: Picture of two connectors side by side. The connector on the left is fully functional while the connector on the right is broken. A quarter is placed in the picture to indicate scale.

Connectors

Connectors took a good deal of abuse, and often broke. During flexibility tests, they are exposed to a wide variety of stresses and unplugging nodes creates high levels of tension in the connector's wires. As seen in Figure 5-6, wires on connectors only break at the point of contact with the connector - a weak joint held by a combination of pressure and solder, and, generally, instead of multiple wires per connector breaking, only a single wire would come loose.

In general, the bending and flexing of connectors both casually and aggressively during flexibility tests was never observed to sever a connector. Connectors only broke when being unplugged from nodes. As such, we estimate connector lifetime based on the number of times a node is unplugged from the connector. For connectors built

with solid wire, on average, we could unplug a node between five and ten times from its connector before one of the connector's wires would fail. For connectors built from stranded wires, we could unplug nodes on average between twelve and twenty times before one of the connector's wires would fail. As discussed in Section 2.8, our connectors were primarily chosen for their ease in rapidly developing a prototype. In practice, a different substrate such as cloth [9], [10], [24] or flex circuitry [33] probably would have suffered less failure.

Whisker Sensor

Intense stimulation, for instance a hand's pressing down hard upon the skin, causes our whisker sensors to shed. Such stimulation repeated around a dozen times may necessitate replacing bristles on the whisker sensor. This procedure is painless, simply requiring one to re-heat the glue on the whisker sensor with the tip of a hot glue gun and insert a fresh batch of bristles. It is certainly conceivable that a better procedure or adhesive could be employed that would address this problem. However, because the whisker sensor's shedding is more of a minor inconvenience than a fundamental flaw, we focused our attention on other, more critical issues.

5.2 Communication Results

Quite a number of non-intuitive factors impact microcontroller function [19]. While these effects are unlikely to be relevant for many applications, because this section focuses on precise performance measurements, for thoroughness, we detail the conditions of our experiments before describing their results:

- All microcontroller code was written in C and compiled using Texas Instrument's Code Composer 3.0.
- Each microcontroller was run at 3.3 V.
- Experiments were run at room temperature.

- Direct comparisons between baud rates were done on the same microcontroller.

5.2.1 Peer-to-peer

Section 4.3 motivates and explains our design of a peer-to-peer communication protocol. Our primary goals for this system were to create an adaptive, fast, and efficient method for transmitting messages between nodes and their immediate neighbors. This section demonstrates the functionality of this system as well as providing benchmarks for its accuracy, speed, and adaptability.

Accuracy

To measure the accuracy of our peer-to-peer protocol, we connected two nodes, A and B. A was set to run at the fastest clock rate its onboard oscillator could support. Although Texas Instruments, the maker of our microcontroller, does not define a specific frequency for our setup, it does guarantee a clock rate between 6 MHz and 8 MHz for the voltage and temperature at which we ran our experiment [19]. A was programmed to perform no other task than to repeatedly send the character 'a' to its neighbor, B. Node B ran from an external 8 MHz oscillator and connected via a 115,200 baud RS-232 line to a Dell Latitude D630 laptop. B was instructed to listen for any messages from A and push any character received from A directly to the laptop. The laptop listened on its serial port for 30 seconds and counted:

1. The number of characters it received from B that were 'a'-s.

2. The number of characters it received from B that were not 'a'-s.

For convenience, Figure 5-7 depicts this experimental setup.

The results of this experiment are presented in Table 5.1.

Baud Rate

Figure 5-8, depicts a flow chart of our experimental setup. Please refer to it to supplement the experimental description that follows. The previous section presented

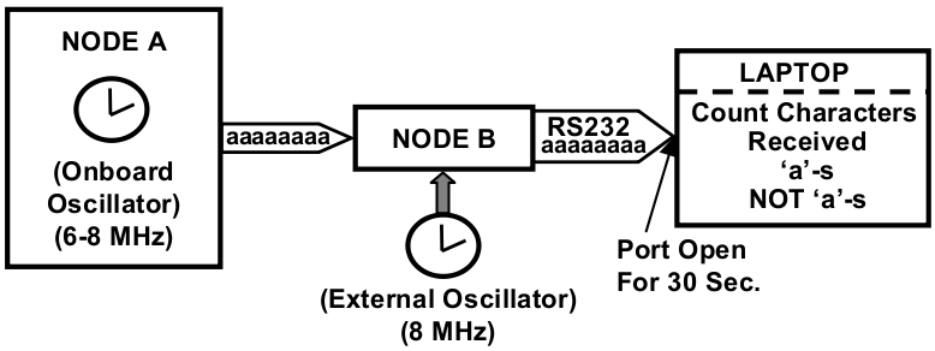

Figure 5-7: Block diagram depicting the experimental procedure used to measure the accuracy of peer-to-peer transmission.

Experiment	Time Period (seconds)	Correct Characters Received	Incorrect Characters Received	Baud Rate (bits/second)	Accuracy rate
A	30.0	21,721	0	5,792.3	100%
B	30.0	21,823	0	5,189.5	100%
C	30.0	21,617	0	5,764.5	100%
D	30.0	21,287	0	5,676.5	100%
AVG	30.0	21612	0	5763.2	100%

Table 5.1: Measurements of accuracy for peer-to-peer protocol

the results of piping information from one node through its neighbor to a computer. Such a setup throttles the rate at which peer-to-peer communication can actually occur: the node that passes information to the laptop cannot receive new messages while transmitting to the laptop. To determine the maximum rate at which peer-to-peer communications can proceed, we designed an alternate experiment consisting of two nodes, A and B. A runs as fast as the MSP430F1611's onboard clock permits - between 6 MHz and 8 MHz [19]. Similar to the setup of the intermediate node in the previous section, B's clock is sourced from an external 8MHz oscillator and connects via an RS-232 line to a Dell Laptop.

A is set to listen for a strong stimulus on its whisker sensor. When it receives that stimulus, it begins sending messages on the peer-to-peer connection to B. When B receives its first message, B turns on its red LED and begins counting the number of messages it receives. Once B performs 10,000 square root operations [2], B:

[2]We could have instead measured the time required to receive some number of messages. Our

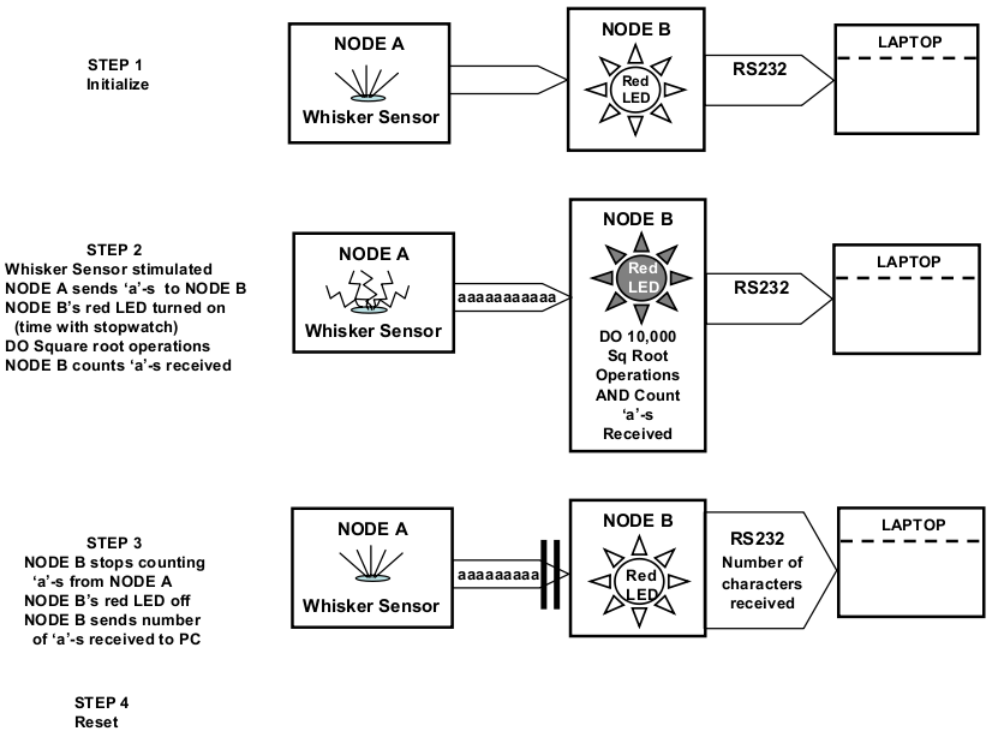

Figure 5-8: Block diagram depicting the experimental procedure used to measure the baud rate of peer-to-peer transmission.

1. Turns its red LED off.

2. Stops listening to and counting messages from A.

3. Begins transmitting via its RS-232 line to a listening PC the number of characters received from A while performing the 10,000 square root operations.

Dividing the number of bits received from A by the time B's red LED is on (as measured with a stop-watch) gives the baud rate. While messages are being transmitted from A to B, no extraneous interrupts are active on either node, ensuring that the rate found is the peak peer-to-peer rate.

The results of this experiment are presented in Table 5.2.

	Time (seconds)	Characters Transmitted	Baud Rate (bits/second)
A	45.81	61,358	10,715.2
B	46.12	61,570	10,697.3
C	46.62	62,278	10,686.9
D	46.75	62,309	10,662.5
E	46.75	62,319	10,664.2
AVG	46.41	61,966.8	10,685.2

Table 5.2: Measurements of baud rate for peer-to-peer protocol.

Adaptability

As explained in Chapter 4, either a receiver or transmitter can unilaterally throttle the rate of peer-to-peer transmission. This throttling can be done at either compile time or dynamically, and may be beneficial in scenarios where a node needs to unilaterally reduce its rate of communication in order to perform a more pressing computational task.

To highlight the computational flexibility this feature allows us, we performed an experiment very similar to that described in the previous section. Please see the flow chart presented in Figure 5-9 or read the next paragraph for a description of

choice of using 10,000 square root operations was largely arbitrary. From earlier experiments, we knew that 10,000 square root operations would consistently run for an easily measurable, fixed amount of time, and therefore we selected it.

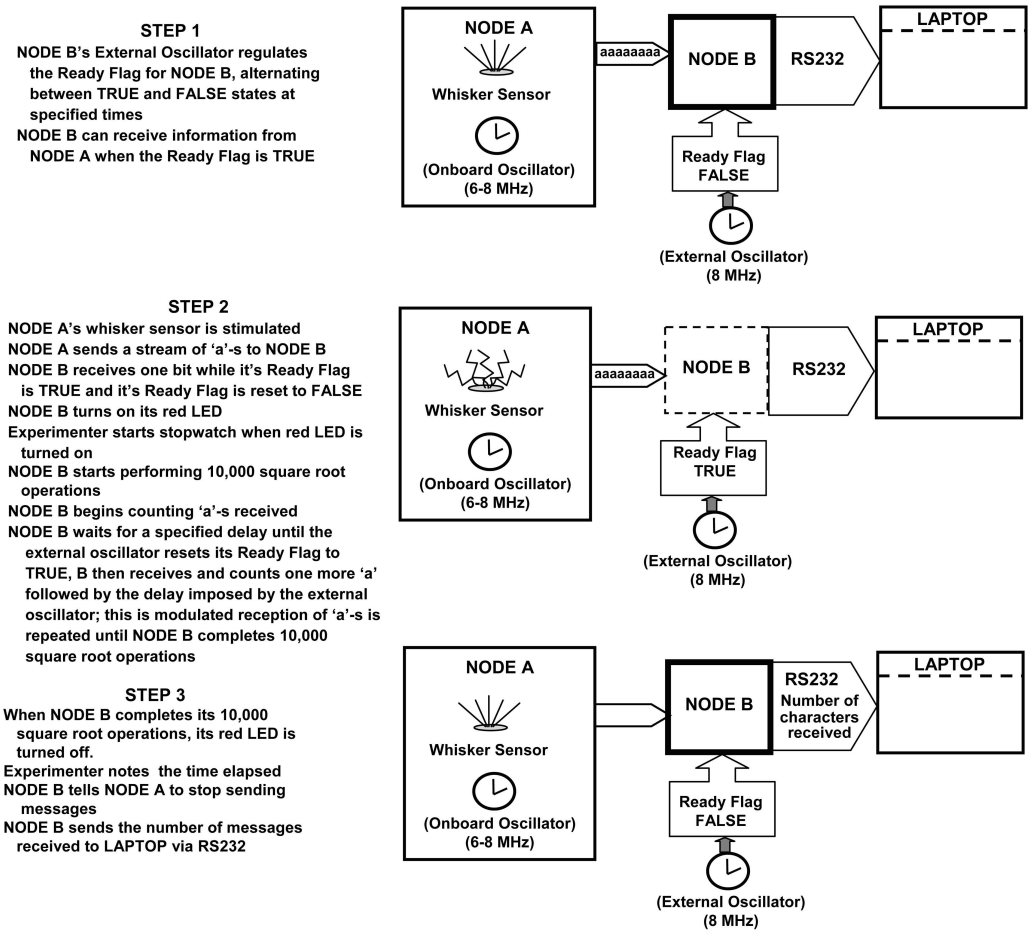

Figure 5-9: Block diagram depicting the experimental procedure used to demonstrate the effects of allowing a node to set different peer-to-peer communication rates.

our experimental procedure, consisting of two nodes, A and B. A runs as fast as the MSP430F1611's onboard clock permits - between 6 MHz and 8 MHz [19]. Similar to the setup of the intermediate node in the previous section, B's clock is sourced from an external 8 MHz oscillator and connects via an RS-232 line to a Dell Laptop.

A is set to listen for a strong stimulus on its whisker sensor. When it receives that stimulus, it begins sending messages on the peer-to-peer connection to B. B is programmed to delay setting its ready flag using one of the MSP430F1611's onboard timers. (Recall from Section 4.3 that A will only continue sending its message once B has toggled its ready flag to true.) The timer returns B to its normal operations until a specified number of clock cycles have elapsed, at which time, B indicates it is

ready to receive an additional bit of information from A[3]. The first time B receives a full 8-bit message from A, B:

1. Turns on its red LED.
2. Begins counting the number of messages it receives (each of these messages are also throttled with the same timer strategy described above).

Once B performs 10,000 square root operations, B:

1. Turns its red LED off.
2. Stops listening listening to and counting messages from A.
3. Begins transmitting via the RS-232 line to a listening PC the number of characters received from A while performing the 10,000 square root operations.

The length of B's ready-delay impacts the balance between how much time B spends 1) performing its 10,000 square root operations and 2) receiving messages from A: if the delay is very, very long, B almost exclusively performs its square root operations and turns off its LED quickly; if the delay is very, very short, B almost exclusively receives messages from A, and B's LED is on for a longer time.

The effects of differing delays on the number of 8-bit characters B receives and the time it takes for B to perform its 10,000 square root computation are presented in Table 5.3. Each table entry contains the averaged values of five separate measurements. For reference, the bottom row of the table gives the time B requires to perform the 10,000 square root operations when receiving no messages.

5.2.2 I2C

As explained in Section 4.2.1, our I2C bus provides an effective route for hierarchical communication. Although capable of supporting additional functionality, we primarily used the I2C bus with a single master querying a network of uniquely addressed

[3]It is important to remember that because the routine nodes use to send peer-to-peer messages are tied to pin interrupts, a transmitter does not hang waiting for the receiver to indicate readiness. Rather, like the receiver, the transmitter continues its normal processing until the receiver is ready for an additional bit.

Length of ready-delay (in clock tics)	Avg. Number of Characters Received	Avg. Time to Complete 10,000 Square-root Operations	Avg. Baud Rate (bits/second)
0	61,986.8	46.4100 seconds	10,685.1
500	16,003.8	24.1440 seconds	5,302.7
1000	7648.8	18.8920 seconds	3,239.0
Baseline: 14.1260 seconds to perform 10,000 square-root operations while receiving no messages			

Table 5.3: Ready-delay's effects on number of 8-bit characters received from neighbor via peer-to-peer protocol and processing time required to perform 10,000 square root operations.

slave nodes. For the most part, our I2C bus operated precisely to its specifications, [37]. Repeated oscilloscope measurements showed that the I2C bus' clock line ran at a frequency between 382.4 KHz and 393.7 KHz - just slightly slower than the 400 KHz rate promised by the MSP430F1611 datasheet [19].

To derive baud rate and accuracy figures, we connected twelve slave nodes in a grid to a single master node. The master node was additionally connected to a PC via an RS-232 line. The master node queried each slave node individually at its fastest possible rate and forwarded all the slave nodes' responses on to the PC.

To test accuracy, we programmed each slave to respond to all master node queries with the character 'a'. Comparing the number of 'a'-s to the number of non 'a' characters received by the PC gave us a sense of how accurate our data transmission scheme was. Repeated runs of this experiment are summarized in Table 5.4. As the reader can see from the results provided, our I2C bus was perfectly accurate.

Experiment	Time Period (seconds)	Correct Characters Received	Incorrect Characters Received	Baud Rate (bits/second)	Accuracy
A	30.0	321,000	0	85,600	100%
B	30.0	320,853	0	85,560.8	100%
C	30.0	325,851	0	86,893.6	100%
AVG	30.0	322,568	0	86,018.1	100%

Table 5.4: Accuracy results of I2C bus.

The baud rates presented in Table 5.4 may be a little misleading. They assume that each slave node solely listens to and immediately responds to any master query.

In practice however, this is a flawed assumption: slave nodes are unlikely to dedicate themselves exclusively to listening for I2C interrupts. More likely, slave nodes will be processing data, sampling their sensors, and communicating with their neighbors via their peer-to-peer connections. All these operations effectively reduce the I2C bus' baud rate. For instance, with twelve slave nodes' sampling their sensors at approximately 16,000 Hz, the I2C line ran only at a baud rate of 49,600 bits per second, a little over half the rates we found when each slave node was configured solely to listening for and responding to master node queries.

5.3 Basic Sensor Results

The following sections show data collected from our skin in response to assorted stimuli. These data were collected with the skin configured in either a three-by-three grid with a listening master node or a three-by-four grid with a listening master node. We exclusively used the I2C line to pipe data from our slave nodes to our master node and on to a listening PC. Our I2C line achieved data rates of approximately 6,000 bytes per second. Because each slave node transmitted a full load of sensor information in 14 bytes, this rate assured us of a theoretical update rate (how many times each individual node transmits all of its sensor information per second) of approximately 35 Hz. In practice, we noted that this update rate varied slightly, and ranged between 30 Hz and 40 Hz depending on what code we were running.

5.3.1 Light Results

Our light sensor proved to be the most reliable device on each node. As we typically observed a 60 Hz modulation from artificial light sources on our light sensor's output, our nodes computed a windowed average of our light sensor's values. This averaging greatly smoothed the AC oscillations that our nodes recorded.

Figure 5-10 presents data collected from an array of twelve nodes as a flashlight zooms in and out on one of the center nodes, *H*. Slightly after the ten second mark, the flashlight is at its closest point to Node *H*. As expected, *H* has its peak values at

this time. Also as expected, all other nodes (except for E) show miniscule values on their light sensor outputs near the ten second mark. (Node E shows a precipitous dip in its light sensor output at the ten second mark. The reason its value does not go to zero when our flashlight zoomed in on H is that our flashlight was bigger than a single node. Therefore our flashlight partially illuminated Node E even while zoomed in primarily on Node H.)

Further showing the light sensor's basic functionality, Figure 5-11 presents light sensor readings as a hand passes over the skin three times casting shadows, showing that shadows are a reliable indication of dynamic stimuli in illuminated environments.

5.3.2 Microphone Results

As noted in the signal conditioning description presented in Section 2.5, the microphone's output is pushed through through a negative peak detector and an envelope follower that extends the length of received audio stimuli. This setup enables each node to throttle its rate of sampling audio data while still detecting important environmental events that produce bursts of audio.

For particular applications, the peak values of the audio signal over a small period of time may prove more relevant than the actual audio waveform. Therefore, we perform a windowing procedure on each node's microcontroller that reports the maximum audio signal received in approximately one-third second intervals. A display of the results of this operation is shown in Figure 5-12 along with a figure presenting the corresponding un-windowed audio signal in Figure 5-13.

We additionally demonstrate our microphone's functionality in Figures 5-14 and 5-15. The first figure presents the windowed microphone values of twelve nodes connected in a grid in response to five claps of varying intensity. The second figure presents the corresponding raw, un-windowed audio values of these same nodes. Unfortunately, Node J's microphone was not properly functioning while data were being collected.

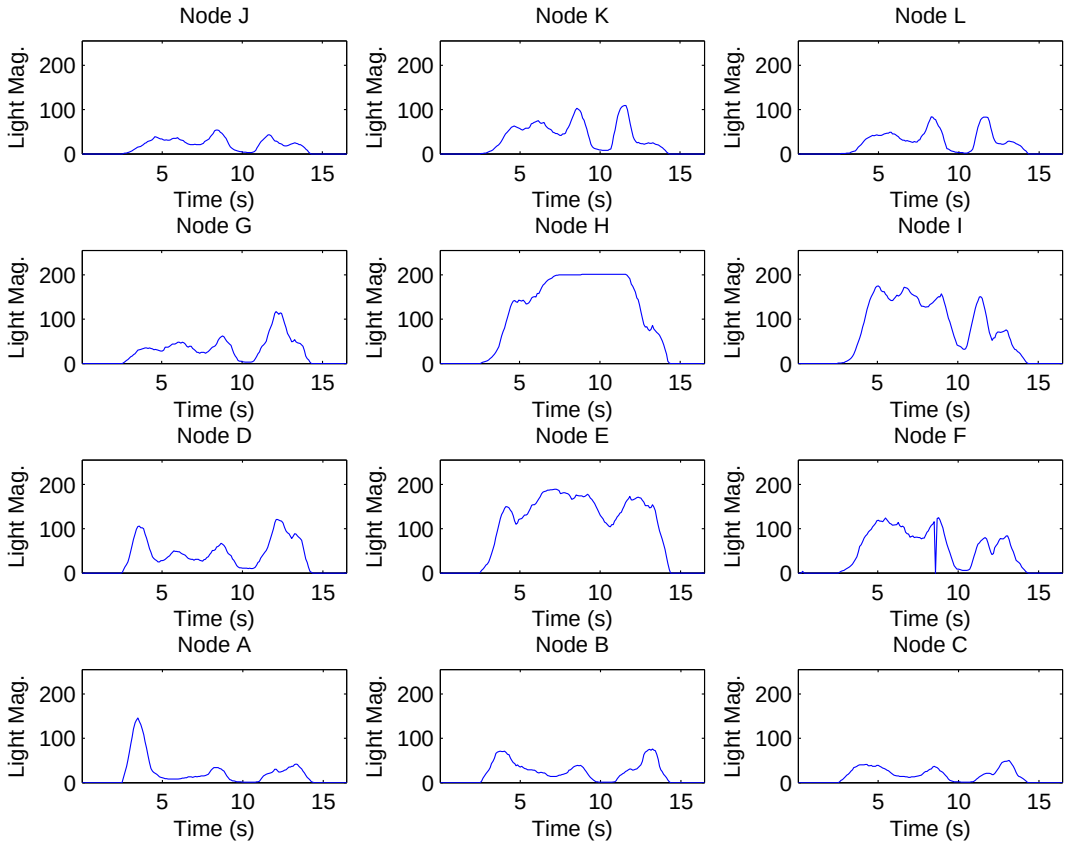

Figure 5-10: Light responses of twelve nodes arranged in a grid as a flashlight zooms in and out on Node H. Slightly after the ten second mark corresponds to the point at which the flashlight was zoomed closest to H. Notice that as expected, all other nodes except E show miniscule values on their light sensors at the ten second mark. Node E only shows a relatively minor dip in its light sensor values because our flashlight was larger than each node, and therefore it partially illuminated Node E even when zoomed in on H very closely. Node F shows a brief sensor failure before the eight second mark.

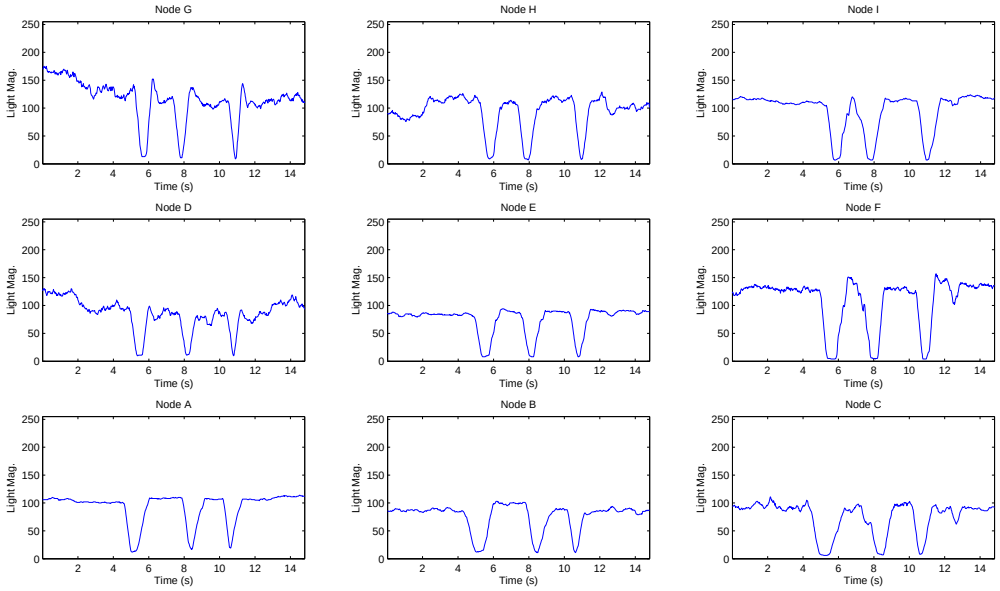

Figure 5-11: Light responses of nine nodes arranged in a grid as a hand passes over the skin casting shadows.

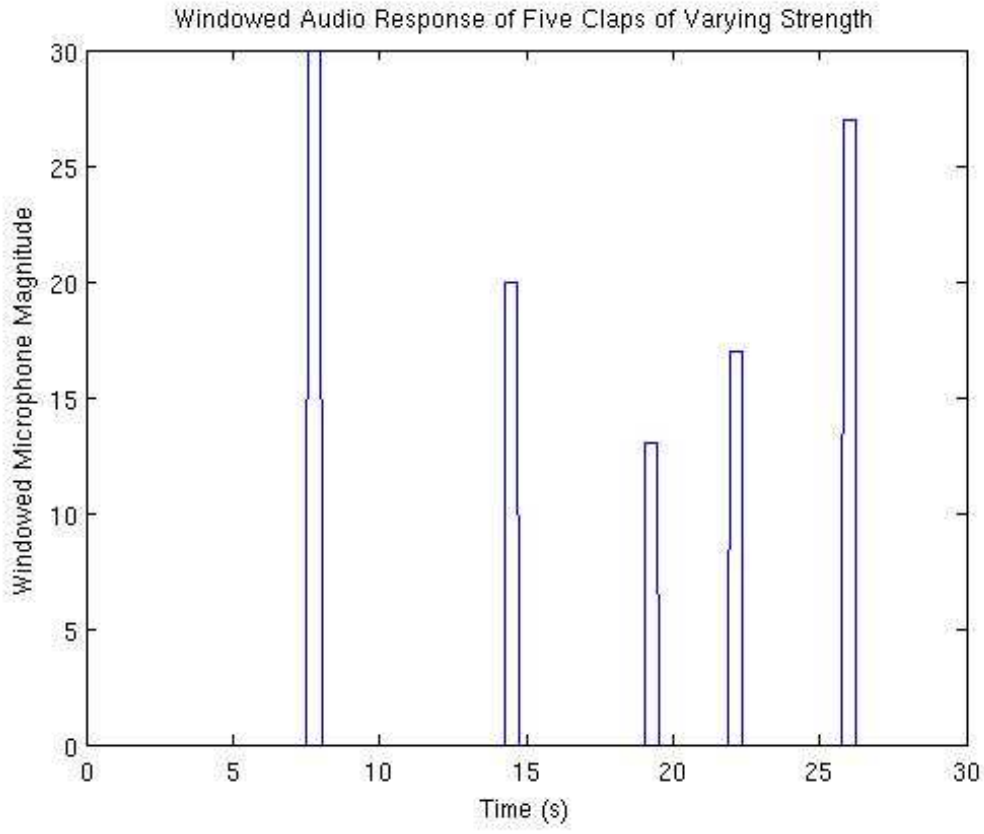

Figure 5-12: Windowed microphone response to five claps of varying intensity.

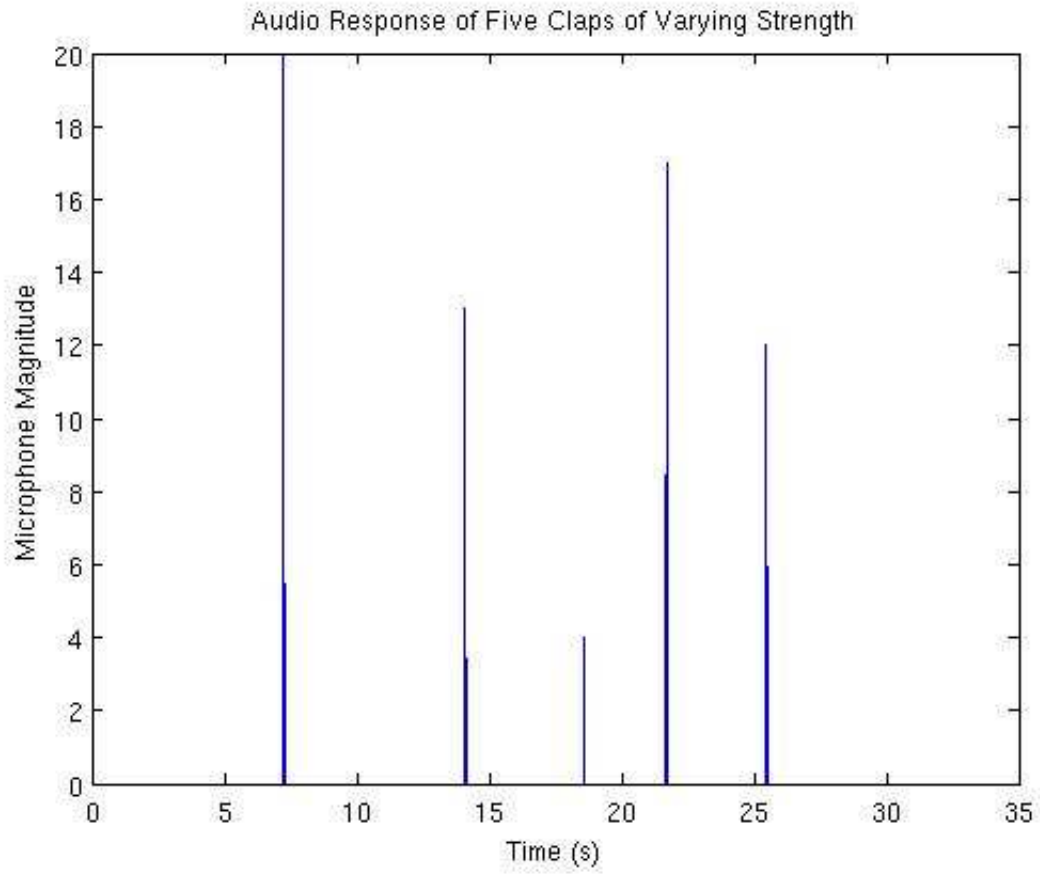

Figure 5-13: Un-windowed microphone response to five claps of varying intensity.

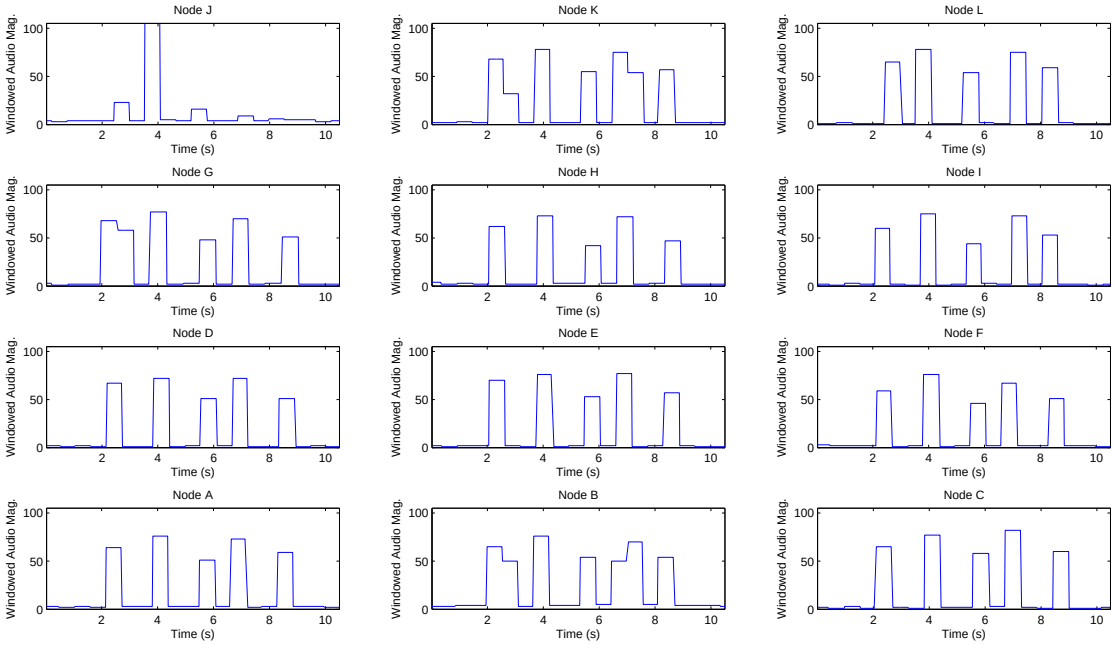

Figure 5-14: Windowed microphone responses to five claps of twelve nodes connected in a grid. Node J was not properly functioning during data collection.

5.3.3 Whisker Results

Taking inspiration from animal whiskers, we glued fine paint-brush bristles directly to a microphone to create a whisker sensor. The microphone responds to vibrations generated from the sensor's bristles' being touched. The glue affixing the bristles to the microphone prevented any audio signal from exciting the microphone. Our whisker sensor was built primarily for proximity detection, but as demonstrated in Appendix B is sensitive enough to pick up other environmental stimuli as well. This section presents basic data collected from our whisker sensors. Our whisker sensor suffered from some of the noise found on our temperature sensor (described in Section 5.3.5) while running our I2C bus. Therefore, we filtered our whisker sensor's output, reporting back only values that exceeded a pre-defined threshold.

Figure 5-16 is a mosaic of data from the whisker sensors of nine separate nodes connected in a grid. A hand, held parallel to the base of the skin, passes across the top of each of the whisker sensors three times. The brief sensor peaks displayed between

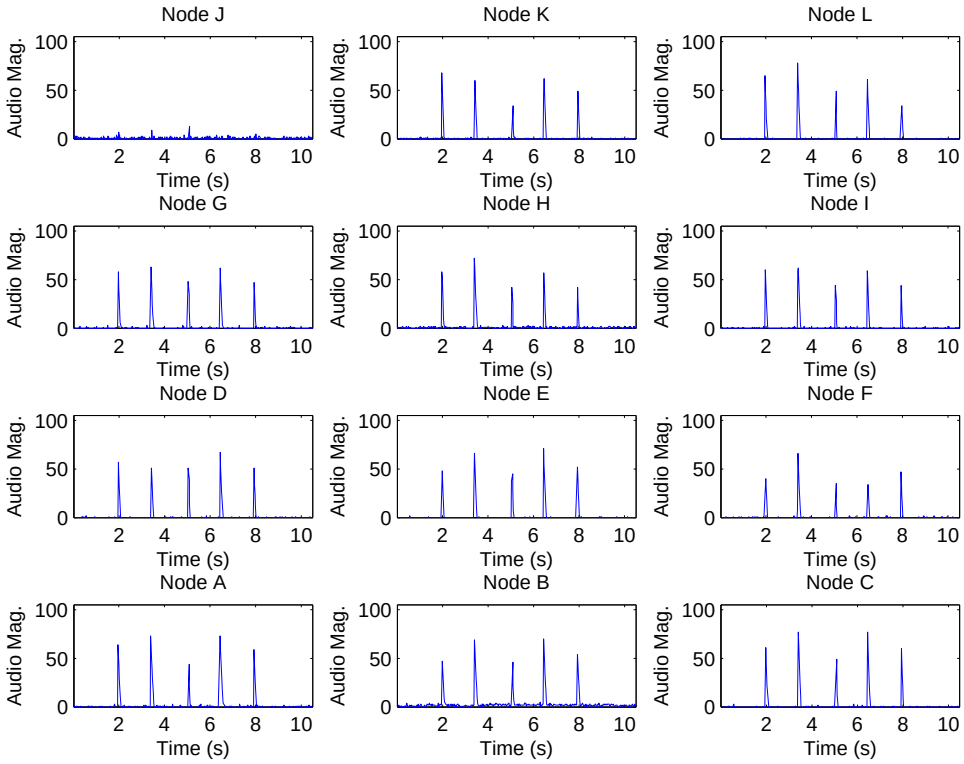

Figure 5-15: Un-windowed microphone responses to five claps of twelve nodes connected in a grid. Node J was not properly functioning during data collection.

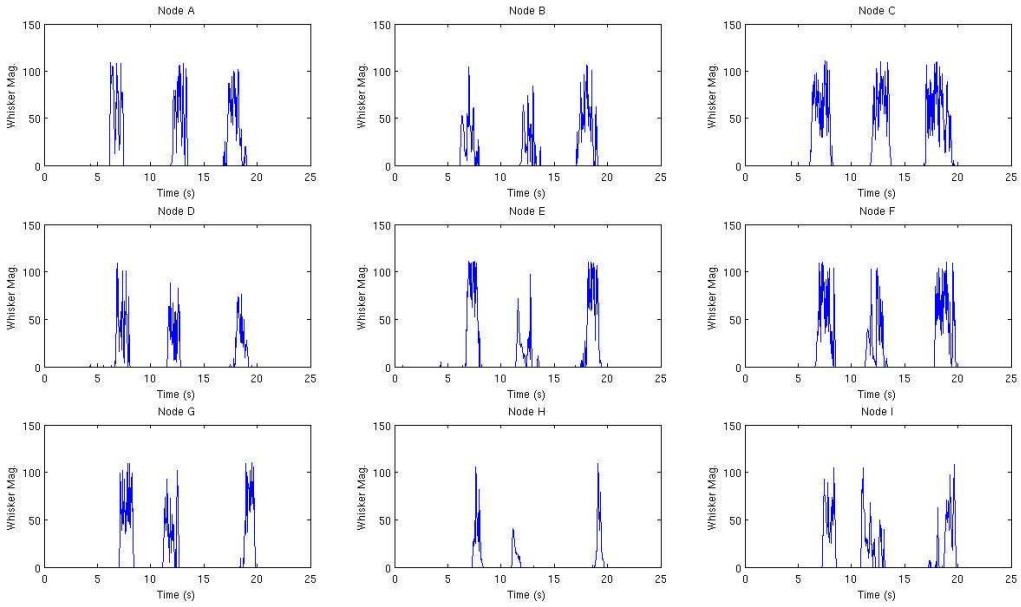

Figure 5-16: Whisker responses of a grid of nine nodes as a hand passes over the tips of the whisker bristles three times.

the five and ten second marks, between the ten and fifteen second marks, and between the fifteen and twenty second marks correspond to the hand's contacting each node's whisker sensor. The data before the five second mark and after the twenty second mark correspond to the sensor's unstimulated, quiescent state.

Nodes E and H in Figure 5-16 show that our whisker sensors can be used for more than just binary motion detection - their relatively slight responses occurring just after the ten second mark indicate that each sensor can perceive the difference between light and heavy stimuli.

5.3.4 Pressure Results

FSR Sensor

Section 2.4 details FSRs and their functions. In brief review: an FSR is a device that exhibits a resistance change with the amount of force applied. Although this resistance change is somewhat non-linear, its monotonicity allows us to use the raw values from the sensor directly as a proxy for force or pressure for several applications.

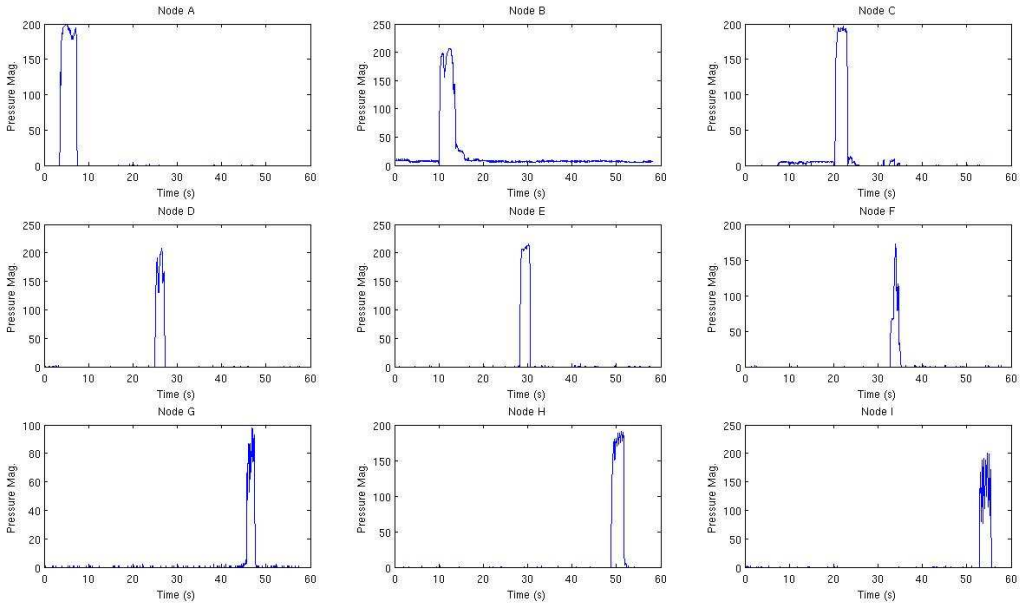

Figure 5-17: FSR responses of a grid of nine nodes as each node is sequentially pressed upon.

Our FSR is mounted on the bottom right corner of each node. Figure 5-17 shows the results of a thumb's sequentially pressing down hard upon nine distinct nodes connected in a grid pattern. As can be seen from the graphs, the FSRs saturate quickly. This effect is due to the directness of the stimuli (each press was targeted on the node immediately above where each FSR was mounted) and the strength of the stimuli. Later sections, such as 5.4.3, show that the sensors are well within their operating ranges for other stimuli.

QTC Sensor

As explained in Section 2.4, we built a pressure sensor from quantum tunneling composite. This sensor performed much worse than our FSR sensor: it was not as sensitive as an FSR nor as easy to mount on our board. However, despite these problems our QTC-based sensors were able to detect pressure stimuli. Figure 5-18, for instance shows a comparison of our FSR sensor against our QTC sensor for the same pressure stimulus. Note that our QTC sensor detects the pressure event, but does not respond

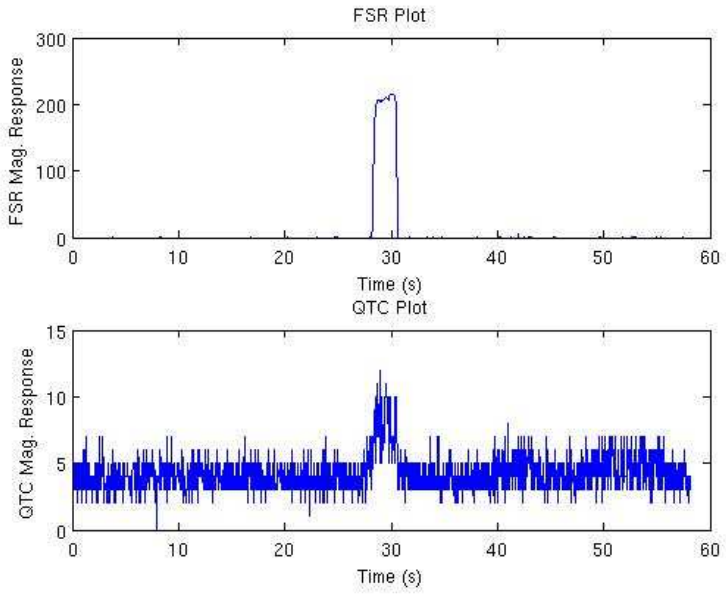

Figure 5-18: Comparison of FSR sensor (top) against QTC sensor (bottom) for same stimulus.

to it as cleanly as our FSR sensor.

5.3.5 Temperature Sensor

Our LM20CIM temperature sensor is highly non-linear and sensitive to a wide range of temperatures (-40^o C to 150^o C). For reasons expanded upon in Section 6.3 of this thesis, our temperature sensor produced noisy data encountered across temperatures 65^o F to 100^o F. Despite this fact, we are still able to determine general temperature trends from the data. Figure 5-19, for instance, shows the response of a node's temperature sensor to a heat gun's being applied to it at the ten second mark. Notice that, although there is quite a lot of disturbance in the response, the suddenness of the stimulus as well as the time at which it began and ended are still quite visible. Of course, a heat gun is an exaggerated stimulus: nothing in our skin's environment will be producing the same intense temperatures. Therefore, we also present the temperature sensor's response to a thumb's being placed directly on it and removed in Figure 5-20. Again, although noisy, the time at which the thumb was applied and removed appear quite apparent. In addition, if we compare the two Figures, we see

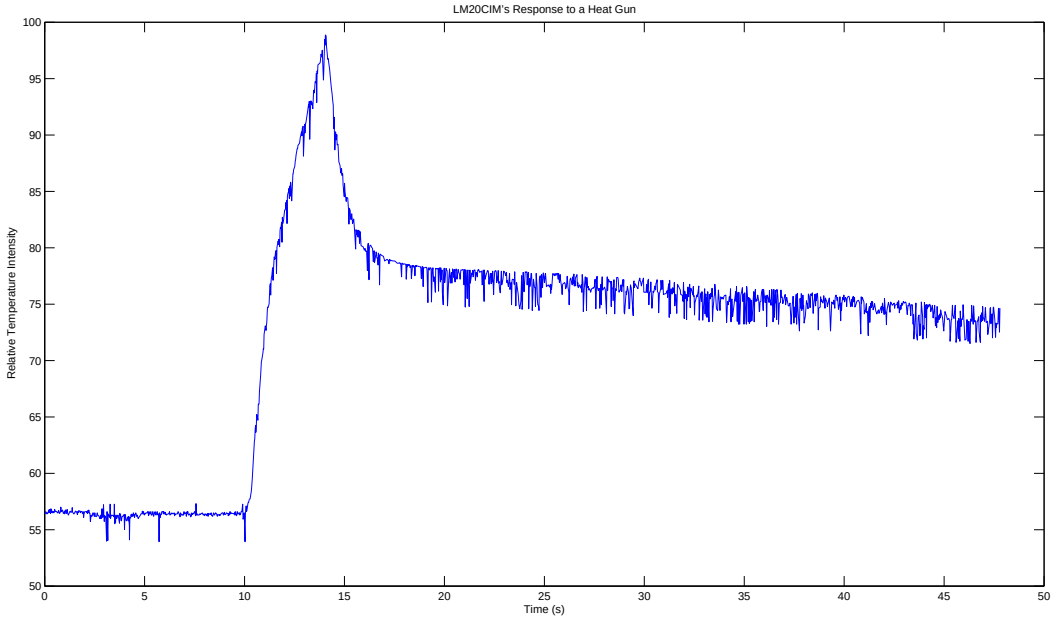

Figure 5-19: Temperature sensor response to heat gun. At the ten second mark, a heat gun was turned on over the temperature sensor. Near the fifteen second mark, the heat gun was removed. After the experiment, the board still felt hot to the touch, explaining why the output of the sensor did not quickly return to its pre-stimulated state.

that our temperature does provide at least some information on relative intensity: the LM20CIM's response to the heat gun dwarfs its response to a thumb's being applied.

To clean our data, we used a fifth order low-pass Butterworth filter on our temperature sensor readings. Figure 5-20 indicates that the data could be further improved by applying a simple outlier rejection algorithm to remove spikes from digital pickup prior to filtering. Thermal dynamics tend to be slow, and therefore this filter introduces inconsequential time lag. Both this low-pass filtering and the non-linear transform of the data were performed offline - experiments indicated that executing the floating-point operations associated with these functions onboard our microcontroller significantly impacted the speed and therefore sampling rate of each node.

Figures 5-21 and 5-22 show the impacts of this filter on the data presented in Figures 5-19 and 5-20 respectively. As can be seen, although the sensor's output still appears slightly jittery, our Butterworth filter went a good ways in smoothing

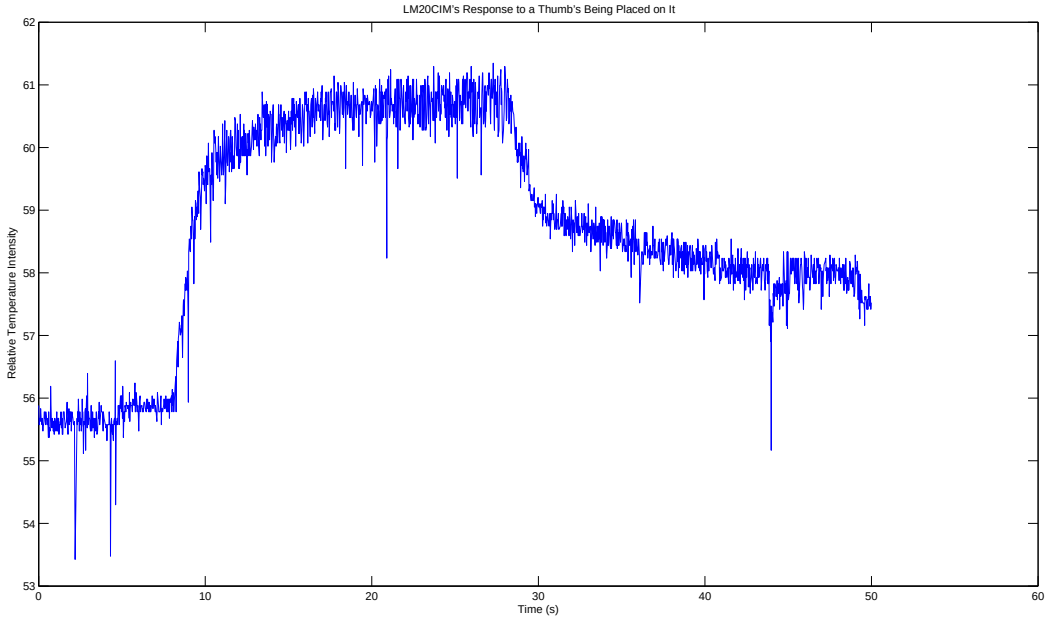

Figure 5-20: Temperature sensor response to thumb's being applied to it. The thumb is applied near the ten second mark and removed near the thirty second mark.

our results while still allowing us to distinguish key events such as direct animate touch from non-animate touch. Of course, as will be discussed in Section 6.3 a more sensitive sensor or a passive infrared (PIR) sensor may have been a better choice for our apparatus.

Governed by the kinetics of heat diffusion, temperature changes may operate on a longer time scale than several other of our stimuli. Interestingly, our skin is capable of detecting not only these temporal dynamics, but also the temperature gradient of a heat source. Figures 5-23, 5-24, and 5-25 specifically highlight this. Figure 5-23 shows the temperature readings of twelve nodes arranged in a grid in response to an incandescent bulb's being placed at the top left of the skin. Figure 5-24 shows the temperature readings of the same twelve nodes when the incandescent bulb was turned off. Notice in both figures that temperature changes are not instantaneous and instead proceed over a relatively long time scale, creating a gradient. Figure 5-25 is an interpolated snapshot of temperature readings at the 83.8 second mark of Figure 5-23. The red in the top left corner indicates areas of higher temperature

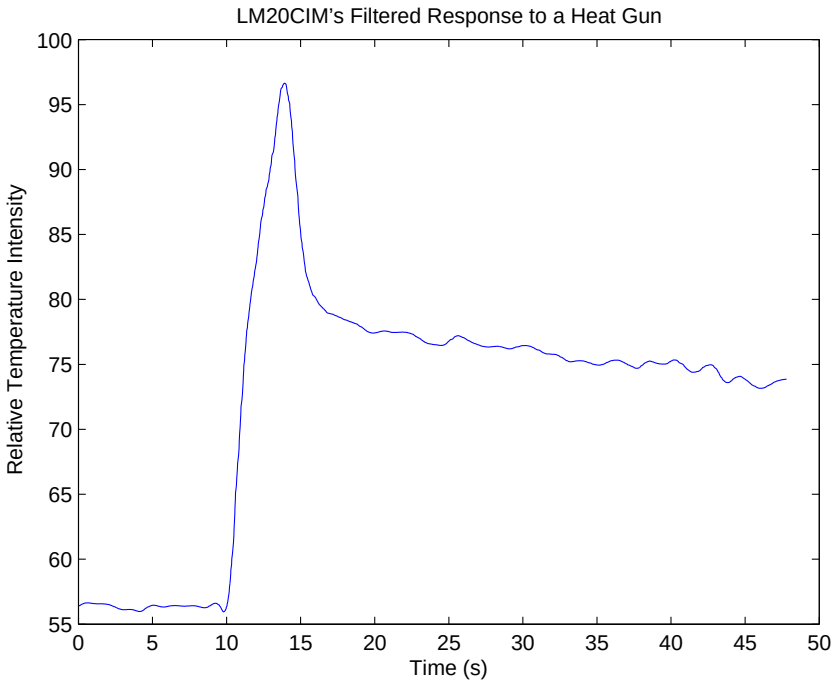

Figure 5-21: Filtered response of temperature sensor to heat gun. We use a fifth-order Butterworth filter with a low cutoff frequency of 1 Hz on the data presented in Figure 5-19. At the ten second mark, a heat gun was turned on over the temperature sensor. Near the fifteen second mark, the heat gun was removed. After the experiment, the board still felt hot to the touch, explaining why the output of the sensor did not quickly return to its pre-stimulated state.

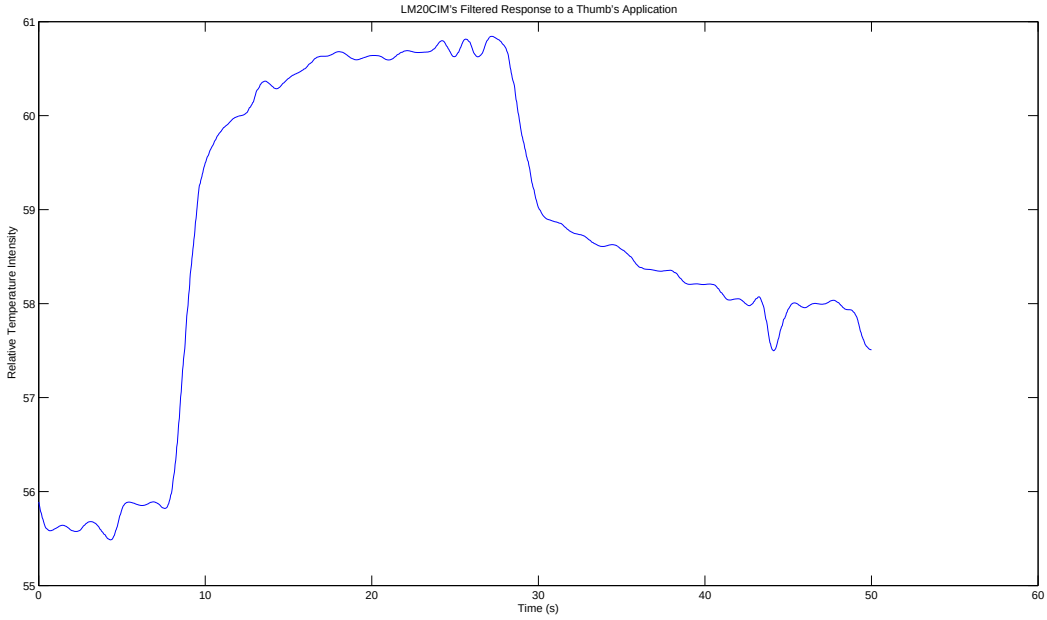

Figure 5-22: Filtered response of temperature sensor to thumb's being applied to it. This is the filtered version of the data presented in 5-20. We use a fifth-order Butterworth filter with a low cutoff frequency of 1 Hz. At the ten second mark, a thumb was placed on the temperature sensor. Near the thirty second mark, the thumb was removed.

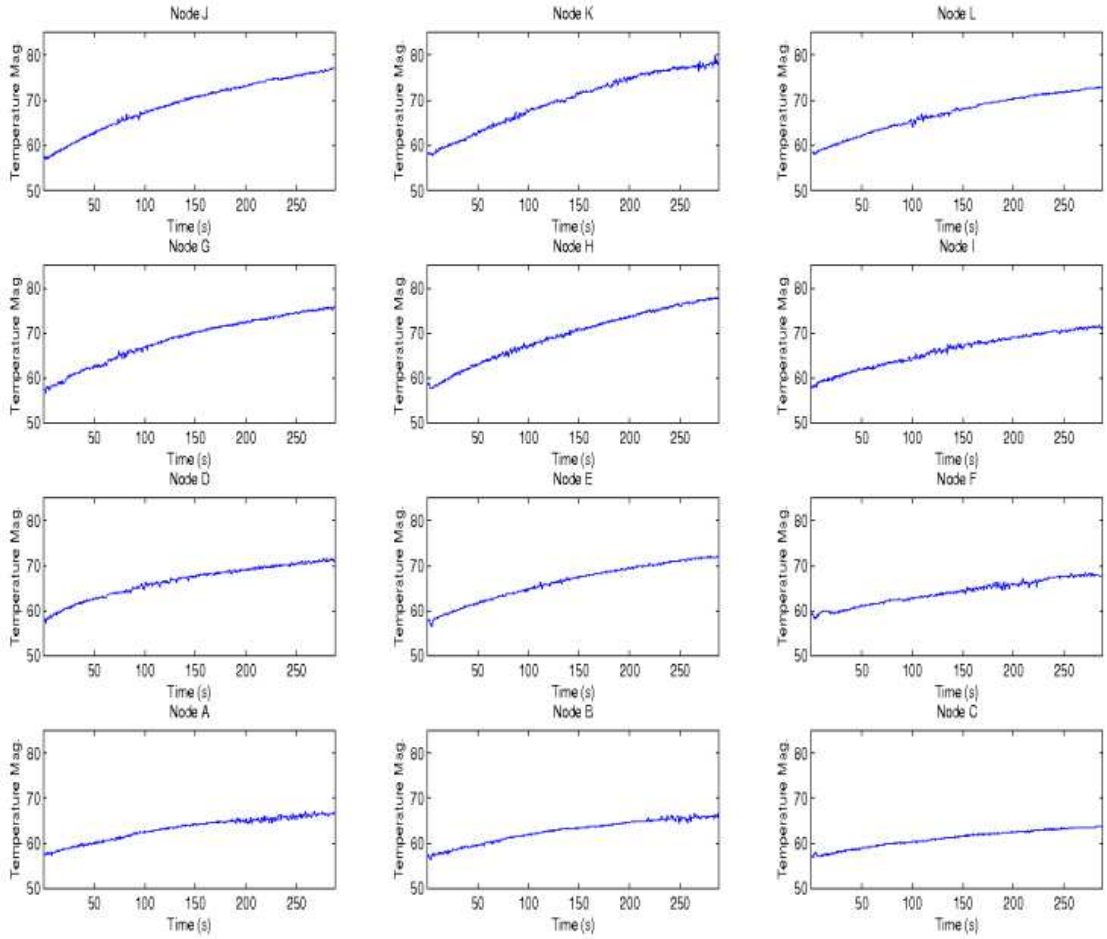

Figure 5-23: Filtered temperature response of twelve nodes arranged in a grid after an incandescent bulb is turned on very near to the top left corner of the grid.

while the blue in the lower right corner indicates lower temperatures. The fiugre itself emphasizes the spatial gradient our temperature sensors are able to detect.

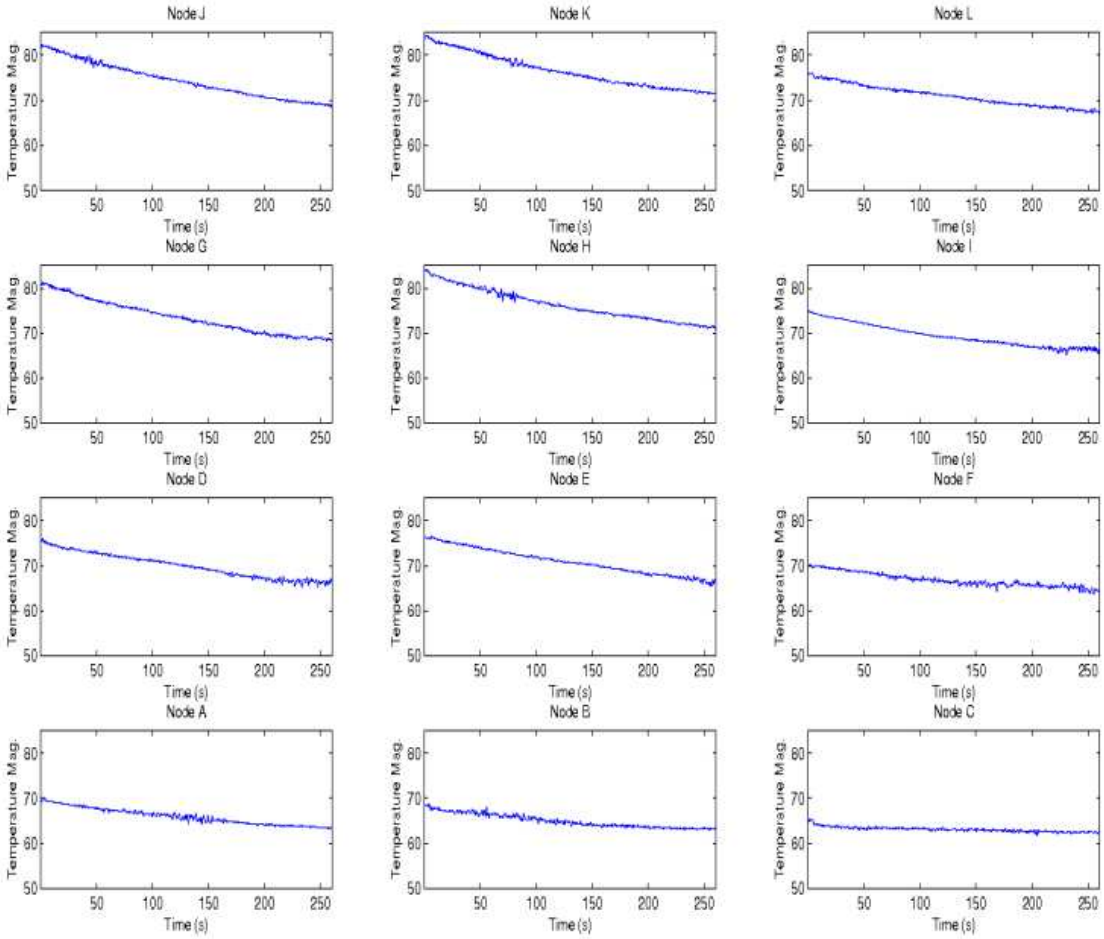

Figure 5-24: Filtered temperature response of twelve nodes arranged in a grid after an incandescent bulb that has been on for some time near the upper left corner of the grid is turned off.

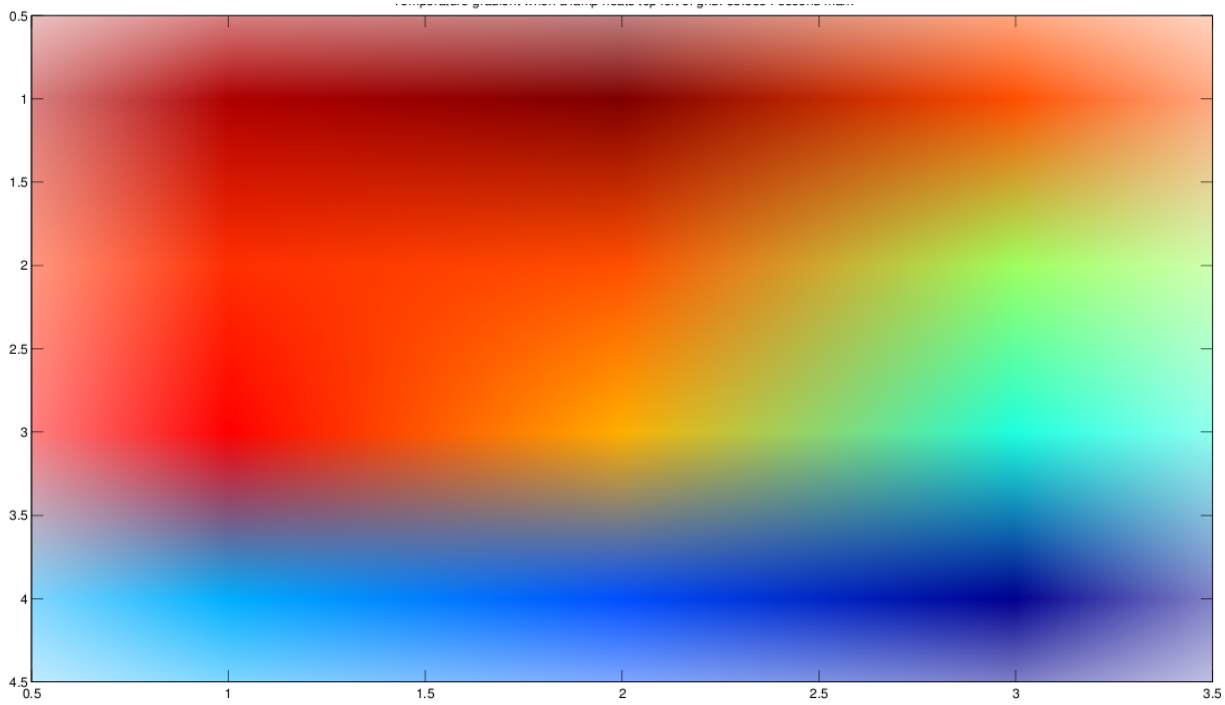

Figure 5-25: The spatial temperature gradient established by an incandescent bulb placed near the upper left corner of a grid of nodes. We performed a cubic interpolation on the grid of nodes to smoothe our discrete data. Red indicates higher temperatures while blue indicates lower temperatures. The figure itself is a snapshot of Figure 5-23's grid's temperature readings at the 83.8 second mark.

5.4 Extended Results

5.4.1 Proximity Events

Overview

S.N.A.K.E. and Tribble, the prior skin-related projects in our research group, promoted the importance of being able to detect proximity events - nearby stimuli that never triggered the pressure sensors. In line with this goal, in Section 5.3.2, we demonstrated our skin's ability to capture audio events. Our light sensor and whisker sensor also have the ability to detect stimuli that make little or no contact with our skin. This section presents data captured from proximity events, and focuses on information that can be deduced from these stimuli.

Edge Detection

Although acoustic waves from abrupt sonic transmitters have something of this character when they pass over our array [22], it does not make sense to think of certain stimuli, such as a loud sound, as having an "edge". However, other stimuli have very well-defined edges. In addition, as noted by Perez [33], detecting an event's edge may be very useful. This section shows how we can use our whisker sensor and light sensor to find the edges of proximity events.

Light

Changes in a shadow's size or darkness may indicate an object's approach or departure. Clearly, a single node cannot determine a shadow's edges, and, therefore, its dimensions. However, when we link several nodes together into a grid and sample the light sensor of each, we can find the rough boundaries of a shadow. Shadow edges on the network can be detected either through message passing between nodes on the skin or offline. In dark environments, this approach can be inverted - the light sensors can detect increasing reflection from the onboard LEDs off of approaching nearby objects.

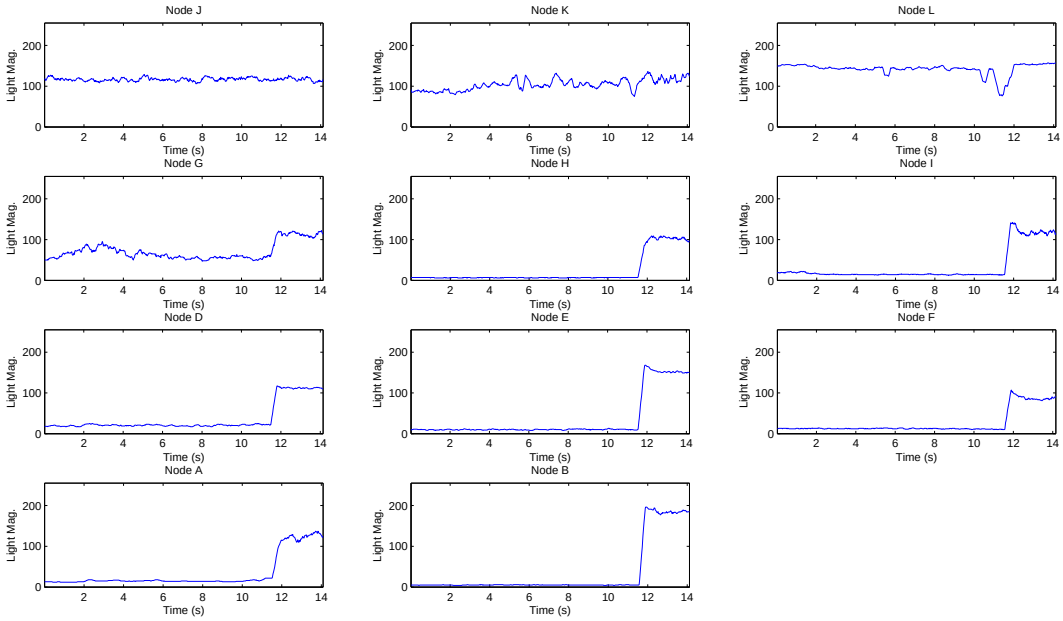

Figure 5-26: Eleven nodes' light sensor response to shadow. The eleven nodes are configured in an array as a hand casts a shadow over Nodes A, B, D, E, F, H, I and is then removed. Due to lighting angles, G is partially shaded.

Figure 5-26 presents the plotted data of 11 nodes arranged in a grid while a hand's shadow covers Nodes A, B, D, E, F, H, and I (the lower right-hand corner) and is then removed. Notice how distinctly different the shaded nodes' light sensor values are from the other nodes on the skin. These clear differences should allow us to find and present a shadow's area and bounds using a simple thresholding algorithm in our visualization program.

Whisker

Although limited to a closer range than our light sensor, our whisker sensor can also be used to detect the edges of proximity events. Figure 5-27 shows readings from eleven nodes' whisker sensors. While recording data, the bristles attached to Nodes B, E, F, H, and I (the right corner of the grid) were heavily tousled. Notice how distinctly different waveforms of the excited nodes appear compared to waveforms of the un-excited nodes. Again, because of the drastic differences in the waveforms, it

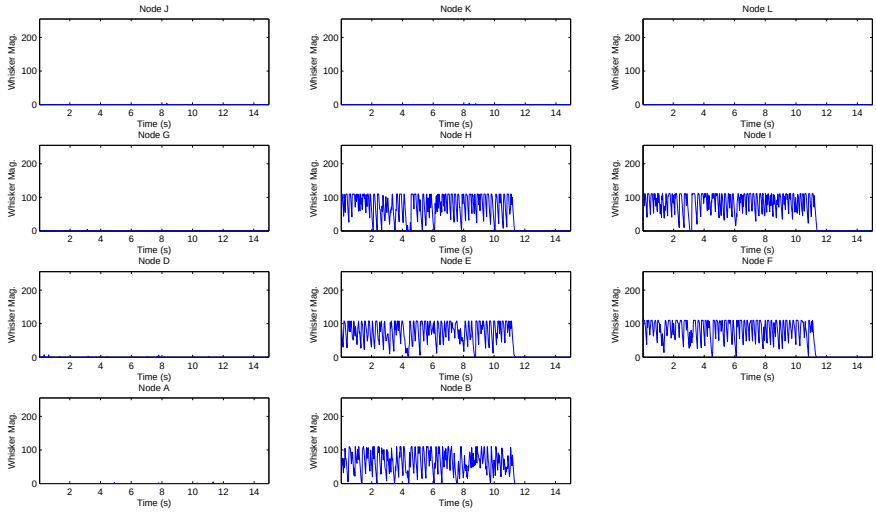

Figure 5-27: Whisker sensor waveforms of grid. Eleven nodes configured in a grid while a hand roughly tousles nodes *B*, *E*, *F*, *H*, and *I*.

should be easy to automatate detection of the shape and edges of the stimulus.

Hand Speed: Whisker

Intuitively, one might think that aggregating n nodes' worth of data would give us n times more information than a single node would provide. In practice, we find that we actually gain much more information than we expect. As a simple example, compare a set of nodes equidistantly spaced to a single node. When a hand runs across the whisker bristles of the single node, we know how intensely the hand interacted with the node's whiskers and for how long. In contrast, when a hand passes over a set of nodes, we can infer not only how intensely and quickly the hand interacted with each node, but also how fast the hand was traveling and in which direction it moved. Using the whisker sensor of each node, this section shows a proof of concept of this example.

To demonstrate how hand speed could be detected using our skin, we connected four nodes, *A*, *B*, *C*, and *D*, in a straight line. We then passed our hand through the whiskers of each node three times. The first and third time, we began our hand at Node *A* and then sequentially proceeded to Nodes *B*, *C*, and finally *D*. The second

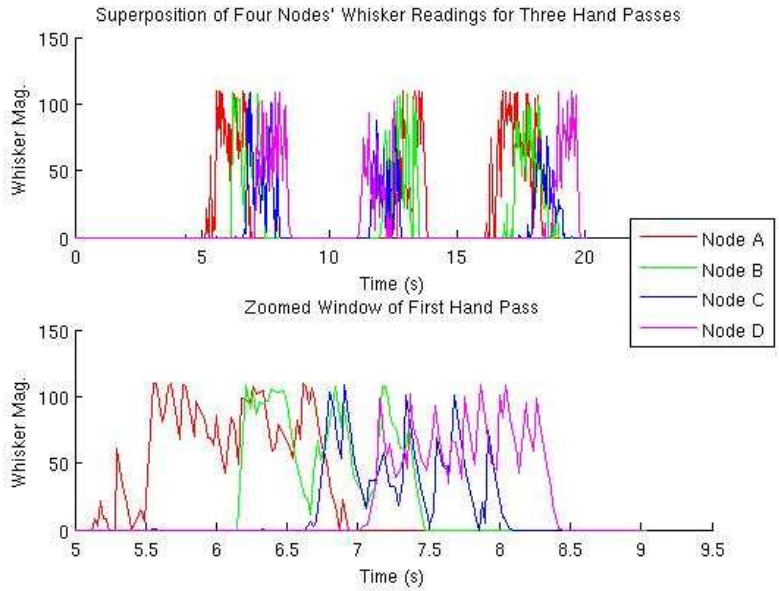

Figure 5-28: Whisker data from four nodes as a hand passes over each sequentially. Top panel: three hand sweeps in a row. Bottom panel: zoomed view of first hand sweep where one can see individual fingers passing over the whiskers.

time, we began our hand at node D and then sequentially proceeded to Nodes C, B, and A. We attempted to time each pass so that our hand moved steadily while taking as close to 3.5 seconds as possible to pass over the entire set of nodes. Because each node is 2.6 cm and each connector and cable was approximately 0.76 cm, this translated to a hand speed of roughly 3.5 cm/s. Figure 5-28 shows the whisker waveforms recorded by each node during the three hand passes in its upper panel. In its lower panel, Figure 5-28 expands the waveforms from the first handpass (we selected the first handpass because it most clearly matched our attempt to move our hand over the entire set of nodes in 3.5 seconds).

We can infer the speed of the hand from Figure 5-28 by comparing the times at which each node first detected a presence on its whisker sensor. These results are summarized in Table 5.5.

5.4.2 Magnetic Bend Sensor

We mounted six Hall effect sensors on each of our nodes. As described in Section 2.7, three sensors were mounted non-colinearly on the West side of our board and three

Node	Time at Which First Significant Stimulus Noted	Speed When Compared to A	Speed When Compared to B	Speed When Compared to C	Speed When Compared to D
A	5.3 s	–	4.1 cm/s	4.7 cm/s	5.2 cm/s
B	6.1 s	4.1 cm/s	–	5.5 cm/s	5.5 cm/s
C	6.7 s	4.7 cm/s	5.5 cm/s	–	6.6 cm/s
D	7.2 s	5.2 cm/s	5.5 cm/s	6.6 cm/s	–
Avg. Handspeed: 5.3 cm/s					

Table 5.5: Hand speeds calculated from whisker sensors.

sensors were mounted non-colinearly on the North side of our board; in addition, two neodymium magnet were placed on our nodes, one on the East side and one on the South side. The purpose of this arrangement was to allow our nodes to triangulate the positions of their neighbors based on the magnetic field.

Figure 5-29 presents the responses of all six of a node's Hall effect sensors as a magnet is passed at a distance of approximately half a centimeter parallel to the West and North edges of each node (so that the magnet's field is perpendicular to the of the board's edges). As can be seen from the plots, all Hall effect sensors detect the disturbance of the magnet, demonstrating both the functionality of our Hall effect sensors and the mux control circuitry used to direct our Hall effect sensors' outputs to our ADC (see Section 2.7).

Recalling Section 2.7, we used the approximation reproduced below to estimate θ from two of our Hall effect sensor's outputs ($Hall_1$ and $Hall_2$).

$$\theta = \frac{Hall_1 - Hall_2}{Hall_1 + Hall_2} \tag{5.1}$$

During preliminary physical testing, we found this approximation worked well to estimate θ when one moved a single magnet in the plane of our board, facing the board's edge. Running our Hall effect sensors at 3.3 V and reading values from our debugger, we could estimate θ to within 5^o or 10^o of its actual value for θ between -30^o and 30^o. (We could not find the radius with any real degree of accuracy, and therefore do not present these results.)

Encouraged by these results, we recorded data from twelve nodes' magnetic sensors

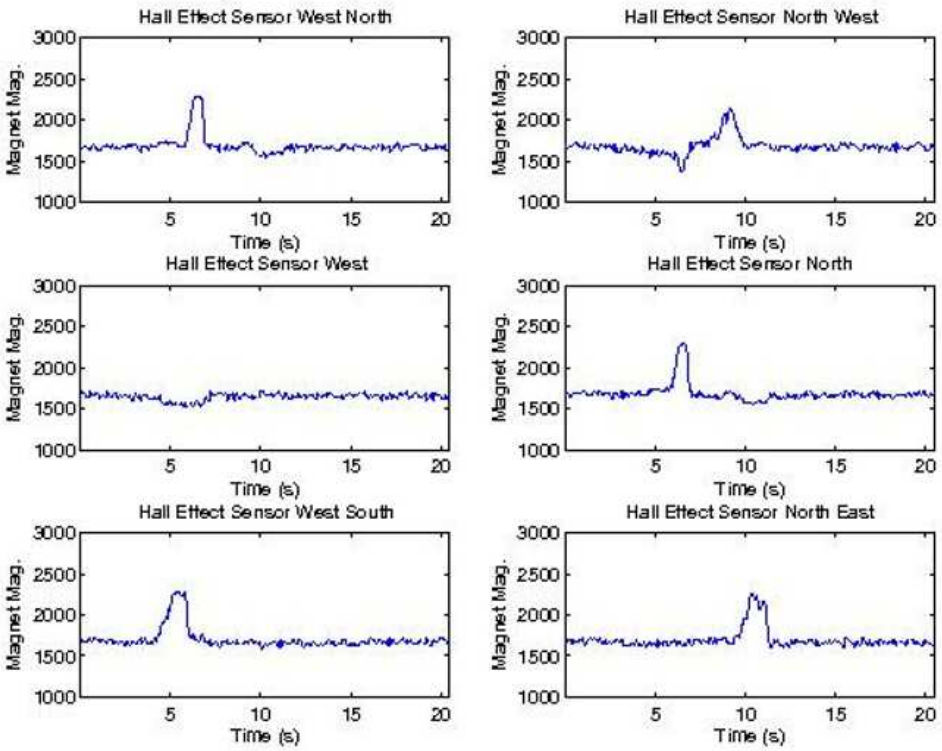

Figure 5-29: Responses of six Hall effect sensors as a magnet passes along the western and northern edges of our node, so that the magnet's field is perpendicular to our board's edges. The order in which the magnet passes our sensors is: West North sensor (bottom left corner of board), West sensor (middle left side of board), West North sensor (top left corner of board, sensor facing West), North West sensor (top left corner of board, sensor facing North), North sensor (middle of top edge of board), North East sensor (top right corner of board).

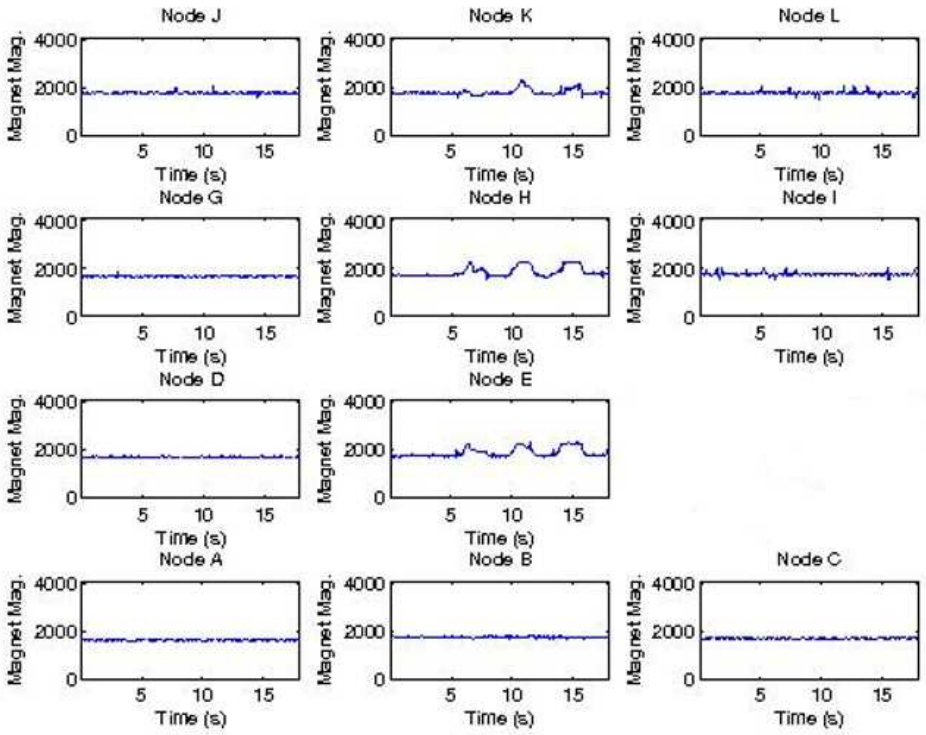

Figure 5-30: West North Hall effect sensor results as skin is bent to an angle of 60^o three times.

in response to various stimuli. We display the most relevant results in Figure 5-30. Figure 5-30 shows the output of our North West Hall effect sensor as our skin is picked up, bent to an angle of approximately 60^o, returned to form, again bent to an angle of 60^o, again returned to form, again bent to an angle of 60^o, and finally again returned to form. Notice that Nodes E, H, and K clearly record magnetic disturbances as a result of this stimulus. (Node B does not record these same disturbance. Due to our connector configuration, B was not physically linked to E, and therefore stayed fixed relative to its western neighbor, A, during this experiment.) Although a noticeable signal is seen, it is unclear whether any level of precision can be gleaned from this implementation of the magnetic sensor.

5.4.3 Hand Press

One of our principle benchmarks in performing this work was to get a clean set of data that captured a hand's approaching, pressing on, and releasing from a grid of our nodes. The following sections present data recorded by our skin for different hand gestures.

Basic Hand Press

We recorded sensor data as a hand approached, descended, pressed on, and pulled away from a four-by-three node grid. The light, whisker, temperature, microphone, and pressure data from this experiment are presented in Figures 5-31, 5-32, 5-33, 5-34, and 5-35 respectively. Because a hand is neither completely flat, nor uniform, the figures (particularly the pressure data presented in Figure 5-35) indicate that the response of the sensors is quite heterogeneous.

Progressive Hand Press

We need not limit ourselves to the basic hand gesture described in the previous section. While the hand gestures from the previous section deal with a hand's carefully and uniformly pressing on the entire skin or a section of the skin, we also have the ability to explore gestures that are more heterogeneous. Figures 5-36, 5-37, 5-38, 5-39, and 5-40 present light, whisker, temperature, microphone, and pressure data respectively from a hand's pressing on the left-most column of the skin, progressively rolling onto the left side of the skin, and finally rolling back. The non-uniformity of the hand stimulus can most easily be seen from the whisker responses in Figure 5-37. Notice how Node A's whiskers recorded excitation before Node C's. Initially, we thought that the pressure sensors on Nodes H and B may have broken. However, repeated tests confirmed that they were completely functional. Therefore, we concluded that the lack of response exhibited by these nodes' pressure sensors is attributable to the non-uniformity of the stimulus. Such an interpretation is confirmed by the audio plots of our nodes: if we consider audio events and pressure events correlated, we see

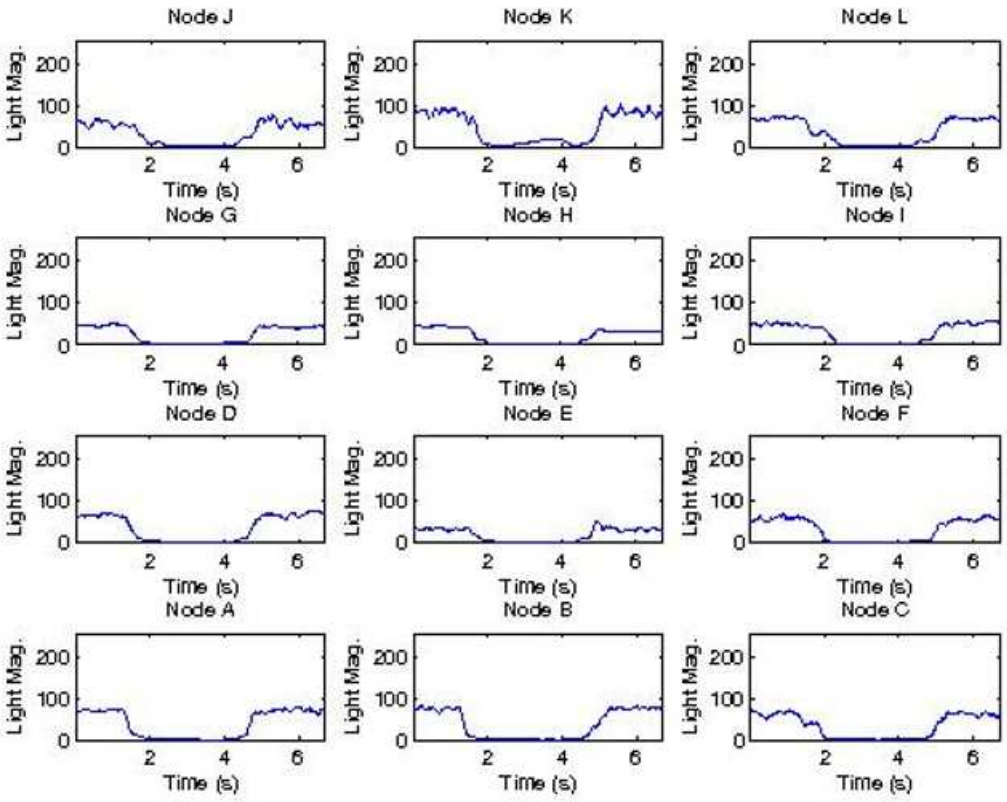

Figure 5-31: Light sensor data from a twelve node grid as a hand descends, presses on, and leaves grid.

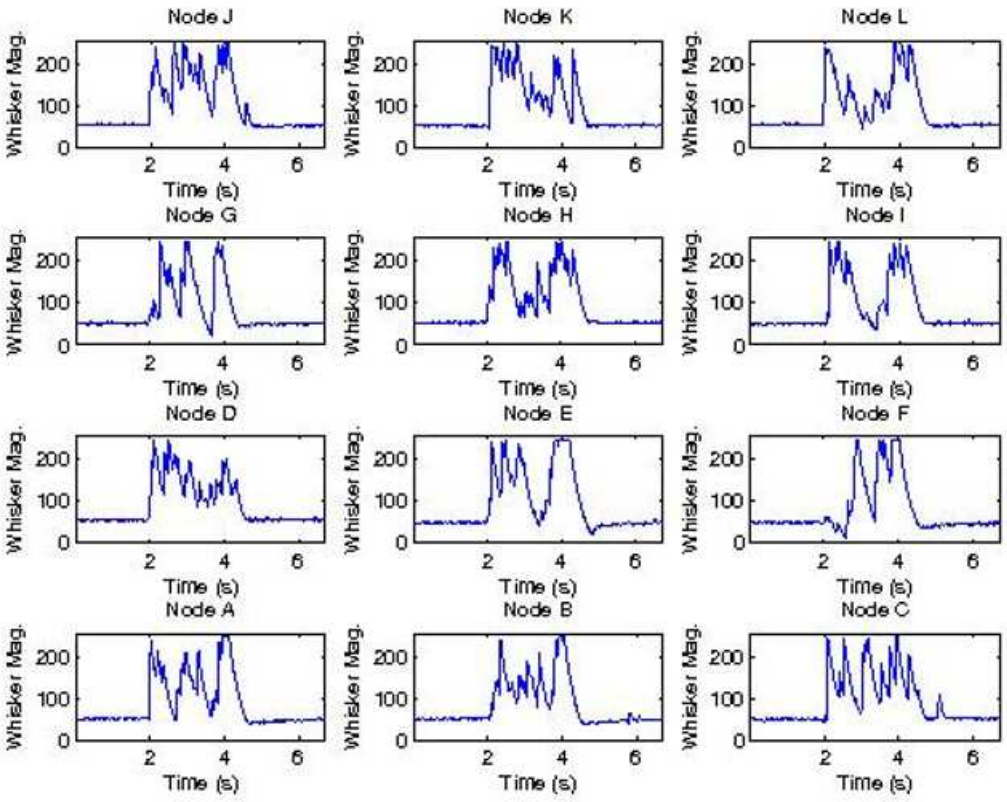

Figure 5-32: Whisker sensor data from a twelve node grid as a hand descends, presses on, and leaves grid.

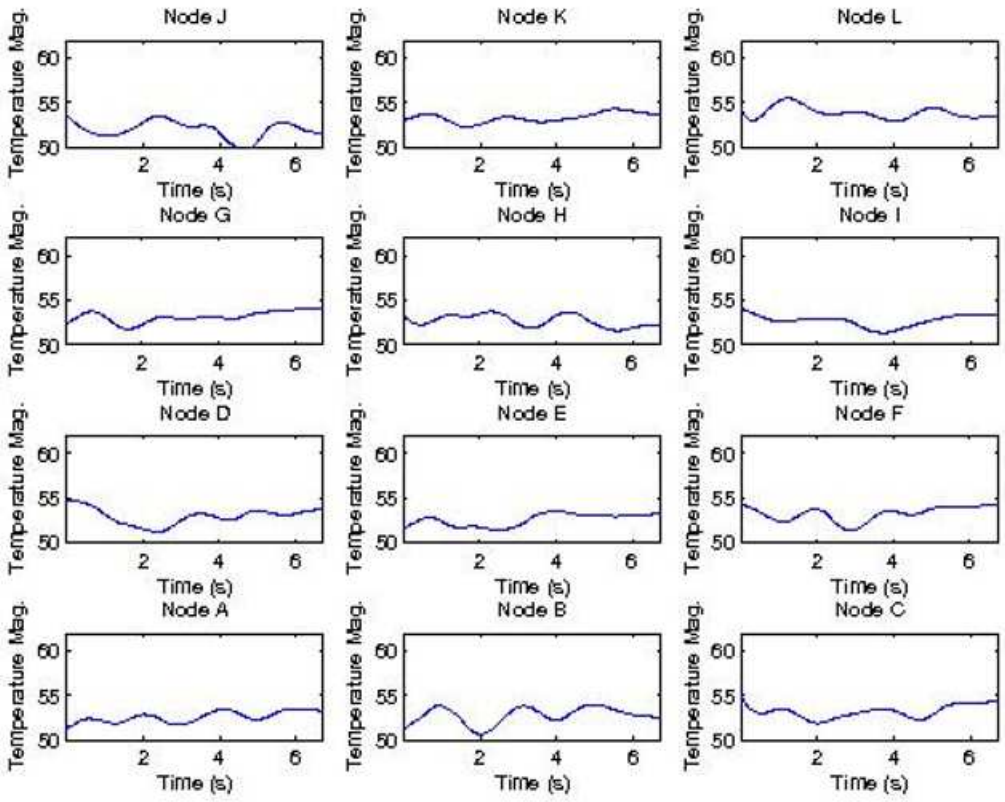

Figure 5-33: Filtered temperature sensor data from a twelve node grid as a hand descends, presses on, and leaves grid.

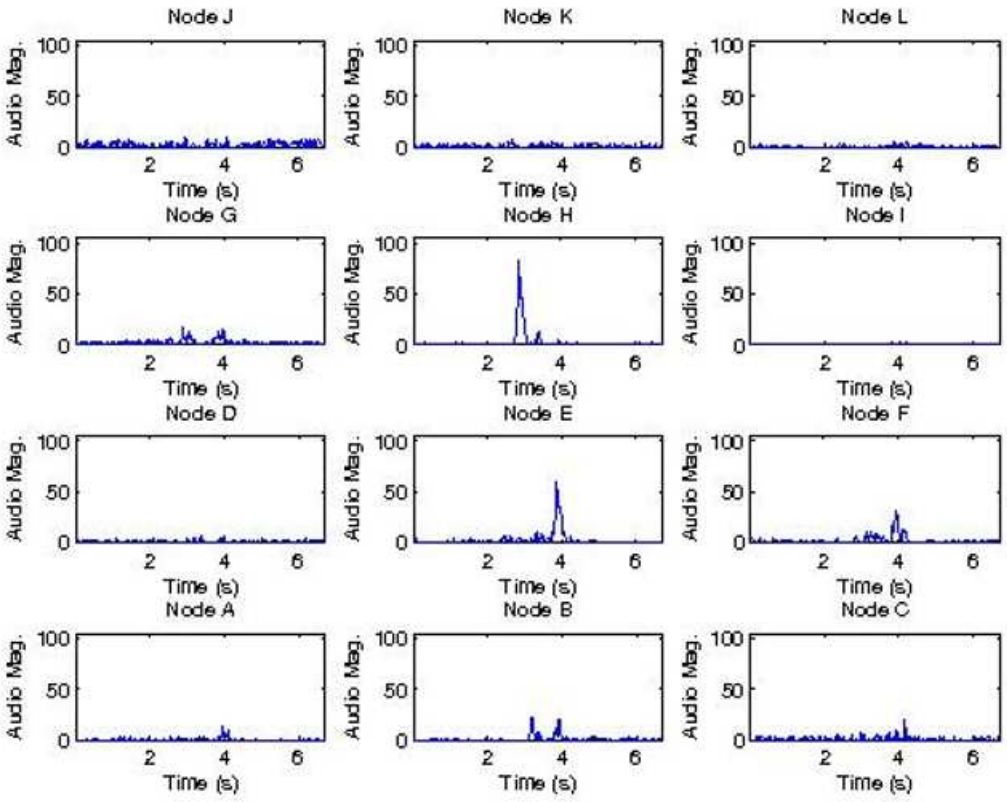

Figure 5-34: Microphone data from a twelve node grid as a hand descends, presses on, and leaves grid. Some nodes hear the hand brushing against them.

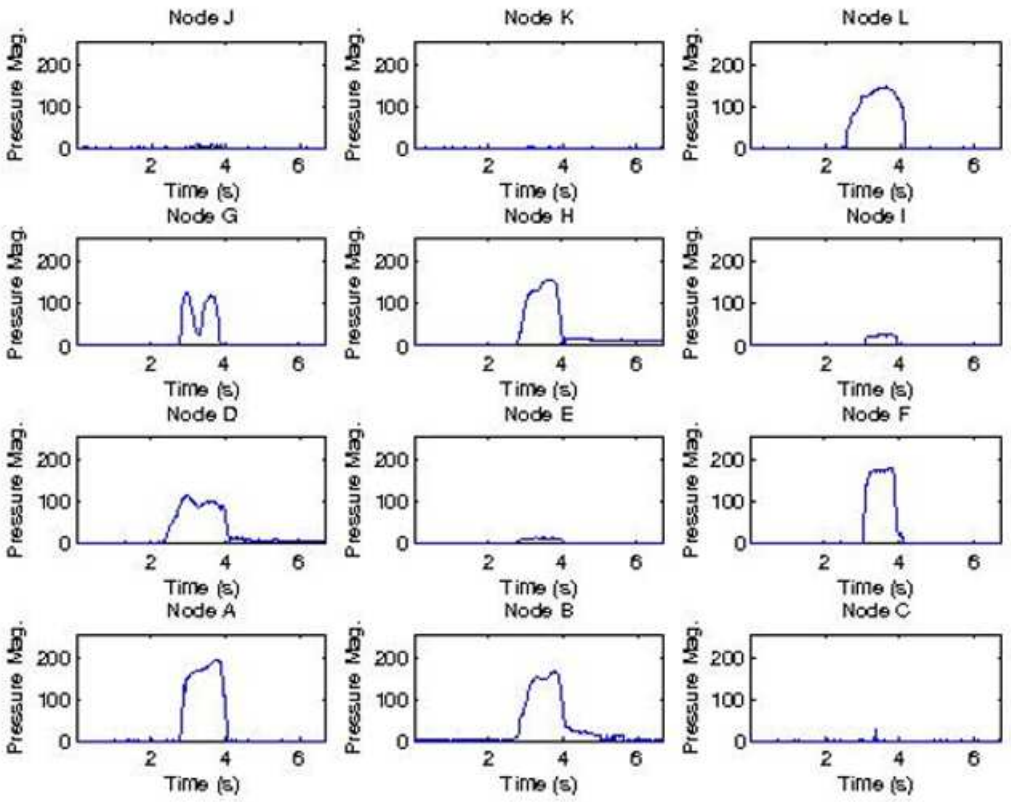

Figure 5-35: Pressure sensor data from a twelve node grid as a hand descends, presses on, and leaves grid.

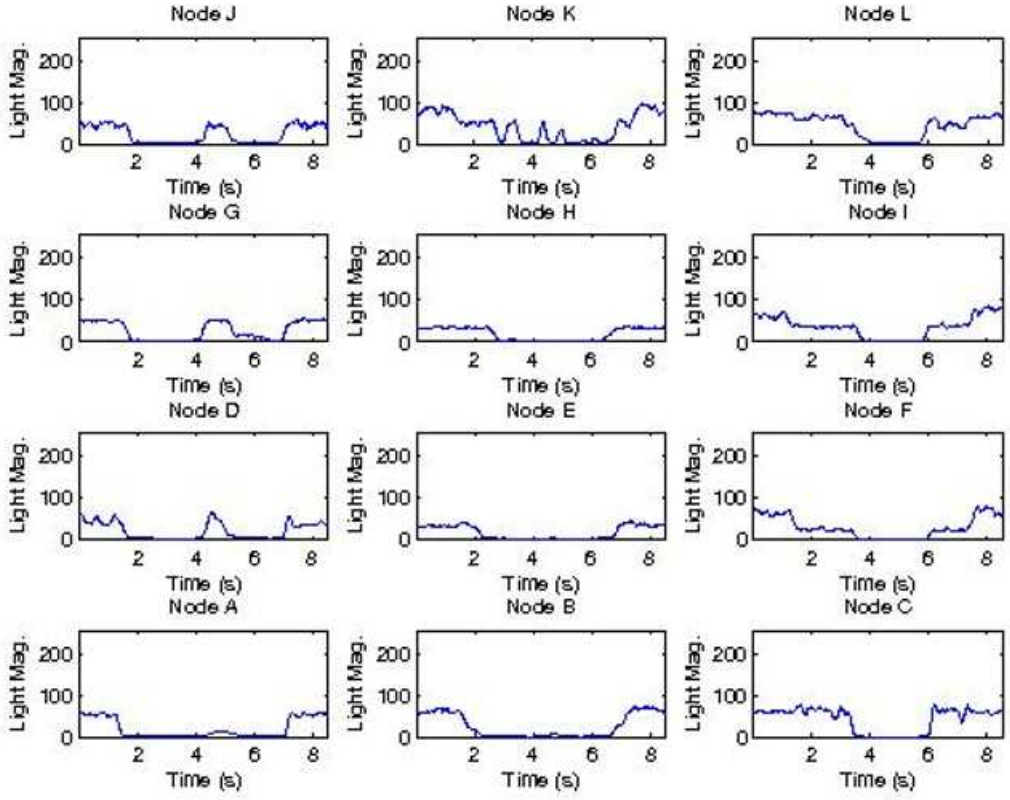

Figure 5-36: Light sensor data from a twelve node grid as a hand rolls from the left side of the skin onto the right side of the skin and back.

that Nodes B and H recorded little or no sounds or pressure at all.

5.5 Power Exploration

As presented in Section 2.10, in full operation, our nodes can consume up to 250 mW of power. For some applications, this figure may be excessive or even wholly unreasonable. Electrically, we can achieve substantial power savings by removing several sensors or LEDs from each node. However, this solution is hardly desirable - it reduces our skin's overall functionality and requires substantial effort to implement or reverse. Fortunately, there is another, more flexible option - we can dynamically throttle the rate of our microcontroller's clock and transition into one of the MSP430F1611's a low power modes. As explained in the MSP430F1611's datasheet, such steps can

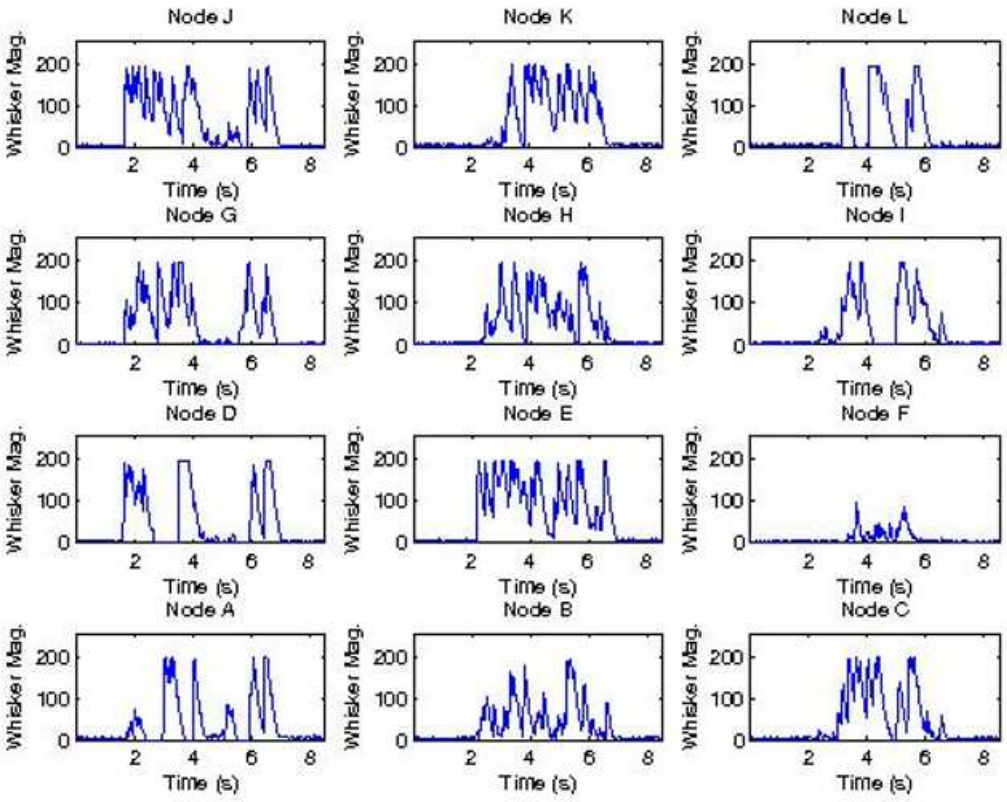

Figure 5-37: Whisker sensor data from a twelve node grid as a hand rolls from the left side of the skin onto the right side of the skin and back.

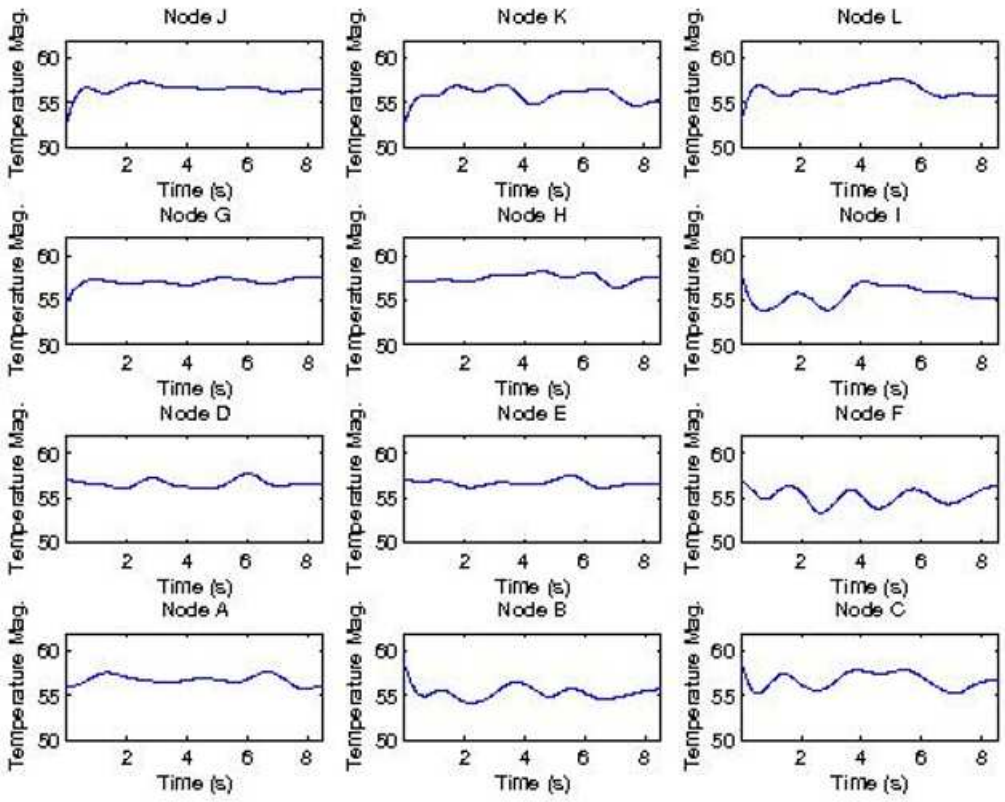

Figure 5-38: Temperature sensor data from a twelve node grid as a hand rolls from the left side of the skin onto the right side of the skin and back.

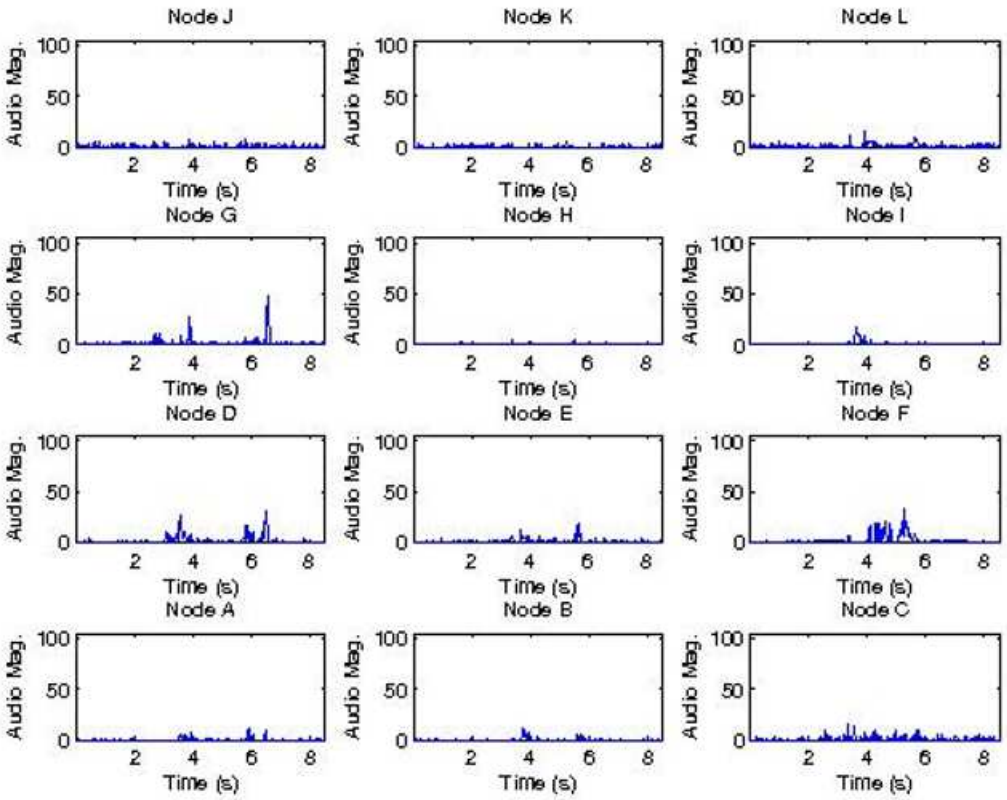

Figure 5-39: Microphone data from a twelve node grid as a hand rolls from the left side of the skin onto the right side of the skin and back.

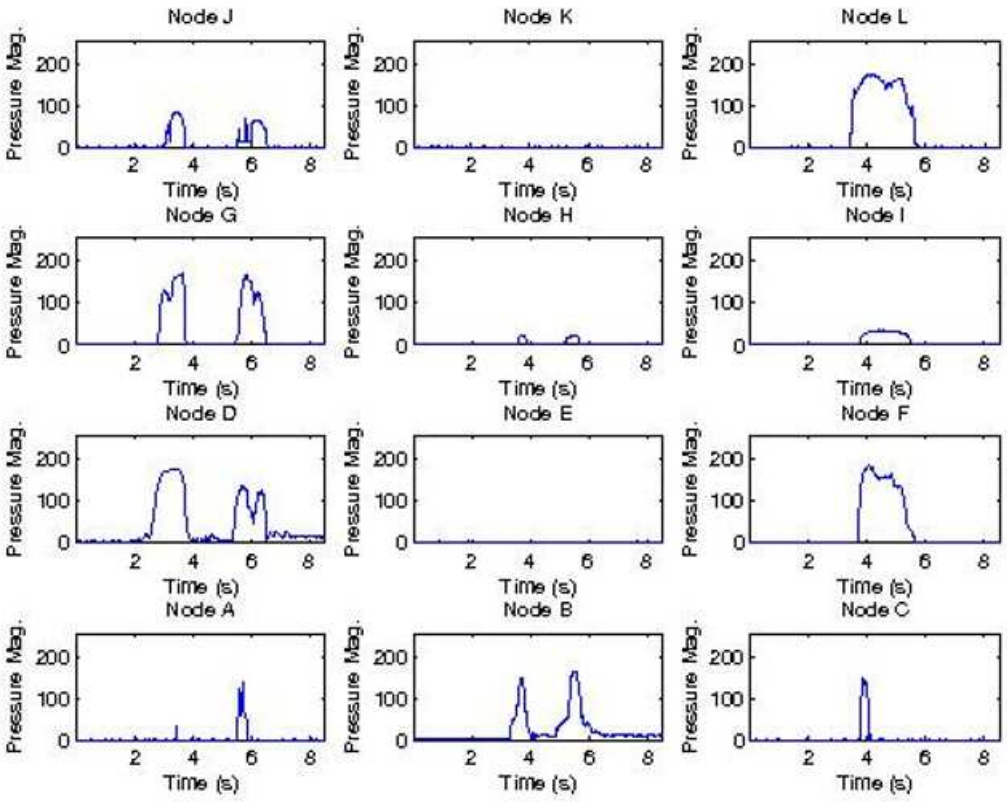

Figure 5-40: Pressure sensor data from a twelve node grid as a hand rolls from the left side of the skin onto the right side of the skin and back.

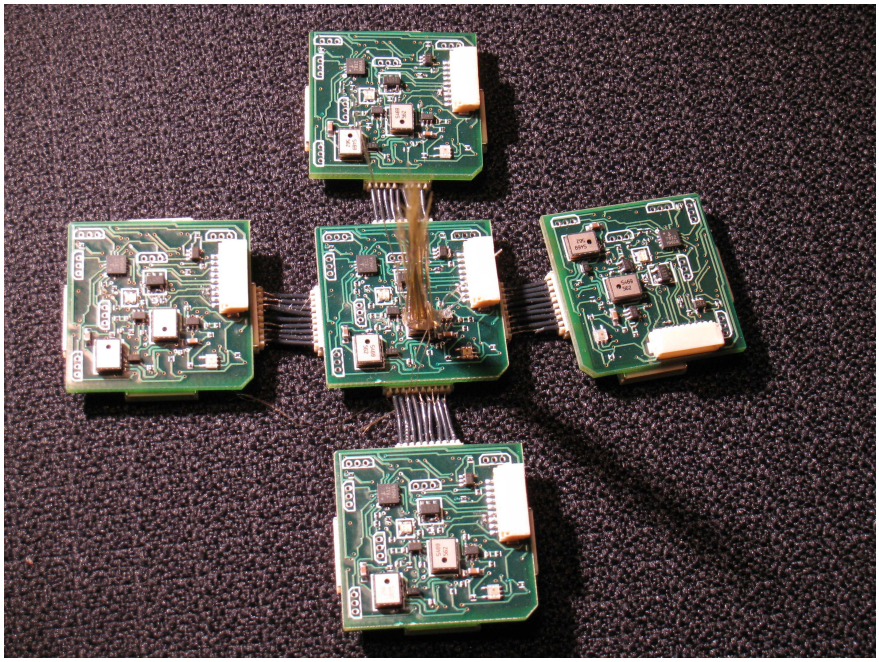

Figure 5-41: Node configuration for low power mode exploration.

reduce our microcontroller's power consumption by 50,000% [19].

We demonstrate the effectiveness of this strategy by connecting five nodes together in the shape of a '+' as seen in Figure 5-41 [4]. Nodes A, B, C, and D are located on the outer edges of this '+' and Node E is at the center. All nodes are initialized into their lowest power states. Additionally, we set an interrupt on Node E's whisker sensor. When E detects that its whisker sensor has been depressed three times, it sends a message via our custom-designed peer-to-peer protocol to A. Upon receiving this message, A switches into its highest power mode at its fastest clock rate. After receiving five, seven, and nine interrupts on its whisker sensor, E sends similar wake up messages to Nodes B, C, and D respectively. Additionally, after detecting nine whisker events, E switches from its low power mode to run at its most power hungry state. Table 5.6 presents the current consumption of the network composed of Nodes A, B, C, D, and E when undergoing this experiment. Notice how dramatically the power consumption of the overall network changes as nodes change their states.

[4]It is important to note that in order to highlight our results, none of the nodes' six Hall effect sensors were populated - if we had included these six sensors, they theoretically would have consumed an additional 48 mA.

While such a result provides insight into potential power regulation schemes on our skin, it also hints at possible future control mechanisms for our skin. Such control mechanisms could play a part in rejecting spurious stimuli, scoping processing, and regulating information flow.

Number Interrupts on Node E	Node A mode	Node B mode	Node C mode	Node D mode	Node E mode	Skin Patch's Avg. Total Current Consumption
0	Lowest Power Mode	Lowest Power Mode	Lowest Power Mode	Lowest Power Mode	Lowest Power Mode	5.144 mA
3	Highest Power Mode	Lowest Power Mode	Lowest Power Mode	Lowest Power Mode	Lowest Power Mode	7.558 mA
5	Highest Power Mode	Highest Power Mode	Lowest Power Mode	Lowest Power Mode	Lowest Power Mode	9.873 mA
7	Highest Power Mode	Highest Power Mode	Highest Power Mode	Lowest Power Mode	Lowest Power Mode	12.198 mA
9	Highest Power Mode	Highest Power Mode	Highest Power Mode	Highest Power Mode	Highest Power Mode	17.288 mA

Table 5.6: Current consumption from skin patch wakeup routine

Chapter 6

Discussion

6.1 Communication

6.1.1 Peer-to-peer Protocol

We did not make as much use of our peer-to-peer protocol as our I2C bus. However, the limited speed and accuracy results shown in Section 5.2.1, demonstrated our peer-to-peer protocol's viability as a stand-alone system. As presented in Table 5.2, our peer-to-peer protocol is blazingly fast, evincing baud rates of over 10,000 bits per second. To put this figure into perspective, at such a rate, we could transmit the readings of almost 1,000 sensors every second - more than adequate to capture basic stimuli. In addition to its speed, our peer-to-peer protocol is also highly accurate: referring to Table 5.1, we see that our peer-to-peer protocol exhibits no errors even when run at its fastest rate.

Additionally encouraging were our results in Section 5.2.1, which highlighted our peer-to-peer protocol's provision for symmetric control over throttling communication channels. In certain applications, it may be important for a node to prioritize foreground processing over communicating with its neighbors. As described in Section 4.3.2, we designed our protocol for this eventuality, allowing either a transmitting node or a receiving node to unilaterally reduce the baud rate of communication. Table 5.3 demonstrates the effectiveness of our implementation, showing a node suc-

cessfully emphasizing processing over communication. As can be seen from the table, as the receiver decreases the communication channel's baud rate, the time required to perform a pre-defined set of foreground operations goes down.

In the interests of full-disclosure, it should be mentioned that this experiment exhibits two potential sources of error, listed and addressed below:

1. **Non-uniform oscillators.** Due to manufacturing variations, the MSP430F1611's datasheet specifies only a range of frequencies over which we can expect our microcontroller to run [19]. Therefore, if we perform the same experiment on different nodes, we would likely encounter minor variations in our numbers. To reduce the effect of this variation on our results, we used the same microcontrollers for our transmitter and receiver across all runs of our experiment. Further, because our design does not explicitly rely on our microcontroller's clock rates, we anticipate that any two nodes selected to perform the same experiment would show similar trends, if not precisely the same numbers.

2. **Human response rate.** As described in Section 5.2.1, a human with a stopwatch timed each run of our experiment. As opposed to a mechanical or electrical timing system, this setup couples delays from human response rates into our results. To address potential variation in such delays, we took repeated measurements and averaged them, likely reducing the effects of delays significantly longer or shorter than the average. In addition, because the timing differences recorded in Table 5.3 are on the order of multiple seconds, it is unlikely that human response rates obscure the general trends presented in the table.

The most serious downside to our peer-to-peer protocol appears to be the number of connections it requires. As will be discussed in Section 7.1.2, requiring five connecting wires for our peer-to-peer protocol greatly reduced our skin's flexibility.

I2C Bus

As can be seen from the results presented in Section 5.2.2, our I2C bus ran quickly and efficiently. Specifically, data transmission from our skin's nodes to a listening

PC occurred with a baud rate of over 85,000 bits per second across our I2C bus, adequately fast to capture the numerous stimuli of interest presented in Chapter 5 and Appendix B. Even at its fastest rate, however the bus was reliable. As can be seen from Table 5.4, our I2C bus exhibited no errors in transmission - a stellar accomplishment considering the rate at which it runs and also taking into account the number of nodes on the bus. Most importantly, this result indicates that we do not need to build in any form error correction into our I2C bus. (Initially, we had prepared error correction code for our skin, which went substantially untested after these accuracy results. This code, as well as any thoroughly tested and debugged error correction code, reduced the speed at which we could transfer sensor information, making it unattractive if unnecessary for accuracy.)

The paragraph above highlights all our positive results with the I2C bus. Unfortunately in our work, we also encountered a curious problem when running our I2C bus. When I2C interrupts were enabled on our nodes, we observed a 10 KHz disturbance on our power and ground lines. (This disturbance may explain the noise seen on our temperature sensor's output in Figures 5-19 and 5-20 to our I2C line.) We have some difficulty fully explaining the origin of this noise, but posit that it arises from our I2C bus' SDA and SCL lines' being routed briefly through the edge of our analog ground plane. Signals on these lines may therefore have capacitively coupled to our analog ground plane, creating the noise that we see on our voltage supply. We are slightly puzzled by the frequency of this noise - our I2C bus runs at approximately 400 Kbits/s, much faster than the 10 KHz noise we observed.

In our early stages of prototyping, we did not notice this problem (we used an oscilloscope and debugger to check sensor states only when our I2C bus was not enabled, and only tested our I2C bus by conveying very basic messages), and therefore could not fundamentally address it in our skin's more final revisions. However, placing additional .1 μF bypass capacitors across sensors' power and ground rails did diminish this noise.

6.2 Light

Our light sensor generally performed well, providing much more information than we anticipated. For instance, Figure 5-10 from Section 5.3.1 shows basic responses of light sensors on our skin as a flashlight zooms in and out on Node *H*. As can be seen in the figure, several of the nodes' outputs vary substantially, exhibiting minor peaks and valleys. Originally, we considered these as evidence of either an unsteady hand holding the flashlight or incorrectly functioning sensors. However, after repeated and carefully studied experiments, we concluded that our light sensor was actually correctly detecting a phenomenon we had never noticed or considered. Specifically, due to the physical properties of a flashlight's reflector, light from a flashlight is highly non-uniform: at close ranges, a flashlight's beam exhibits a pattern of rings. It was these rings that our light sensors were detecting as they varied.

Of course, our light sensor provided insight into far more than just flashlight operation. If we examine Figure 5-36 which records our skins' light sensor responses to a hand's pressing down on the left side of the skin, rolling onto the right side of the skin, and then rolling back, we find that even such a basic stimulus is filled with detail. In particular, observe Node *K*'s light sensor response to the stimulus. As opposed to the values of light sensors below and to either side of *K*, *K*'s waveform contains several peaks and valleys. Repeating this experiment, we realized these peaks and valleys were caused by our middle finger's shifting across the node's surface several times as we rolled our hand, thereby obscuring and revealing our light sensor multiple times. While such a small event may have gone wholly unnoticed by even a very intent observer, our skin easily captures it in rich detail.

In addition, as noted in Section 5.3.1 we were able to distinguish the 60 Hz modulation from lights plugged into a wall socket. Although we did not write specific code that exploited this ability, it is conceivable that we would be able to distinguish roughly whether our skin was indoors (if our light sensor detected a signal with 60 Hz noise) or outdoors (if our light sensor detected a signal lacking such noise). Such an ability might prove useful if we found an application for our skin in the world of

context-aware computing.

6.3 Temperature

Our LM20CIM temperature sensor was advertised as having +/- 5^o C of accuracy. However, our temperature sensors' outputs never were in a range that could be construed as reasonable, indicating that a normal room was at almost 80^o C. In addition, as can be seen in Figure 5-19, our LM20CIM's output was very noisy, and required us to perform low-pass filtering that took up computational time and may have glossed over important events. Adding bypass capacitors immediately before our LM20CIM's power pin reduced noise in our sensor's output. However, we still were required to perform low-pass filtering.

In addition, the non-linearity of our sensor also introduced problems. The LM20CIM's data sheet specifies the following equation to convert between the sensor's output voltage, V_o, and temperature, T:

$$T = -1481.96 + \sqrt{2.1962 \times 10^6 \frac{1.8639 - V_o}{3.88 \times 10^{-6}}}$$

(6.1)

Due to the numerous floating point operations required by this equation, it was impractical to compute the temperature at our sampling rate onboard our microcontroller. Therefore, we experimented with two strategies: reducing the rate at which we sampled our temperature sensor and off-loading this computation to a listening PC. Each strategy had its downsides. Reducing our temperature sensor's sample rate destroys the sensor resolution which we value so much; off-loading the computation to a listening PC constrains our skin, preventing its being used as a stand-alone device. In theory, we could have also linearized Equation 6.1 around a room-temperature operating point. However, this approach would have impacted accuracy; and because

our sensor was outputting values well outside its expected range, may have proven unreliable. Therefore, we did not attempt any linearization.

Finally, although our choice of sensor performs admirably in detecting a heat gun (see Figure 5-19) or a thumb's being directly applied to it (see Figure 5-20), as seen in Figure 5-33, our sensor sees almost no change in response to a hand's pressing down on our skin. Some of our LM20CIM's lack of sensitivity is likely due to its placement on our boards: a nearby Hall effect sensor shields our temperature sensor from immediate physical contact. However, we attribute most of its insensitivity to poor sensor selection. The LM20CIM is rated for detecting temperatures between -55^o C and 130^o C. Across such a broad range of operation, it is hardly surprising that our sensor exhibits inadequate resolution when used across the much more restrictive range of room temperature. As will be described in Section 7.1.1, it would have been much wiser to choose an alternate sensor than the LM20CIM, such as a PIR.

6.4 Whisker

Recalling Section 2.5.2, we exerted quite a lot of effort in building our whisker sensor: we custom-designed and tested two novel implementations (gluing bristles to a PKGS piezo shock sensor and gluing bristles to an SPM0102NE-3 microphone), and evaluated the performance of a third type of sensor which relied on a piezo-film vibra-tab.

Designed to detect only proximity events, such as the presence of a hand or a strong air current, our whisker sensor performed admirably in these tasks. Consider for example Figure 5-32 which displays the outputs of our skin's whisker sensors as a hand approaches our skin. Notice that all our whisker sensors detect the hand's approach before actual physical contact is made (indicated by the times at which our pressure sensors start detecting an interaction), mimicking the role of hair on biological skin.

Our whisker sensor provided much more information than we anticipated. In the Huggable work cited earlier, Stiehl and his colleagues emphasize the importance of

distinguishing different types of stimuli [39], [40]. As demonstrated in Section 5.4.1 we were able to aggregate the information from a grid of whisker sensors to analyze the direction of travel of a stimulus as well as its intensity. Although we never explicitly used this information to characterize stimuli, it is completely conceivable that our work could be extended along this path.

In addition, due to the inertia of the paint brush bristles used in building our sensor, we were able to detect sudden movements of our skin. As demonstrated in Figure B-4, all of our whisker sensors react violently to a sudden tug on our skin's base, behaving almost like a large accelerometer. While it is unlikely that our skin, as currently constituted, will be deployed mobilely, future iterations of our work may be. In such a case, our new skin would be aware of the context in which it was operating, perhaps regulating its function depending on whether and how fast it was being moved.

While the above paragraphs provide a list of all the positive features of our custom-designed whisker sensor, we also encountered several failings. In particular, as can be seen from Node E in Figure 5-32, the high gain of our whisker sensor occasionally caused our sensor to saturate for heavy stimuli. This presents an interesting problem. For certain applications, such as detection of light or glancing stimuli, we appreciate the high sensitivity of our whisker, and therefore recoil at the idea of simply reducing our overall gain to prevent sensor saturation.

One potential solution to this issue might be to incorporate dynamically controllable gain. If we replace the standard 4.3 MΩ resistor shown in our whisker's conditioning circuitry in Figure 2-9 with a FET circuit that acts as a voltage controlled resistor, our microcontroller would be able to decrease gain if it detected our sensor's saturating. The only downside to this proposal is that it would likely complexify routing, add to component count, and potentially increase board area.

Another problem with our whisker sensor arose from its manufacture. Because we built our whisker sensors by hand, they were not exactly identical: some sensors had slightly more paint brush bristles than others, sensors' bristles were positioned at slightly different angles to others, etc. Recalling the controlling metaphor of our

work - biological skin - this issue may be less of a problem and more of an accurate imitation: biological skin's hairs are non-uniform as well. Some hairs on biological skin are larger than others, some hairs point in radically different directions, and some portions of the body have a lower hair density than others.

6.5 Microphone

After repeated tests with our microphone, we removed the 4 KHz low-pass filter described in Section 2-8: our microphone's output did not show any high-frequency noise and therefore all our low-pass filter did was obfuscate the sharp edges of audio stimuli. Other than this minor change to our conditioning circuitry, we were very pleased with our microphone. Despite its small size, it was able to detect stimuli at distances of several feet. In addition, as can be seen from Figure 5-34, our microphone was sensitive enough to detect sound from a hand's pressing down on our skin. As with our light sensor discussed above, our microphone provided unanticipated detail of this simple stimulus. Notice from Figure 5-34 how different the responses of each microphone are to a hand press: Node B, E, and H record large spikes in volume, while other microphones on our skin such as those of Nodes A, D, K, L, and I are barely excited. These plots therefore indicate a heterogeneity to the basic hand press stimulus that we wholly did not expect.

6.6 Pressure

While our project should be thought of as a rude approximation of biological skin, nowhere is the imperfection of our metaphor more apparent than in our pressure sensors. Skin on human fingers has up to 1,500 separate pressure sensors per square centimeter [38]. In contrast, our node has only three pressure sensors per square inch. As explained in Section 2.4, we were generally disappointed by our QTC sensors. Our QTC sensors were relatively insensitive to pressure stimuli: a strong stimulus that excited our FSR sensor across its full dynamic range, which is shown in top panel of

Figure 5-18, excited our QTC sensor to less than an eighth of our QTC sensor's full range, which is shown in the bottom pannel of the same figure (note the y-axis of the bottom panel only extends to 16 instead of up to 255 as the y-axis of the top panel does).

Because of this insensitivity concern, as well as the difficulty we experienced in mounting our QTC pills, which we discussed in Section 2.4, we focused instead on our FSR sensor. In contrast to our QTC pills, we were very satisfied with our FSR pressure sensors: as demonstrated in Figure 5-17, our FSR sensors tracked simple stimuli very well. Like our light sensor and microphone, our FSR pressure sensor also highlighted the heterogeneity of stimuli.

Figure 5-40, which captures the pressure responses of a skin's FSR sensors as a hand is placed on the left side of the skin, rolled to the right side, and then rolled back. Nodes H and B evince almost no response. Initially, we thought that this was a sign that the nodes' pressure sensors were broken. However, repeated tests confirmed that they were completely functional. Therefore, we concluded that the lack of response exhibited by these nodes' pressure sensors is attributable to the non-uniformity of the stimulus. Such an interpretation is confirmed by the audio plots of our nodes: if we consider audio events and pressure events correlated, we see that Nodes B and H recorded little or no sounds at all.

6.7 Magnetic Bend Sensors

Our magnetic bend sensor did not perform as well as we hoped: we were never able to accurately use our system of Hall effect sensors for fully accurate localization of neighbor nodes. This poor performance was not due to sensor failure: Figure 5-29 clearly shows each Hall effect sensor responding to magnets passed near them. Instead, we attribute our failure to our own limited physical understanding, as well as simplifying assumptions made we in the design, simulation, and prototyping phases of this project.

In particular, during simulation, we treated our magnet as a point-source. How-

ever, in actuality, our magnet is highly directional. Tests using our magnets showed that our Hall effect sensors only detected magnetic disturbances if a magnet was almost directly pointed at the sensor. In addition, during simulation, we assumed our sensors would have 32-bit resolution, and therefore would be able to detect even slight movements of our magnet. In reality, our ADC provided us with only 12-bit resolution, preventing us from detecting the minor changes in our magnetic requisite for accurate localization.

Further, we overlooked the effects of multiple magnets on our skin. Figure 5-30 shows readings from our West North Hall effect sensor as we repeatedly bent our skin along the axis along the intersection of Nodes *D*, *G*, and *J* and Nodes *E*, *H*, and *K*. Figure 6-1, below, shows the values recorded by our North West sensor in response to these actions. Although the magnets on the southern edges of our nodes did not move relative to their counterpart North West Hall effect sensor during this experiment, notice that the North West Hall effect sensor of *E*, *H*, and *K* all detected magnetic events. We attribute this response to the fact that our Hall effect sensors along the northern edge of our nodes were not truly orthogonal to the field produced by the magnets placed on the eastern edges of our boards. Therefore, changes in an eastern magnet's position will be detected by some of our northern Hall effect sensors as well.

Our final shortcoming in this phase of our work was that we satisfied ourselves with tests of our Hall effect sensors using a magnet only in the plane of the board, and carefully moved by hand. In reality, our skin never exhibited such close and careful motion: even simple interaction with our skin caused nodes to quickly bend at strange angles and form into strange topologies (as seen in Figure 5-4). Rather than assuming our results would easily translate from such gentle tests, we should have specifically tested our angular localization infrastructure under more strenuous conditions.

Despite all these shortcomings, as can be seen from Figure 5-30, our Hall effect sensors were able to coarsely detect some strong stimuli and exceptional events. Such functionality may be useful to waken nodes from low power modes in a manner similar to the work presented in Section 5.5 or to change node behavior.

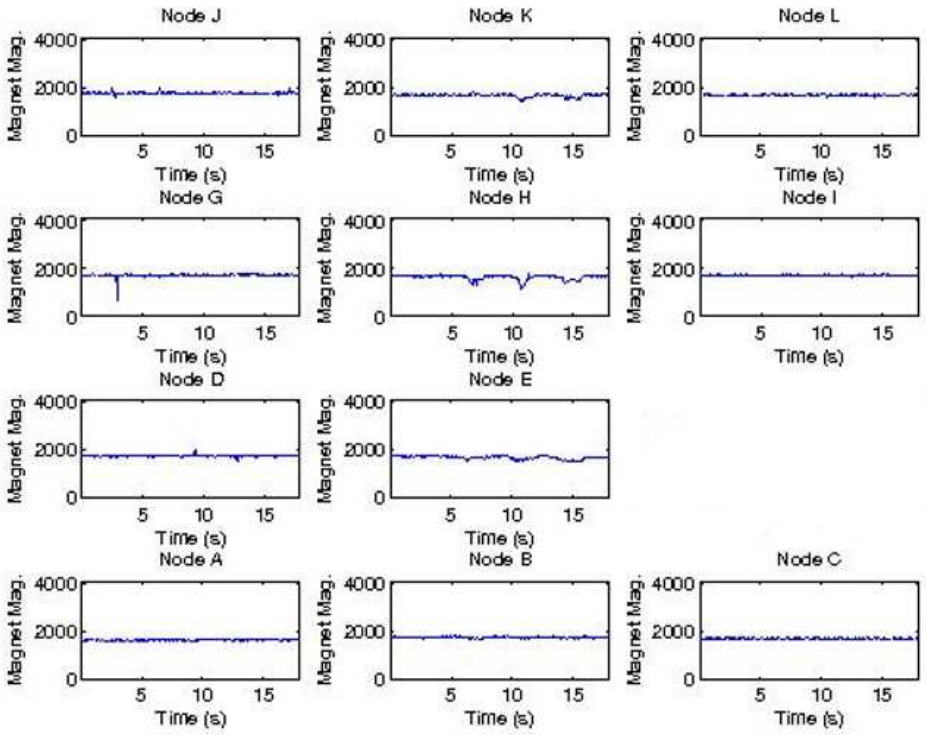

Figure 6-1: North West Hall effect sensor readings in response to skin's being bent between Nodes D, G, and J and Nodes E, H, and K. Figure demonstrates coupling between magnet placed on east side of boards and Hall effect sensors placed on northern edge of board.

Chapter 7

Conclusion and Future Work

This chapter is laid out in three separate sections. The first details solutions to problems encountered during the development of our skin and discussed in Chapter 6. The second section of this chapter suggests interesting and novel extensions of our work. And the last provides a brief summary of our skin, accomplishments, and failures.

7.1 Errata to be Fixed

7.1.1 Temperature Sensor

As noted in Section 6.3, our temperature sensor did not behave as we expected. Much of our difficulty we encountered was likely due to our sensor sensor selection. The LM20CIM is rated for detecting temperatures between -55^o C and 130^o C. Across such a broad range of operation, it is hardly surprising that our sensor exhibited inadequate resolution. Further, because of the non-linearity of our sensor, we had to offload computation of temperature values to a listening PC to prevent the required floating point operations from swamping our processor. For these reasons, in retrospect, it would have been a better decision to choose an alternate sensor for detecting temperature. In particular, a linear PIR or more sensitive temperature sensor would have performed better.

7.1.2 Connectors

Our connectors frequently broke. In our lab environment, this proved to be little more than an inconvenience. However, the frequent failures of our connectors would certainly limit our skin's functionality in any extended deployment. A highly attractive solution to this problem would be to transition our device away from our rigid-nodes-with-flexible-interconnects design to a fabric-based layout in which circuit components were mounted on a flexible fabric such as a cotton shirt and connected by conductive thread. Depending on our implementation of such a design, a fabric substrate might reduce the strain on interconnects between circuit elements while increasing the overall mechanical flexibility of our skin. Many researchers are working on fabric circuiting and interconnections that might prove useful if future iterations of this project are directed towards fabrics [9], [10], [24].

7.1.3 Hall Effect Sensors

In retrospect, our basic approach to bend detection was naive and cumbersome. Overall, the magnetic system we designed to track a node's neighbors performed poorly. In addition, the infrastructure for such a system is expensive. As explained in Section 2.7, we had to create a second power line, and deploy a mux to handle our six Hall effect sensors' outputs. This approach therefore took up a significant amount of board area on our nodes. We estimate that without our Hall effect sensors we would be able to shrink our board size by almost 30%. In addition, our six Hall effect sensors required a large amount of power. As detailed in Table 2.1, each sensor requires at least 6 mA of current. Operating at 5 Volts, our Hall effect sensors therefore consumed at least 180 mW - almost ten times the power consumed by an entire node without its Hall effect sensors mounted (and LED off).

For these reasons, we would encourage future iterations of this work to steer away from attempting bend sensing in a similar manner. A more fruitful approach might be to build such sensors into nodes' interconnections, perhaps using a flex circuit connector with a strain gauge similar to that described by Perez in [33]. Other

options could include using FSR bendy sensors or stretchy FSRs, similar to [29]. We initially rejected these approaches, deeming them too cumbersome and difficult to obtain. However, their documented reliability suggests that we should reconsider our initial assessment, and try to incorporate such a sensor into any future work.

7.2 Potential Extensions

7.2.1 Visualization

We do not need to constrain our understanding of our skin and visualization to be a one-way device. Just as easily as information can be pumped from our skin to an individual using our visualization, so too can information be pumped from an individual using the visualization to our skin. With this in mind, we built our visualization with an easy back-door that supported a user's sending messages to the master node of our skin. We never tested this functionality beyond getting nodes to echo back characters, but one might easily extend our visualization and node firmware to support more complex behavior. For instance, a user could direct segments of our skin to enter low power modes, or direct our skin's master node to focus its attention on a particular node.

7.2.2 Enlarging our Skin

The largest grid that we worked with had 13 nodes. However, due to our modular design, our skin can support additional nodes. The primary issues that we anticipate in scaling up our skin are:

Reduced effectiveness of our I2C bus. As mentioned in Chapter 4, adding nodes increases the parasitic capacitance on our bus. Eventually, the delay imposed by parasitic capacitance may require us to run our I2C bus at a lower rate, or abandon it altogether for our peer-to-peer protocol. Further, additional nodes imply additional information. Currently, the bandwidth of our I2C bus supports a nearly 30 Hz update rate for each node's sensor values. Adding nodes would reduce our effective update

rate.

Power. Our connectors were rated for a maximum current flow of 0.75 A. Depending on how many nodes we would add and how we connected our skin (in a grid, in a straight line, etc.), we might exceed our connector's maximum current rating. For instance, if we connected all our nodes in a straight line and ran them at full power, our skin would support a maximum of 15 nodes.

A larger skin would allow us to perform additional sensing and collect data on more extensive stimuli.

7.2.3 Distributed Control

We extensively researched different control structures for our skin, and took specific inspiration from game theoretic literature, particularly mechanism design. Mechanism design focuses on creating sets of rules so that individual agents in a multi-agent system autonomously choose to perform actions that maximize the entire system's utility. Needless to say, such work could be an enormously powerful tool in developing our skin. Our one master, multiple slave architecture is an artifact of using our I2C bus. Because we also designed a peer-to-peer protocol for communication, we can run our skin so that all nodes behave autonomously, choosing which sensors to sample, what power mode to be in, and which neighbors to communicate with. In such a setting, insights from mechanism design might permit our skin to evince positive aggregate behavior while each node maintained its autonomy.

We formulated our inquiry as a question of sensor selection: based on which sensors a node, A, was sampling, which sensors should its neighbor, B, sample to provide the maximum amount of non-overlapping information?

We hoped to prove the existence of either a pure or mixed strategy set of Nash Equilibria that could be arrived at simply (either through iterated strict dominance or through an efficient bargaining scenario).

A Nash Equilibrium is a state in which no agent in a game can benefit from unilaterally swithing its choice of strategies [15]. As opposed to a pure strategy

equilibrium in which an agent's best decision is to always play a particular strategy, a mixed strategy equilibrium is a probability distribution mapping across all available actions. This mapping indicates the optimal frequency an agent should play particular strategies [15]. We treated each node on our skin as a separate agent. Each node's action space (the strategies which each could play) was composed of 16 separate strategies:

1. Sample temperature sensor.
2. Sample pressure sensor one.
3. Sample pressure sensor two.
4. Sample pressure sensor three.
5. Sample light sensor.
6. Sample West South Hall effect sensor.
7. Sample West Hall effect sensor.
8. Sample West North Hall effect sensor.
9. Sample North West Hall effect sensor.
10. Sample North Hall effect sensor.
11. Sample North East Hall effect sensor.
12. Sample microphone.
13. Sample whisker sensor.
14. Send message to northern neighbor.
15. Send message to southern neighbor.
16. Send message to western neighbor.

17. Send message to eastern neighbor.

18. Sleep/no action.

We regarded the probabilities put on pure strategies in a mixed strategy equilibrium as a "sensor schedule", in which a node would cycle through the weighted sensors at a frequency proportional to the probabilities with which they were weighted. Such a sensor schedule would dynamically adjust and react to changes in the skin's state.

As a simple proof-of-concept, we were able to construct a limited two-player game with naively chosen utility functions and complete information such that we obtained a mixed strategy equilibrium. However, we had difficulty extending our results as we went from a two-player game to an n-player game and relaxed assumptions about how much information A knew about B's sensor selection and vice versa.

Arslan, et. al suggested an approach for solving a similar assignment problem in [3]. We re-formulated Arslan et. al's basic problem statement from a hypothetical scenario in which multiple vehicles bargain to determine an optimal set of assignments to destroy enemy targets to a less bellicose scenario in which each of our nodes was selecting the order and frequency with which to sample its sensors. However, early on we ran into mathematical resistance. Further complicating matters, due to the amorphous application space of our skin, we had difficulty clearly specifying our skin's global utilities. Unfortunately, due to time considerations, we were unable to progress past these obstacles. However, we hold great hope that future iterations of our skin will address and overcome them.

7.2.4 Power

We primarily addressed power concerns in Chapter 2 with our component selection. However, in Section 5.5, we briefly explained and showed an alternate method for reducing our skin's power consumption. Specifically, we demonstrated our nodes' ability to "sleep" in low-power states and quickly wake when receiving peer-to-peer messages. This simple example only touched on a rich vein of future research. Algorithmically, there may be ways to intelligently put to sleep certain sections of our

skin. For instance, a period of relative tranquility in the environment might require only 42% of the nodes to be actively sensing and processing information. The remaining 58% of nodes would either put themselves to sleep or be directed to sleep by their neighboring nodes - greatly reducing the overall system's power consumption for that period of tranquility. Of course, given a suitable stimulus detected by those 42% of awake nodes, the awake nodes might wake some or all of the sleeping nodes. Selecting which nodes to wake and sleep to ensure the sensing, processing, and communication of a suitable amount of information would be a non-trivial task that could show results that generalized to a host of other sensor networks.

7.3 Summary

Our work was concerned with building and deploying a sensate skin with embedded processing. This thesis demonstrated an adolescent system. Overall, we met most of, but not all, the goals of our project, which we laid out in our preliminary chapter. The skin that we developed showed good flexibility for a prototype, ran quickly and efficiently, and could detect and respond to a variety of stimuli. However, as noted throughout our work, our skin does evince troubling issues that should be addressed before this project can be considered a success. Particularly concerning were noise on our power and ground lines engendered from running our I2C bus, the unreliability of our temperature sensor, the frequent breaking of our connectors, and the poor performance of our magnetic bend sensor. The early sections of this chapter suggested potential solutions to these problems and further inquiries that our system might support.

Appendix A

Simulation Code

A.1 Skin Simulation

Before we began construction of our skin, we wrote a large body of code to test our project's basic algorithmic framework. This code supported peer-to-peer message passing, as well as a bus communication protocol. Figure A-1 demonstrates the visual output of our simulation as several nodes nodes on a grid are put under shadow.

We used this code only in a limited fashion because it was written with the assumption that all nodes on our skin would be synchronized. Nevertheless, it provided a good starting point for considering design approaches for our skin's aggregate behavior.

A.2 Magnet Simulation

As mentioned in Section 5.4.2, we wrote a simulation program to determine the feasability of angular localization using our Hall effect sensors. This simulation treated our magnets as point sources of magnetic field, and assumed our Hall effect sensors had 32-bit resolution. Even with these shortcomings, we still could perform minor algorithmic tests to determine how quickly and accurately different algorithms for computing magnet positions. Figure A-2 shows the visual output of our magnet simulation.

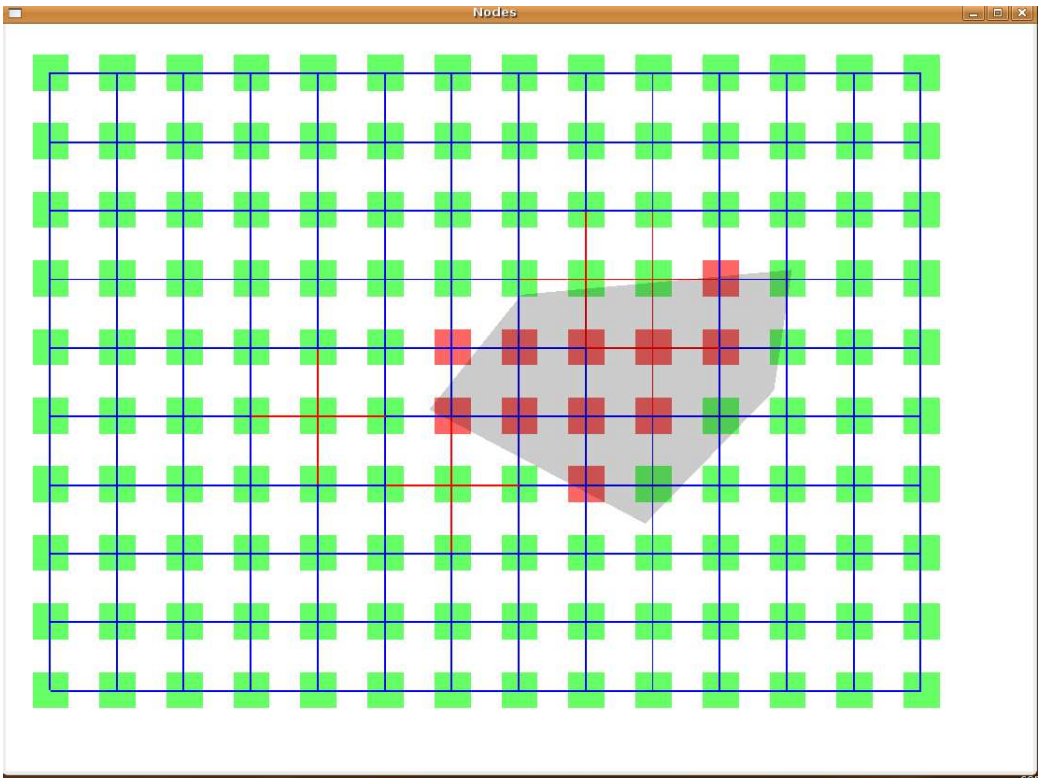

Figure A-1: Visual output of skin simulation program. Individual nodes look for and report being under shadow.

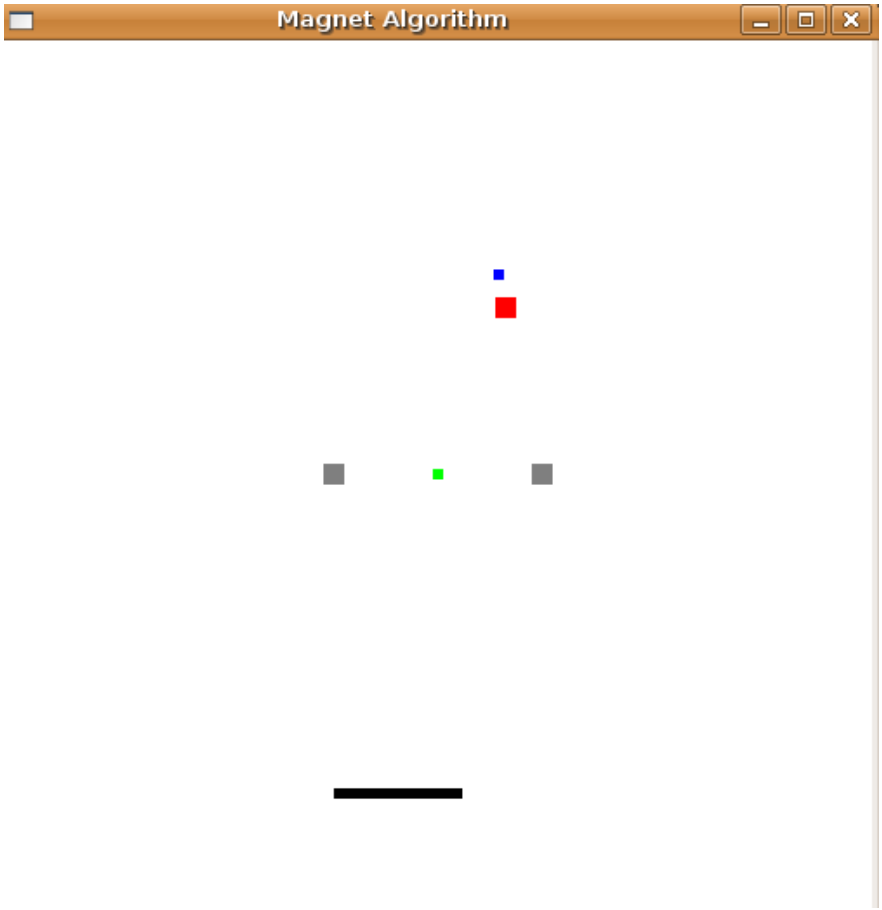

Figure A-2: Visual output of magnet simulation. Large box indicates actual magnet position, small box north west of it indicates calculated magnet position; three horizontally colinear boxes are two Hall effect sensors and their midpoint.

Appendix B

Additional Data

We applied several atypical stimuli to our skin. For some of these stimuli, the responses of our whisker sensor and microphone were particularly interesting. This appendix presents those results.

B.1 Table Pound

If we place our skin on a table, our whisker sensors can detect a closed fist's pounding down on the table surface, as can be seen in Figures B-1 and B-2.

B.2 Stomp on floor

Our skin, when placed on a table, can detect sudden events in its environment. For instance, Figure B-3 presents our whisker sensor's response to a foot's stomping on the nearby floor twice. In this way, we see that our whisker sensors can be used almost like mini-seismometers.

B.3 Skin Yank

When our skin is subjected to a sudden force or tug, our whisker sensors easily detect it, as demonstrated in Figure B-4.

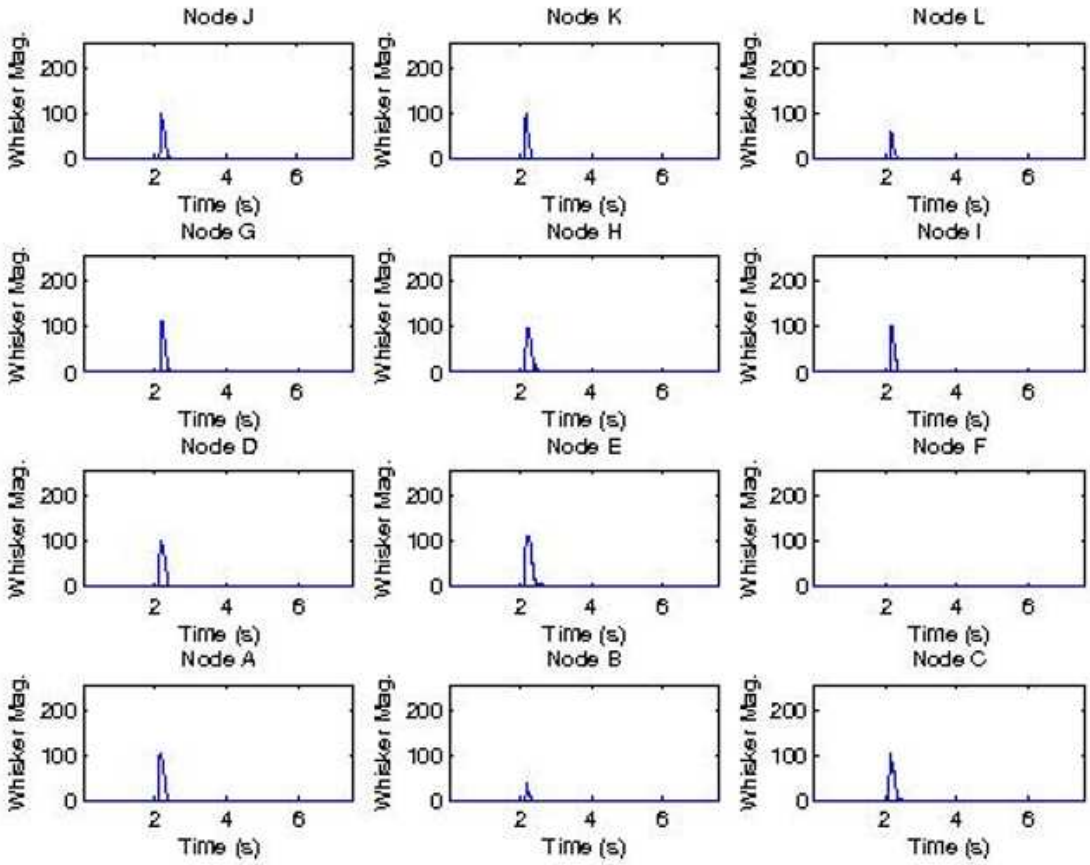

Figure B-1: Whisker sensor's response to a fist's pounding the table on which our skin is placed.

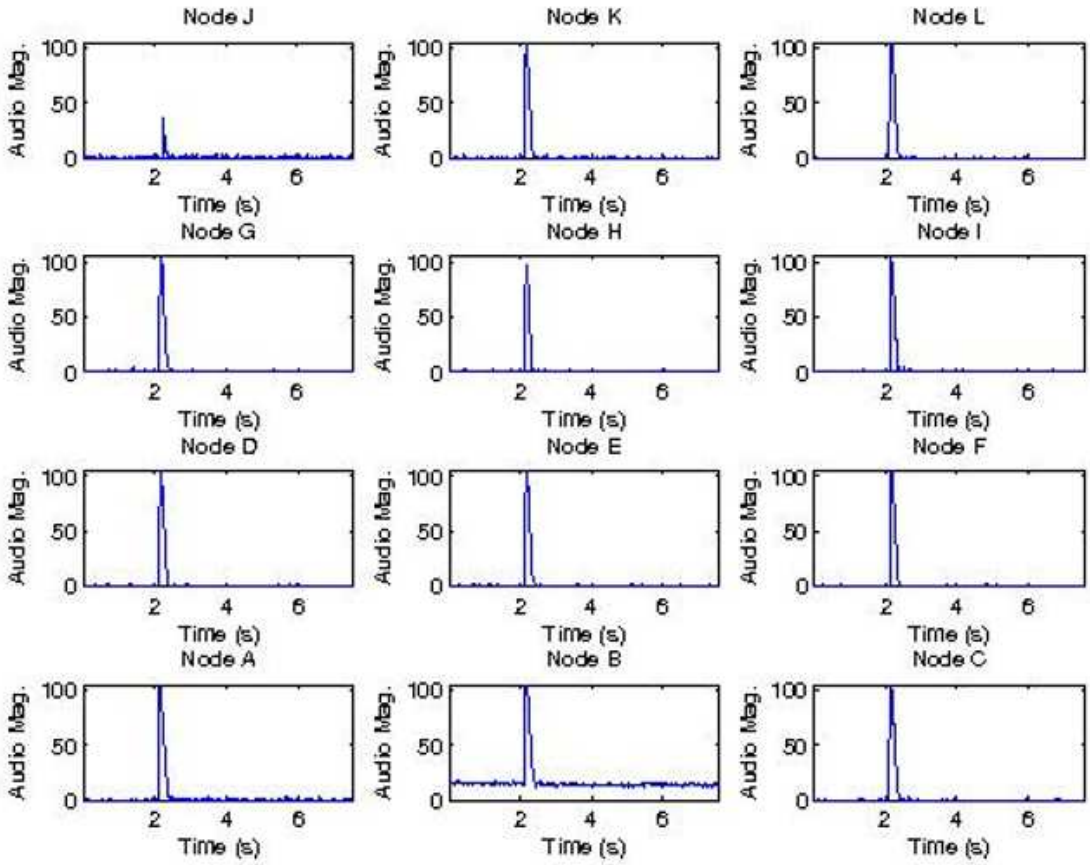

Figure B-2: Microphone's response to a fist's pounding the table on which our skin is placed

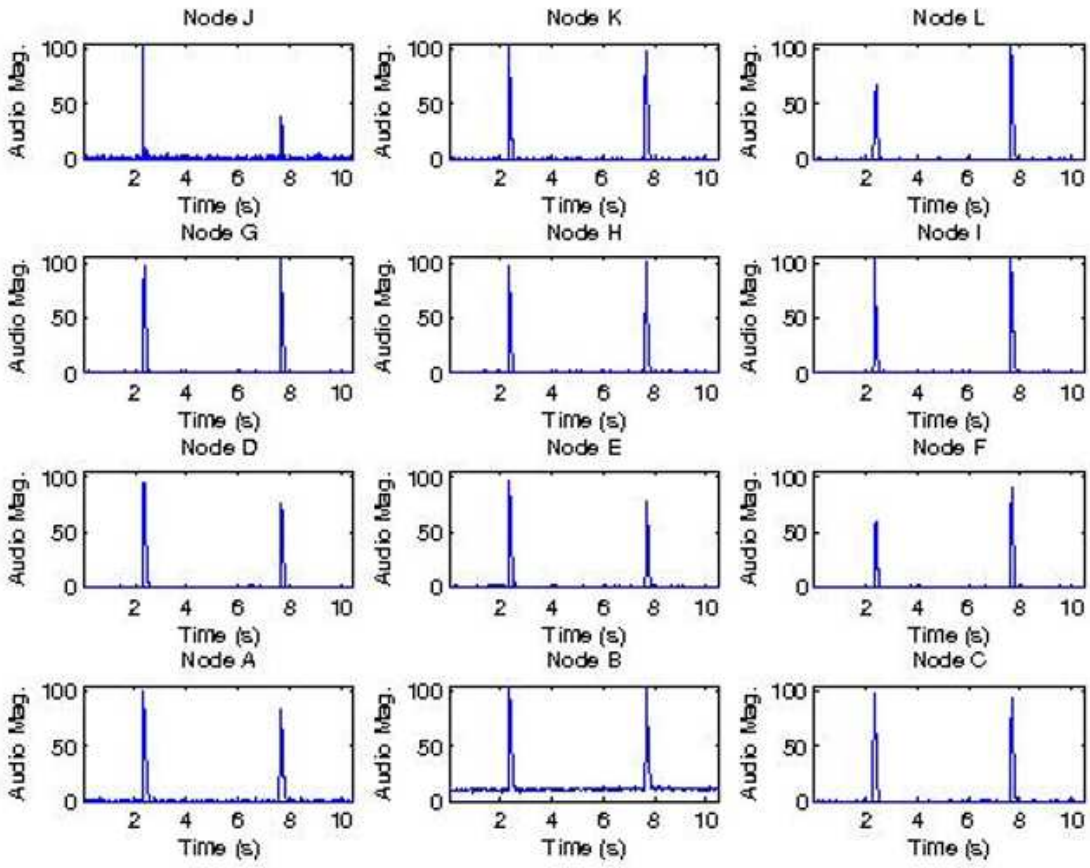

Figure B-3: Whisker sensor's response to a foot's stomping the floor near our skin twice

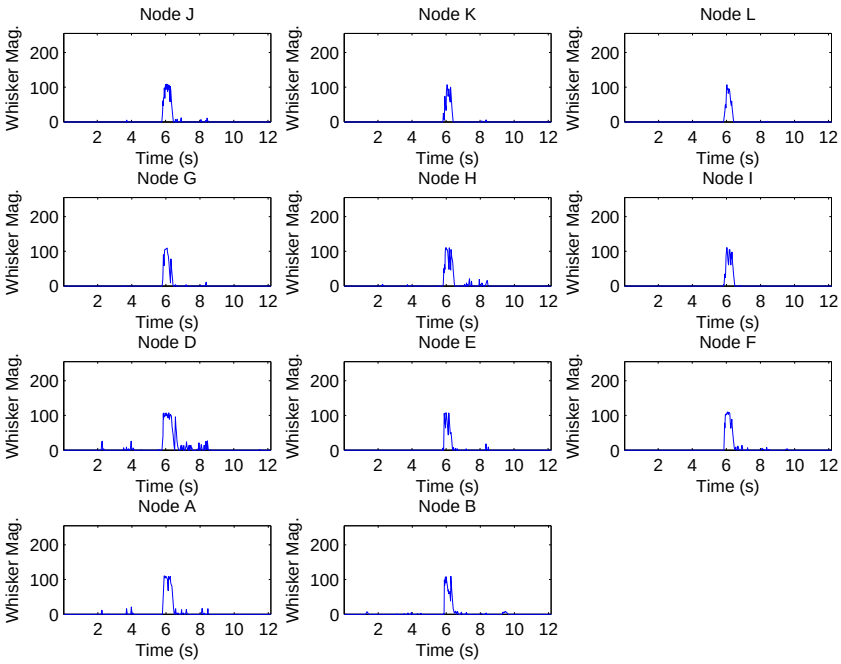

Figure B-4: Responses of whisker sensors to a sudden tug on the base of our skin.

B.4 Whisker Blow

Our whisker sensors are not only useful for detecting hand movements (Section 5.3.3) and sudden shock events (above).

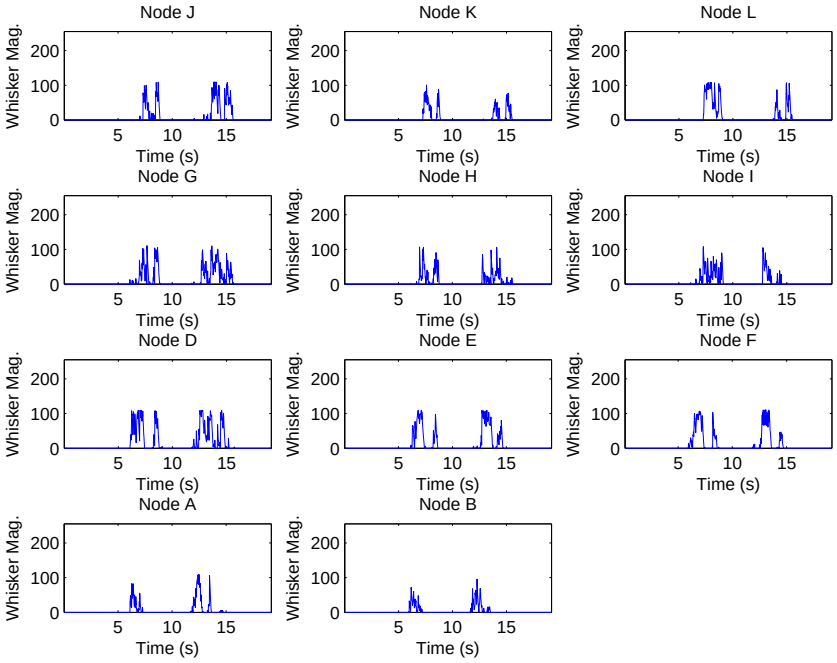

Figure B-5: Responses of whisker sensors to someone's blowing across skin.

Appendix C

PCB Layout

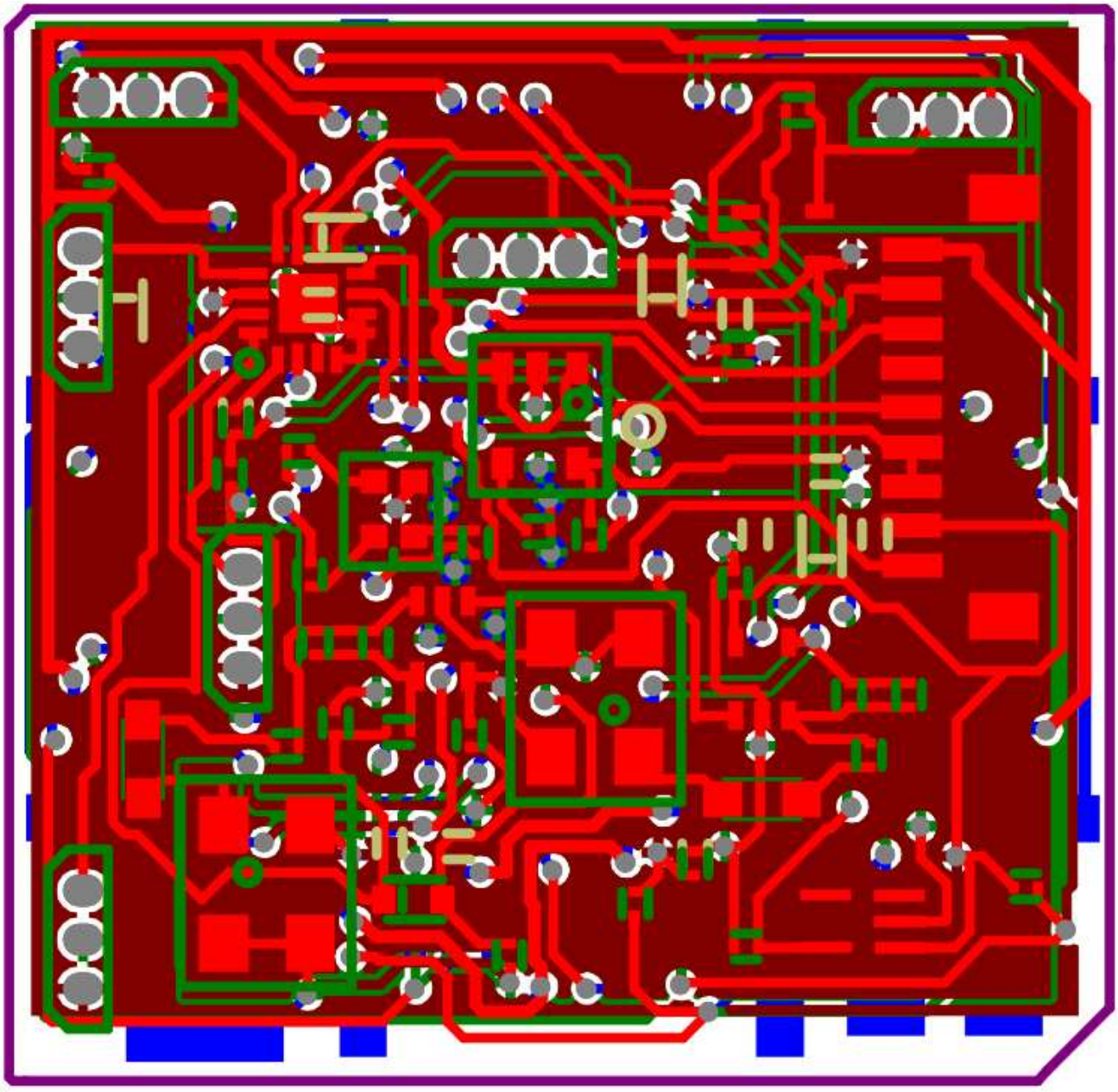

Figure C-1: PCB layout of entire node board.

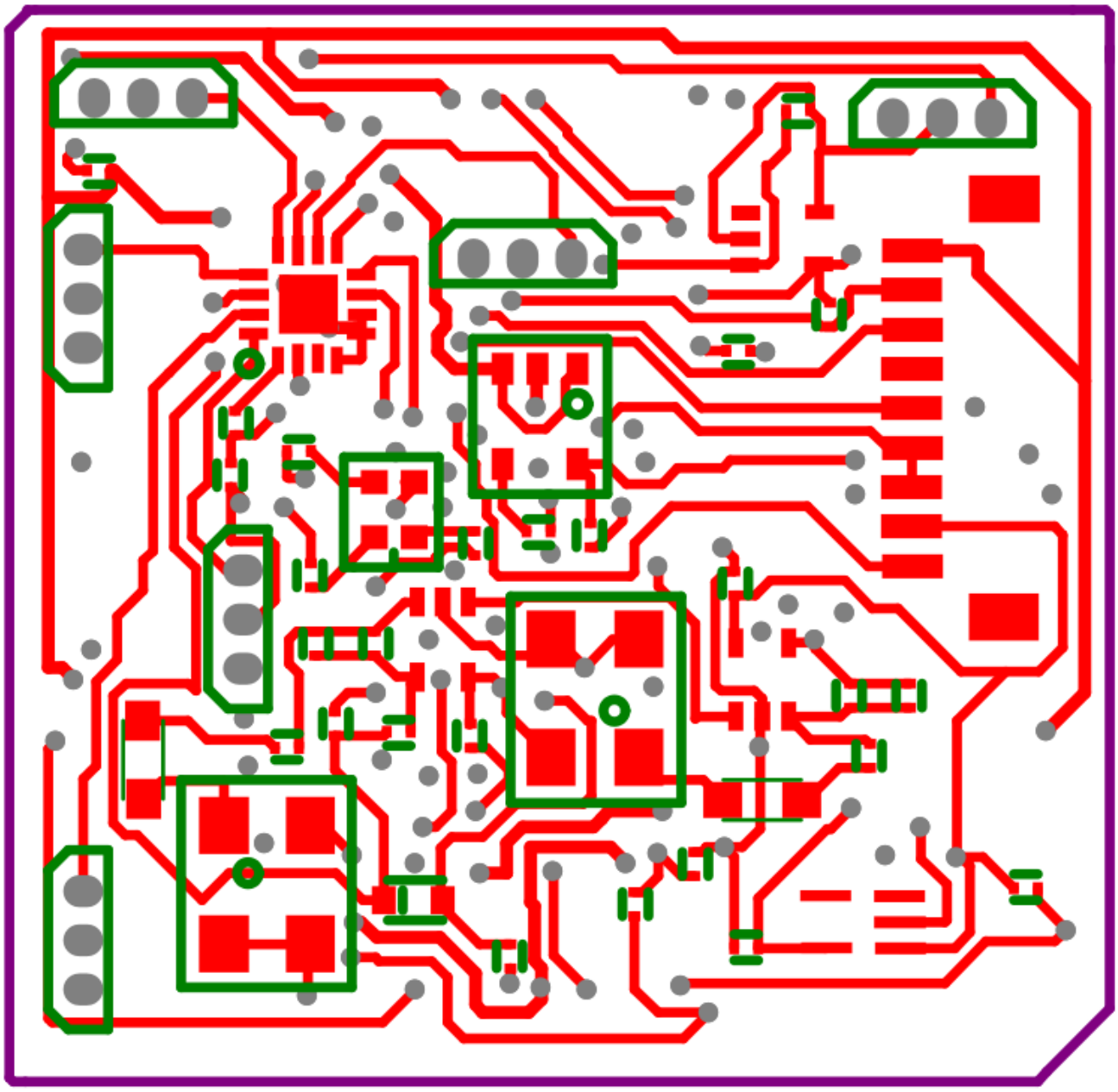

Figure C-2: PCB layout of top layer of node board.

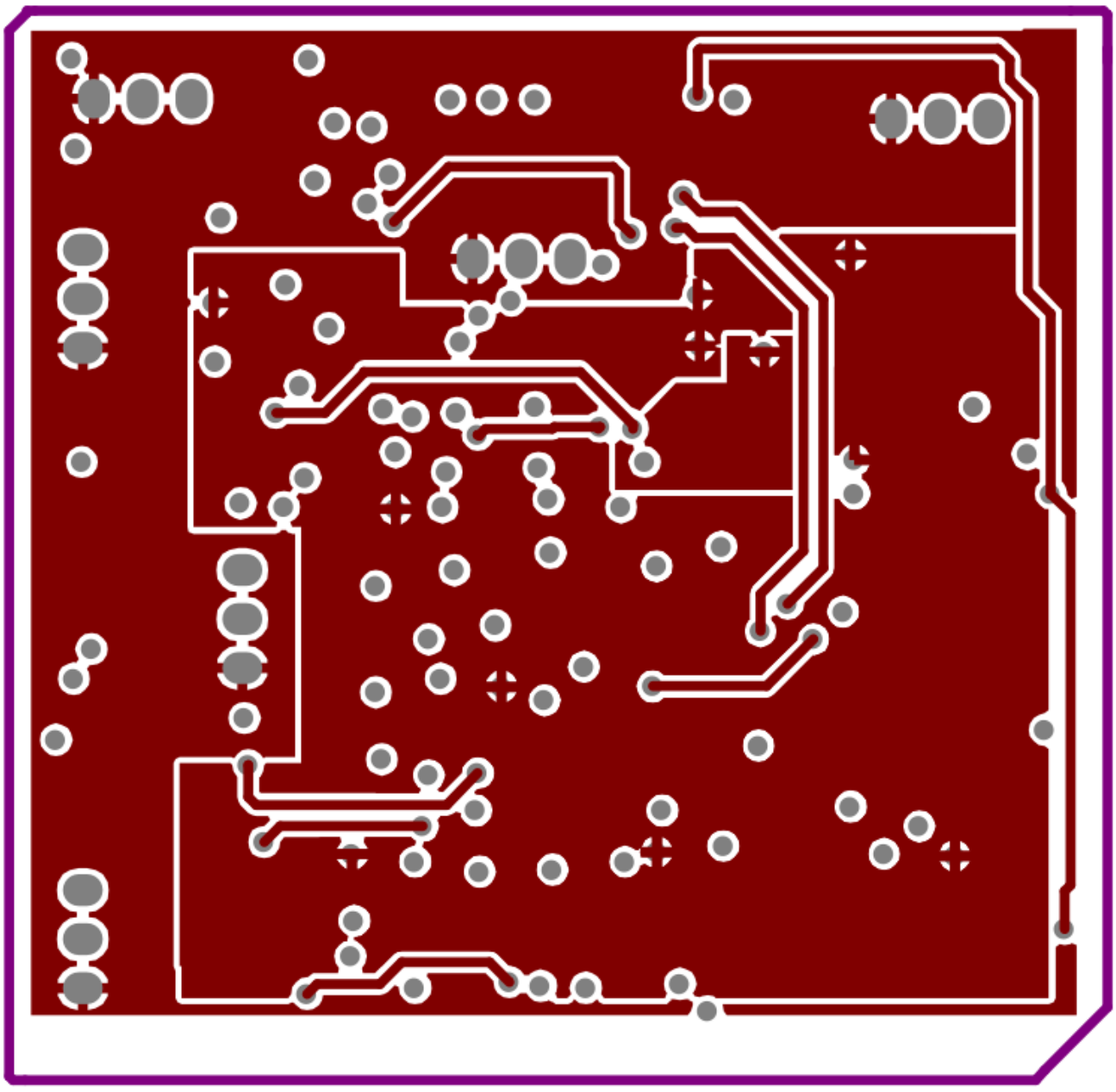

Figure C-3: PCB layout of interior layer of node board: VCC plane.

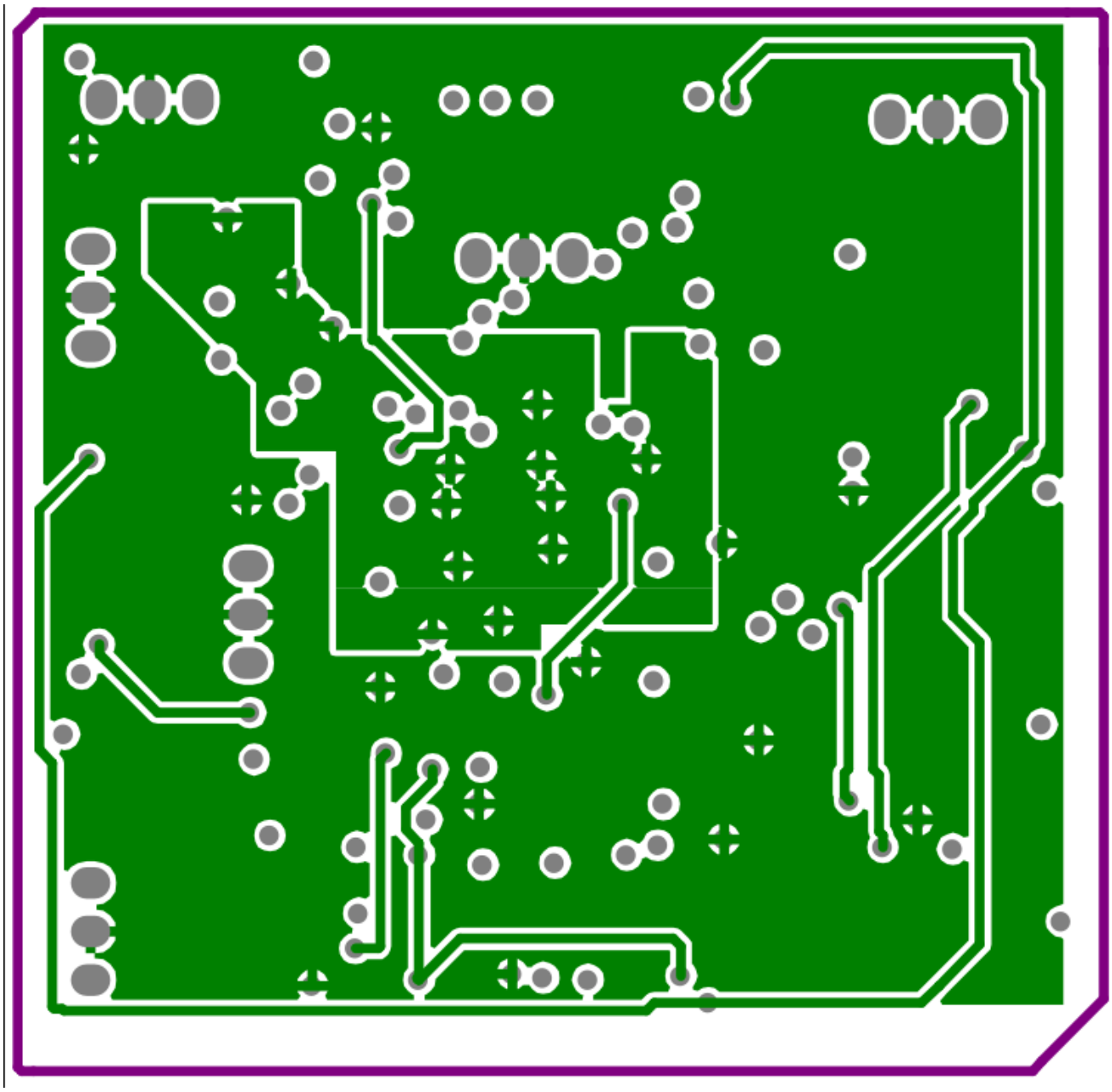

Figure C-4: PCB layout of node board: GND plane.

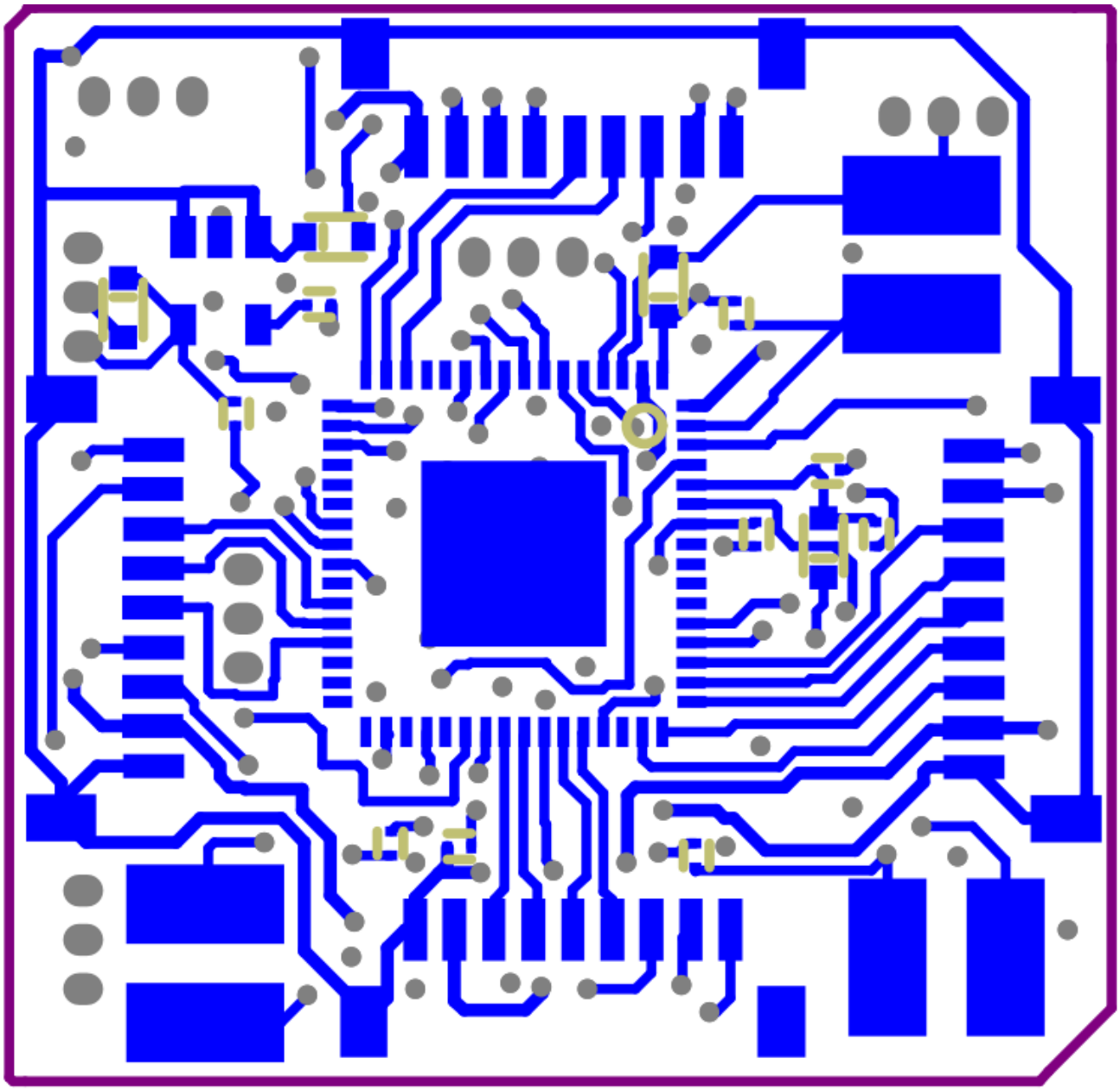

Figure C-5: PCB layout of bottom layer of node board.

Appendix D

Circuit Schematics

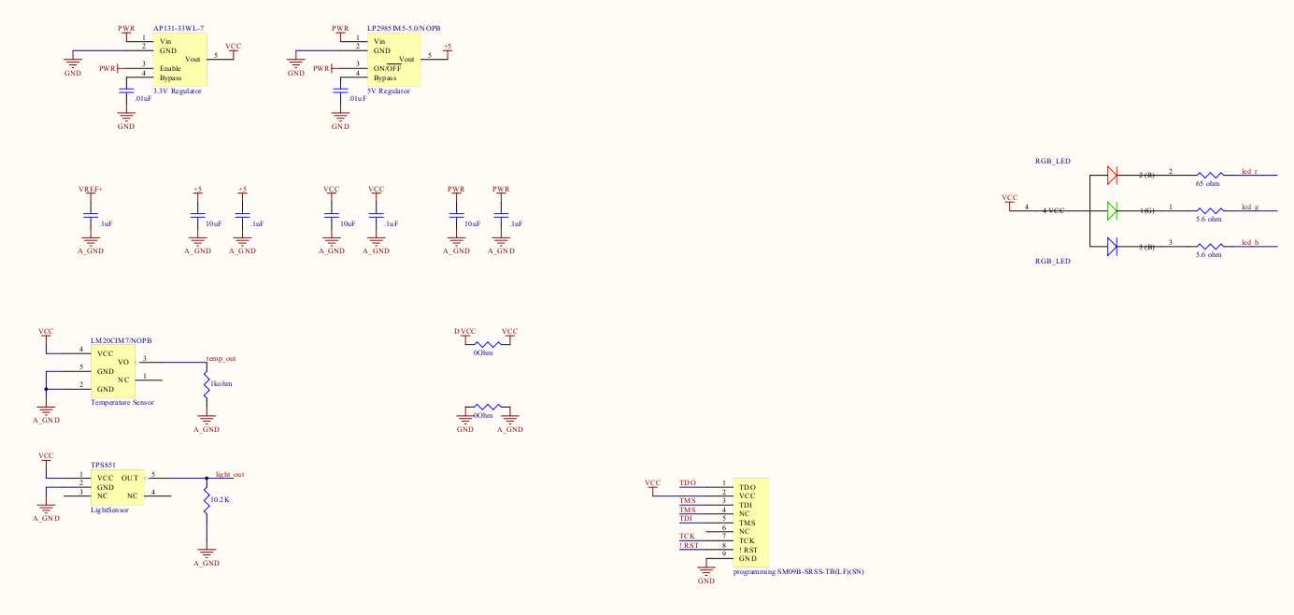

Figure D-1: Power, LED, temperature sensor, and light sensor circuitry for node. Note that the 1 K resistor for our LM20CIM temperature sensor is not populated.

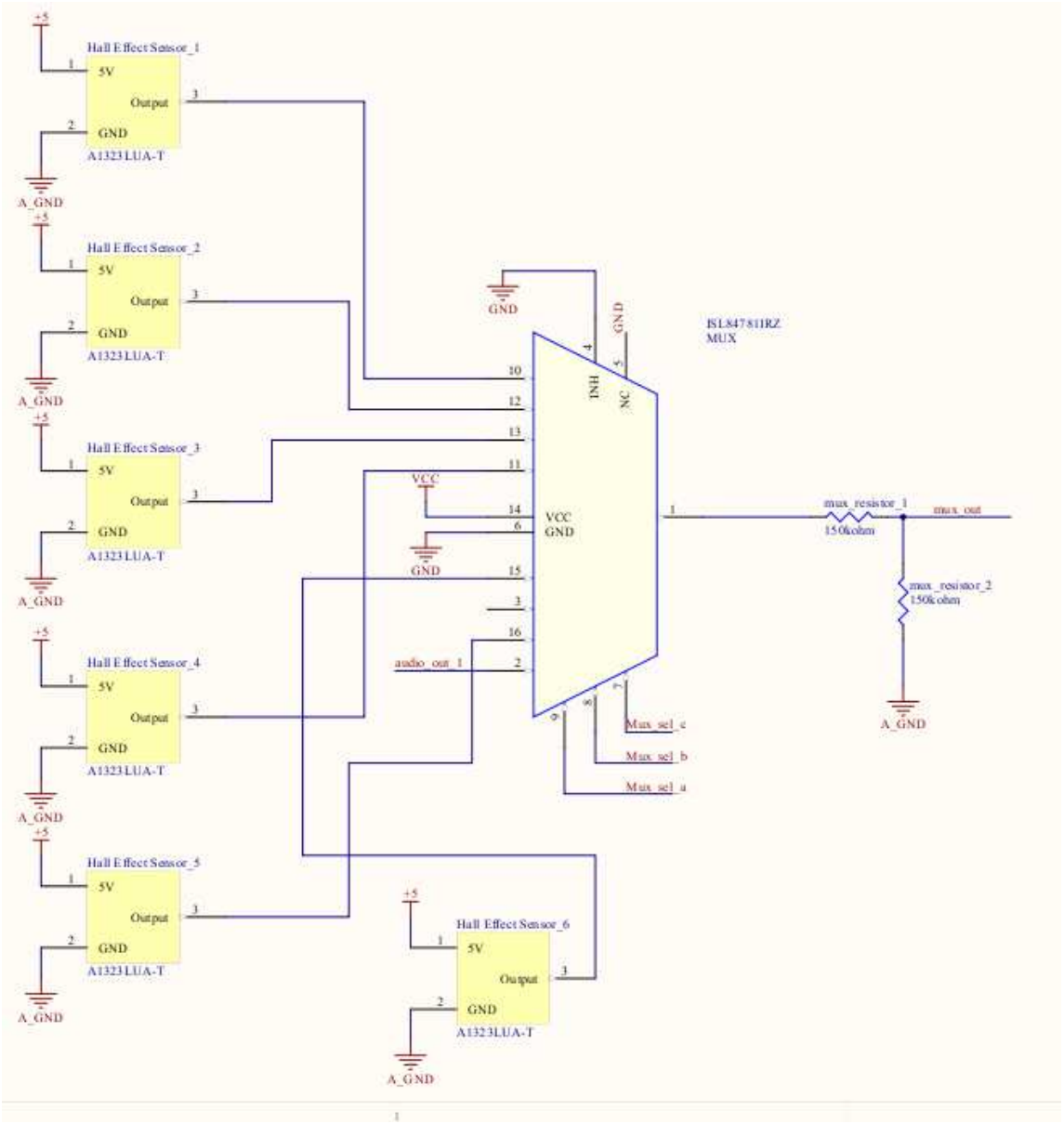

Figure D-2: Mux control circuitry for node.

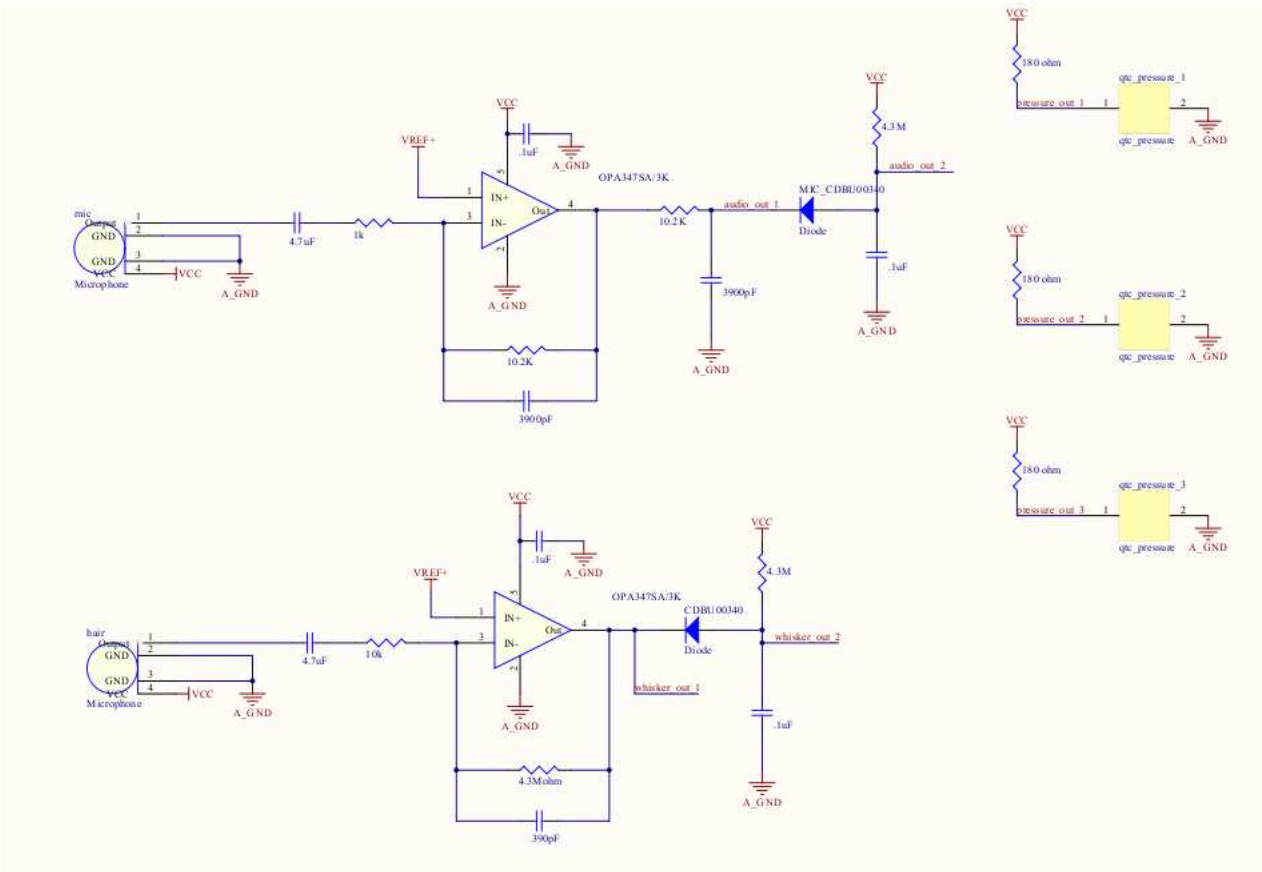

Figure D-3: Microphone and whisker conditioning circuitry.

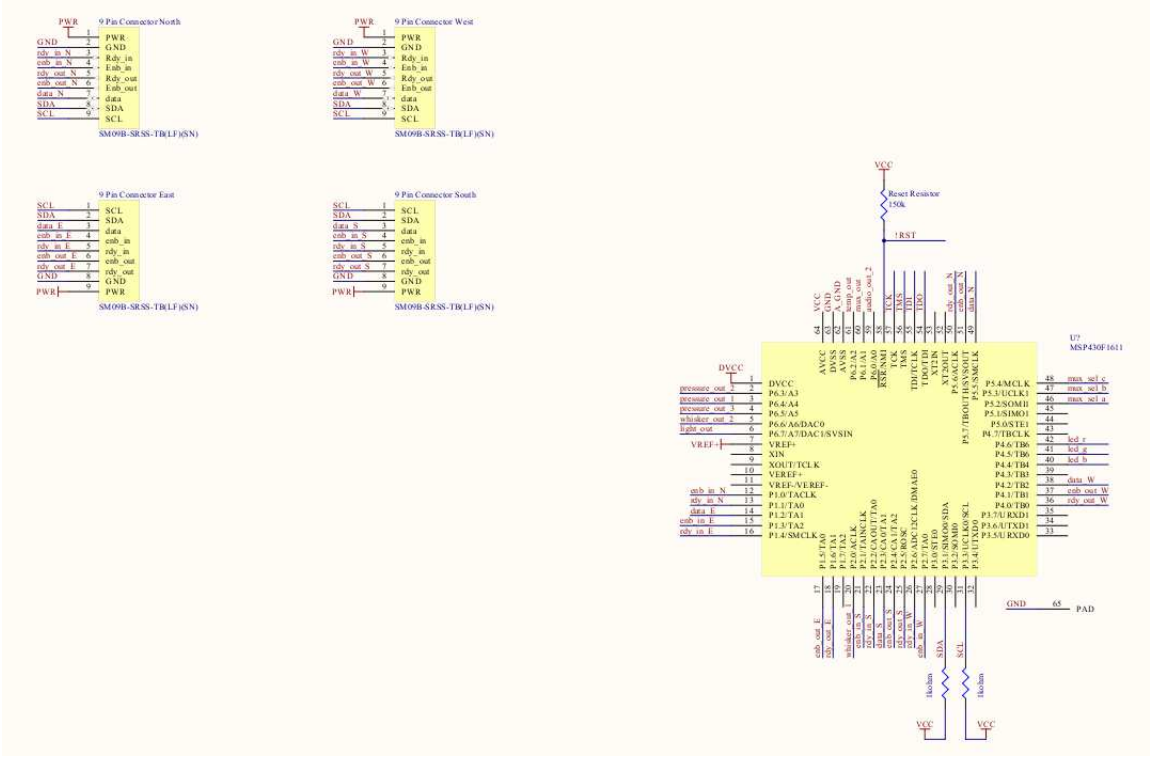

Figure D-4: Microcontroller and connector circuitry. Note that the 1 K resistors on the SCL and SDA lines are only populated on the master nodes.

Bibliography

[1] Knowles Acoustic. Mini sisonictm microphone specification. Online.

[2] I. F. Akyildiz, Weilan Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 2002.

[3] G. Arslan, J. R. Marden, and J. S. Shamma. Autonomous vehicle-target assignment: A game theoretical formulation. *ASME Journal of Dynamic Systems, Measurements, and Contorl*, 2007.

[4] D. Arvind, K. Elgaid, T. Krauss, A. Paterson, R. Stewart, and I. Thayne. Towards an integrated design approach to specknets. In *IEEE International Conference on Communications*, 2007.

[5] D. Arvind and K. Wong. Speckled computing: Disruptive technology for networked information appliances. In *IEEE International Symposium on Consumer Electronics*, 2004.

[6] Stacy J. Morris Bamberg, Ari Y. Benbasat, Donna Moxley Scarborough, David E. Krebs, and Joseph A. Paradiso. Gait analysis using a shoe-integrated wireless sensor system. *IEEE Transactions on Information Technology in Biomedicine*, 2008.

[7] Michael Broxton, Joshua Lifton, and Joseph A. Paradiso. Localizing a sensor network via collaborative processing of global stimuli. In *Second European Workshop on Wireless Sensor Networks*, 2005.

[8] Michael Broxton, Joshua Lifton, and Joseph A. Paradiso. Wireless sensor node localization using spectral graph drawing and mesh relaxation. *ACM Mobile Computing and Communications Review*, 2006.

[9] Leah Buechley and Michael Eisenberg. Fabric pcbs, electronic sequins, and socket buttons: Techniques for e-textile craft. *Journal of Personal and Ubiquitous Computing*, 2007.

[10] Leah Buechley, N. Elumeze, and M. Eisenberg. Electronic/computational textiles and children's crafts. In *Proceedings of Interaction Design and Children*, 2006.

[11] Colin Burnett. I2c. Published on Wikipedia, December 2007.

[12] Interlink Electronics. Fsr force sensing resistor integration guide and evaluation parts catalog. Online.

[13] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. *Mobile Computing and Networking*, 1999.

[14] Inc. EXOS. Dexterous hand master user's manual. Published on Wikipedia, 1989.

[15] Drew Fudenberg and Jean Tirole. *Game Theory*. MIT Press, 1991.

[16] Mitsuhiro Hakozaki, Atsushi Hatori, and Hiroyuki Shinoda. A sensitive skin using wireless tactile sensing elements. In *Technical Digest of the 18th Sensor Symposium*, 2001.

[17] Tian He, Sudha Krishnamurthy, John A. Stankovic, Tarek Abdelzaher, Liqian Luo, Radu Stoleru, Ting Yan, Lin Gu, Jonathan Hui, and Bruce Krogh. Energy-efficient surveillance system using wireless sensor networks. In *Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services*, 2004.

[18] W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *IEEE Transactions on Wireless Communications*, 2002.

[19] Texas Instruments. Msp430x15x, msp430x16x, msp430x161x mixed signal microcontroller. Published as microcontroller datasheet on Texas Instruments' website., August 2006.

[20] Intersil. Ultra low on-resistance, low-voltage, single supply, 8 to 1 analog multiplexer. Online.

[21] Matthew Josephson. *Edison: A Biography*. John Wiley & Sons, Inc., 1992.

[22] Daniel Sang Kim. Sensor network localization based on natural phenomena. Master's thesis, The Massachusetts Institute of Technology, 2006.

[23] J. Lifton, M. Mittal, M. Lapinski, and J. A. Paradiso. Tricorder: A mobile sensor network browser. In *Proceedings of the ACM CHI 2007 Conference, Mobile Spatial Interaction Workshop*, 2007.

[24] T. Linz, C. Kallmayer, R. Aschenbrenner, and H. Reichl. Fully integrated ekg shirt based on embroidered electrical interconnections with conductive yarn and miniaturized flexible electronics. 2006.

[25] B. Lo, S. Thiemjarus, R. King, and G. Yang. Body sensor network - a wireless sensor platform for pervasive healthcare monitoring. In *The 3rd International Conference on Pervasive Computing*, 2005.

[26] Peratech Ltd. Peratech evaluation kit integration guide. Online.

[27] Vladimir J. Lumelsky, M. S. Shur, and S. Wagner. Sensitive skin. *IEEE Sensors Journal*, 2004.

[28] Mateusz Malinowski, Matthew Moskwa, Mark Feldmeier, Mathew Laibowitz, and Joseph A. Paradiso. Cargonet: A low-cost micropower sensor node exploiting quasi-passive wakeup fo adaptive asynchronous monitoring of exceptional events. In *Proceedings of the 5th ACM Conference on Embedded Networked Sensor Systems*, 2007.

[29] C. Mattmann, O. Amft, H. Harms, G. Troster, and F. Clemens. Recognizing upper body postures using textile strain sensors. In *11th IEEE International Symposium on Wearable Computers*, 2007.

[30] Sergio Maximilian. A textile based capacitive pressure sensor. *Sensor Letters*, 2004.

[31] Joseph A. Paradiso, Joshua Lifton, and Michael Broxton. Sensate media: Multimodal electronic skins as dense sensor networks. *BT Technology Journal*, 2004.

[32] Joseph A. Paradiso, Joshua Lifton, and Michael Broxton. Sensate media: Multimodal electronic skins as dense sensor networks. *BT Technology Journal*, 2004.

[33] Ǵerardo Baroeta Perez. S.n.a.k.e.: A dynamically reconfigurable artificial sensate skin. Master's thesis, The Massachusetts Institute of Technology, 2006.

[34] Jun Rekimoto. Smartskin: An infrastructure for freehand manipulation on interactive surfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Changing Our World, Changing Ourselves*, 2002.

[35] J. Rossignac, M. Allen, W. J. Book, A. Glezer, I. Ebert-Uphoff, C. Shaw, D. Rosen, S. Askins, Jing Bai, P. Bosscher, J. Gargus, Byung Moon Kim, and I. Llamas. Finger sculpting with digital clay: 3d shape input and output through a computer-controlled real surface. *Shape Modeling International*, 2003.

[36] National Semiconductor. Lm20 2.4v, 10 ua, sc70 micro smd temperature sensor. Online.

[37] Philips Semiconductors. The i2c-bus specification version 2.1. Published as a standard online by Philips Semiconductor company., January 2000.

[38] Eric Smalley. Flexible sensors make robot skin. *Technology Research News*, 2004.

[39] Walter Dan Stiehl, Jeff Lieberman, Cynthia Brazeal, Louis Basel, Roshni Cooper, Heather Knight, Levi Lalla, Allan Maymin, and Scott Purchase. The huggable: A therapeutic robotic companion for relational, affective touch. In *IEEE CCNC 2006 Proceedings*, 2006.

[40] Walter Dan Stiehl, Jeff Lieberman, Cynthia Brazeal, Louis Basel, Levi Lalla, and Michael Wolf. The design of the huggable: A therapeutic robotic companion for relational, affective touch. In *Proceedings of AAAI Fall Symposium on Caring Machines: AI in Eldercare*, 2006.

[41] Toshiba. Tps851. Online.

[42] Unknown. Dipole. Published on Wikipedia, August 2008.

[43] G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M. Welsh. Deploying a wireless sensor network on an active volcano. *IEEE Internet Computing*, 2006.

[44] K. Wong, D. Arvind, N. Sharwood-Smit, and A. Smith. Specknet-based responsive environments. In *Proceedings of the Ninth International Symposium on Consumer Electronics*, 2005.