DEVELOPMENT OF A REMOTE IOT LABORATORY FOR CYBER PHYSICAL SYSTEMS

By

Amee Mayur Patel

DEVELOPMENT OF A REMOTE IOT LABORATORY FOR CYBER PHYSICAL SYSTEMS

By

Amee Mayur Patel

A Thesis Submitted to the Faculty of the University of
Tennessee at Chattanooga in Partial
Fulfillment of the Requirements of the Degree
of Master of Science: Engineering

The University of Tennessee at Chattanooga
Chattanooga, Tennessee

August 2017

ABSTRACT

A remote Internet of Things (IoT) laboratory has been developed for use in teaching and research of cyber physical systems. The laboratory is configured such that users interact via the web to control and collect data from connected devices. The concept and technology of the remote IoT lab is successfully demonstrated in two applications. First the laboratory is configured for UTChattSat, a ground model small-satellite system (CubeSat) designed to enhance K-12 learning in the space sciences. The system utilizes real-time communication and web-based user control to create a distributed multi-user interface. Second, the laboratory is configured for a distributed sensor network with a single-user interface. The interconnected, real time, and smart system with embedded sensors and processors is used to provide data for assessment of current building energy models. Finally, challenges posed by interconnected and reconfigurable systems and the implementable future works are discussed.

DEDICATION

This work is dedicated to my family.

ACKNOWLEDGEMENTS

I would first like to thank the supporter of this work in part, the Tennessee Higher Education Commission (THEC) through Center of Excellence in Applied Computational Science and Engineering (CEACSE) program. Next, I would like to thank my thesis advisor Dr. Daniel Loveless for his continual support, encouragement, and guidance throughout this thesis. Dr. Loveless has always been an amazing teacher, shaping and enriching my journey of graduate school with his tireless efforts and dedication. Thank you Dr. Loveless for the most special "out-of-this" world experiences!

I would also like to thank Dr. Donald Reising who has been the co-advisor in this work. Dr. Reising has always guided me through his inspirational words and experiences to not only be a sharp technical person but also be a good person in life. I would like to thank Dr. Raga Ahmed who has been constantly an enormous support throughout my graduate school.

I would like to thank Dr. Loveless, Dr. Reising and Dr. Raga Ahmed for reviewing this work. Their valuable inputs in this paper had much more improvements. Also, I would like to thank Dr. Charles Margraves to be a great support in this work. Also, I would like to thank Department of Electrical Engineering, University of Tennessee, Chattanooga (UTC), for providing all the support and wonderful professors. I would also thank my companions Ellis Richards, Daniel Johnson and Mathew Joplin for making this experience more enjoyable and memorable. Also, I would like to thank Pratiksha Patel for providing moral support and companionship at every moment of this journey. This journey at UTC was more beautiful because of you, Pratiksha.

My life and the journey at UTC as a graduate student have been always incomplete without my family. I would like to thank my parents, parents-in-law for being a tremendous source of inspiration and guidance. My father lives in me forever to support and inspire me. My brother, Het has supported me in a lot of the technical aspects that I was not able to visualize. My husband, Mayur is always at my side to encourage me and support me in all ways I ever imagined. Thank you Mayur for being so patient and sacrificing all the time which was yours.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I
INTRODUCTION

Overview

The remote collaboration concept is a project where, from any corner of the world it can be interfaced, controlled and utilized by researchers as seen in Figure 1.1. This thesis develops a remote Internet of Things (IoT) laboratory that uses the concept of Cyber Physical Systems (CPS) to connect physical projects and networks via Internet. The laboratory exhibits features such as remote access, sensor networking, real-time communication and data retrieval. This idea is demonstrated in two example implementations. The first implementation demonstrates the use for science education by allowing for remote connection to space technology in classrooms. The second implementation demonstrates a distributed sensor network project, which involves development of real-time sensor networks used to assess building energy dissipation. The next sections describe the problem statement for each implementation.

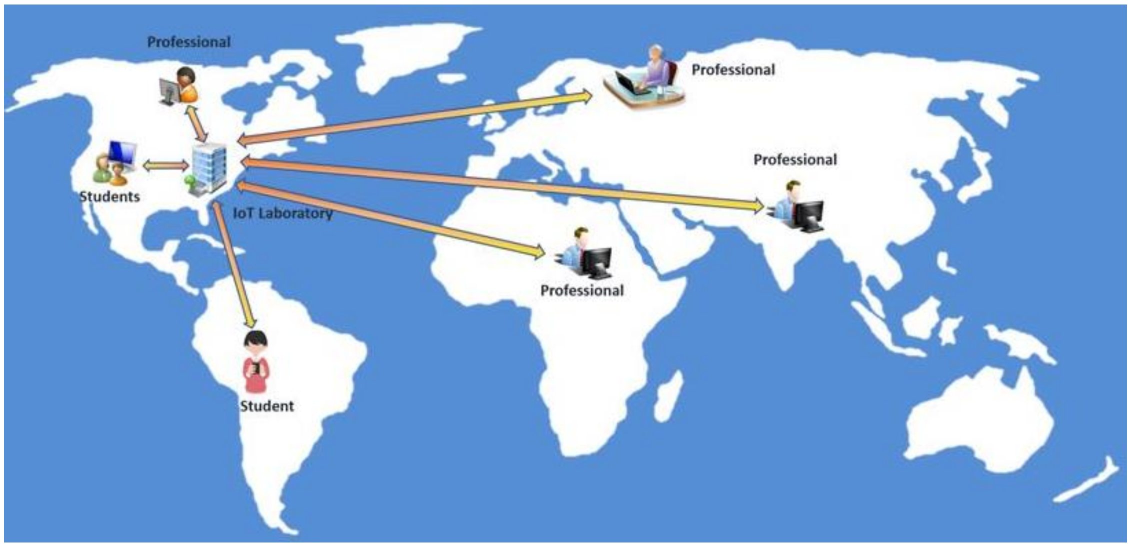


Figure 1.1 Remote Collaboration: Concept visualization

Science Education

The shifting emphases of debate in science education over the past 40-50 years is clearly reflected in numerous slogans and rallying calls including ‘Being Scientist for a Day’, ‘Learning by Doing’, ‘Science for all’, Children Making Sense of World’ and ‘Science as a Way of Knowing’[1]. Exposure to science technology is critical for student engagement. Models are important in scientific research both in formulating hypotheses to be tested and in describing scientific phenomena. Model-based teaching is any implementation that brings together information resources, learning activities, and instructional strategies intended to facilitate mental model-building both in individuals and groups of learners [2]. This thesis presents a space science based model, which helps learners understand the basics of space science and trains them to use, modify and present the ideas.

CubeSats (or nanosatellites) are small satellites with a standard unit size of 10 x 10 x 10 cm^3 and weight of less than 1 kg per unit. They are low-cost and highly compact by design, size and weight when compared to conventional satellites. They can be used with a high-altitude balloon or placed in a low-earth orbit in space. CubeSat model-based learning is a part of technical future from which the students can learn about space sciences. Getting familiar with space science impacts the students' thought process and their determination to advance in the fields of science and engineering.

UTChattSat is an educational kit that provides interactive learning of space sciences to students in their crucial stage of career advancement. UTChattSat is an Internet of Things (IoT) device that provides real-time communication and user interfacing using low cost components and off-shelf electronics. The model feature meets CubeSat standards, which enable multiple uses. The basic building blocks, network devices, topologies and protocols used for this implementation are explained in Chapter 2. Further, Chapter 3 describes the block diagram, hardware and software.

Distributed Sensor Networks

The phrase "smart city" has a broad meaning and can refer to any of the various assets such as people, activities, organizations, facilities, services and resources that comprise a city [3]. It refers to an urban development initiative that leverages secure CPS as a means of improving life of the people who live, work and play within the city [4]. Achieving the goal of the Better Building Challenge will require use of secure CPS for the specific purpose of developing "Smart Buildings" that have the capability of monitoring and reporting energy and resource usage in real-time.

The development of an interconnected real-time smart system with embedded sensors and processors presented here will be used to assess current building energy models in the remote laboratory. The data produced and collected will be used to improve existing building energy models to facilitate real-time, high fidelity energy usage and effective analysis.

The basic building blocks, network devices, topologies and protocols used for this implementation are explained in Chapter 2. Chapter 4 explains the block diagram, hardware development and the software algorithms and the results for the distributed sensor networks.

CHAPTER II

REMOTE IOT LABORATORY

Introduction

This chapter provides the general structure of the remote IoT laboratory. A short introduction of IoT and CPS provides the background for the concept. Then the system structure, main system components used in the system, networking devices used and inter-controller communication patterns are discussed.

Internet of Things and Cyber Physical Systems

IoT marks a new era of computing technology where physical systems and projects embedded with electronics, sensors, actuators and software are connected to the Internet. The Cisco Internet Business Solutions Group (IBSG) says IoT is at point in time when more objects will be connected to the Internet than people. Cisco IBSG estimates IoT was “born” sometime between 2008 and 2009 as shown in Figure 2.1 and predicted that there will be 50 billion devices connected to the Internet by 2020. The world cares about IoT because it has and will transform the way people, companies and institutions are going to interact with the objects surrounding them in their personal, social and economic lives [5].

Smart objects are autonomous physical/digital objects augmented with sensing, processing, and network capabilities. In contrast to RFID tags, smart objects carry chunks of application logic that let them make sense of their local situation and interact with human users. They sense, log, and interpret what's occurring within themselves and the world, act on their own, intercommunicate with each other, and exchange information with people, like the sensor networks developed here perform.

CPS refer to a new generation of systems with integrated computational and physical capabilities that can interact with humans through new modalities. The ability to interact and expand the capabilities of the physical world by efficient use of computation, communication, and control is a key for future technology developments [6]. CPS are comprised of interconnected real-time "smart networked systems with embedded sensors, processors and actuators that are designed to sense and interact with the physical world, cyber world and support real-time, guaranteed performance in safety-critical applications" [7]. CPS provide a means by which to monitor resource consumption to enable improved asset management and facilitate real-time response to challenges, which leads to the efficient allocation and usage of these assets and a subsequent reduction in operating costs and resource consumption. The buildings, that comprise a city, are a significant asset in which CPS can be leveraged to reduce operating costs and resource consumption [8, 9].

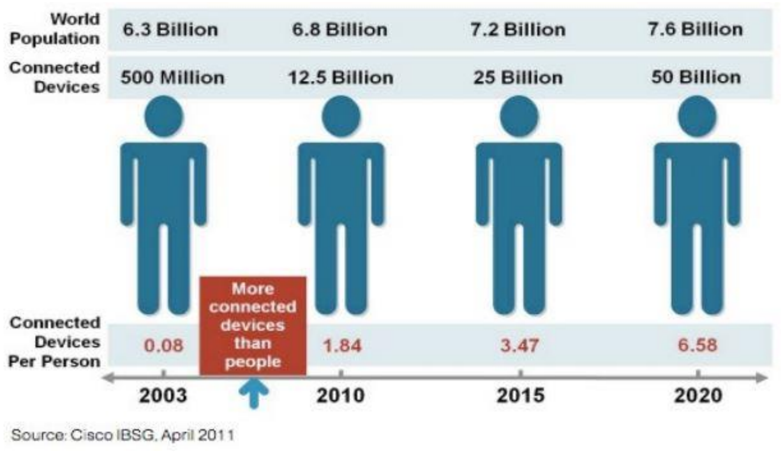


Figure 2.1 Internet of Things Was “Born” Between 2008 & 2009 [5]

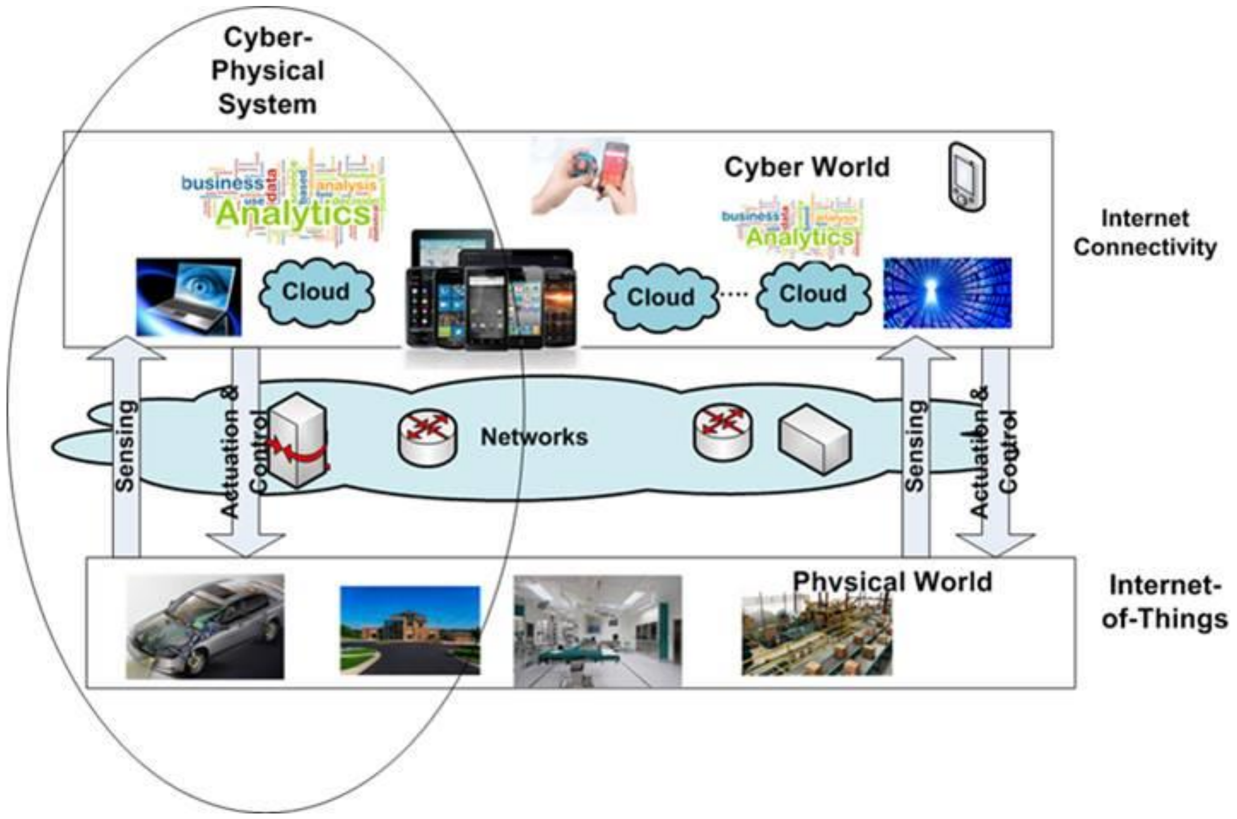


Figure 2.2 IoT and CPS: where each lie [10]

System Structure

The remote IoT laboratory developed here is structured in two types of interfaces based on accessibility and measurability, i.e., Single User Multi Node Interface and Multi User Single Node Interface. These interface structures are used in different applications that explained in Chapter 3 and 4. The next two sections explain these two interfaces:

Single User Multi Node Interface

The Single User Multi Node Interface allows a single authorized user to access a system. It consists of a distributed sensor network that consists of multipoint measurements via several sensors connected to a hub in a star topology. All sensor nodes are affixed at multiple points in the laboratory. The nodes are wireless. A basic structure is shown in Figure 2.3.

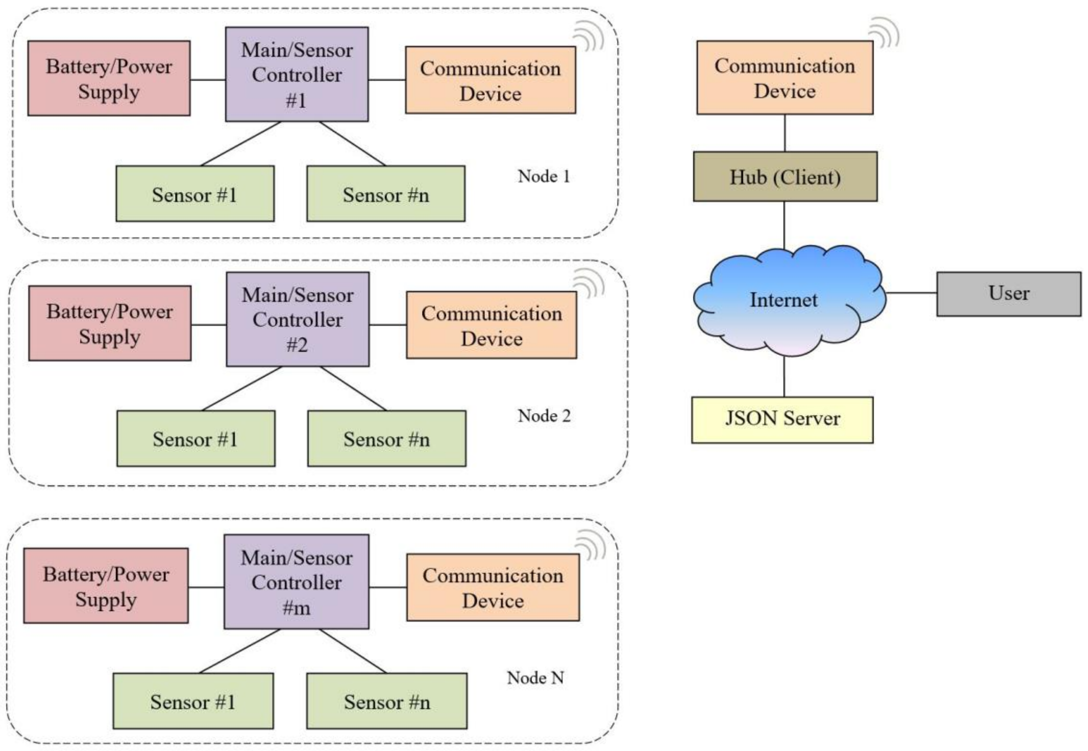


Figure 2.3 Distributed Sensor Single User Multi Node Interface structure diagram

Multi User Single Node Interface

The Multi User Single Node Interface is a system where multiple remote users can access the system. The device hardware is unified and consists of single point measurements. The data transmission is encrypted and decrypted by only authorized users. Point-to-point communication is used for command and data transmissions. This system is wireless using a battery power. A basic structure is shown in Figure 2.4 and the modules mentioned above are explained in the System Components section.

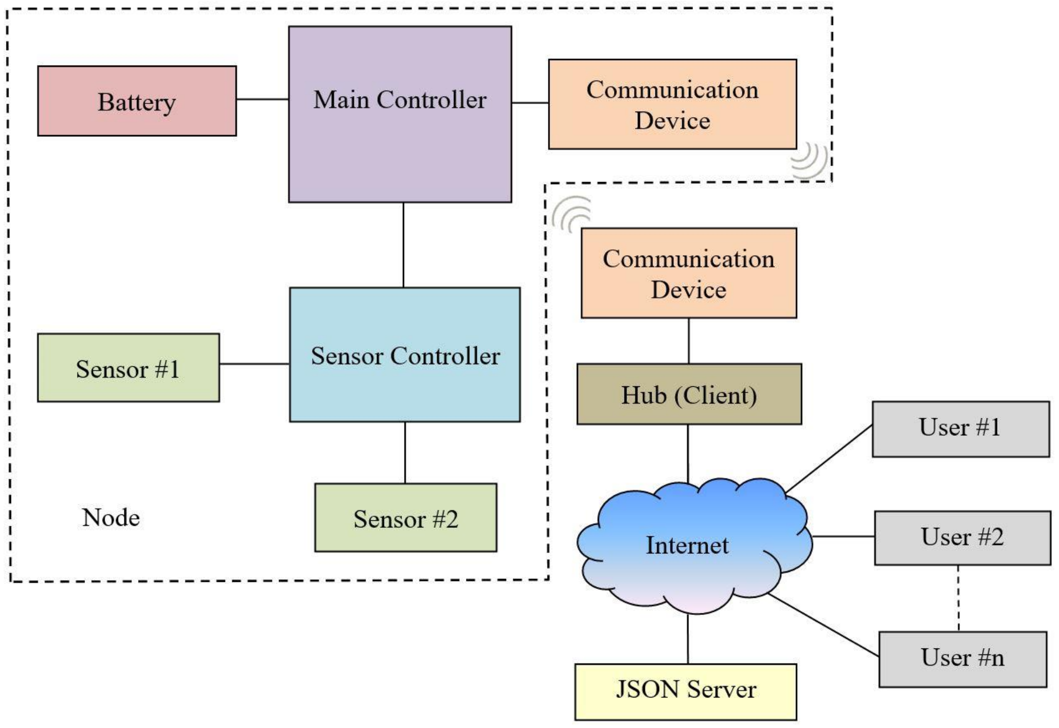


Figure 2.4 Multi User Single Node Interface structure diagram

System Components

The system structures explained in the previous section consist of three controlling components explained here: Main Controller, Sensor Controller and Hub.

Main Controller

The Main Controller (MC) interfaces with all of the peripherals attached to a given node. The MC is comprised of a ATmega328. The prototype was developed using Arduino Duemilanove/UNO boards. The MC receives commands from the user via the controlling device and if separate, sends the commands to the Sensor Controller (SC).

The MC and SC are kept separate such that the MC handles the tasks of command and data pipelining and data storage, and handling other SCs. The SC handles the complexity of interfacing with the sensors. Several SCs can be connected to a single MC. The MC receives data from the SC and transmits to the output device via the communication device. It interfaces with the SC via a defined protocol for command and data as described in the Protocols section. The peripherals include sensor controllers, power, communication device, storage devices, etc. as required by the implementation. The MC is powered by a battery to facilitate mobile applications.

Sensor Controller

The sensor controller (SC) interfaces with different sensors and fetches data to and from MC when required. It works as an interfacing unit between the MC and sensors. The SC consists of a ATmega328 chip. The prototype is built using an Arduino Duemilanove board. The main task performed by the SC is to interface with sensors by serial communication.

The data from these sensors are collected as configured by the sensor configuration field in the command word received from the MC. It transmits the data collected from the sensors in the format specified by the data word to the MC. Additional sensors can be added to the sensor controller via minor changes in the software. The SC is powered through the MC.

Hub

The Hub is a computer for collecting data from the Nodes via a communication device. The Hub software is explained in Chapter 4. Briefly, the Hub parses, decrypts and stores the data and controls the data rate. The Hub acts as a client when client JSON (JavaScript Object Notation) files are loaded providing the capability to stream real-time data to the Internet. The Raspberry Pi 3 Model B is used as the Hub and remains connected to the Internet.

Networking Devices

The networking devices used for laboratory communication are Digi XBee S2. The XBee S2 Modules are engineered to meet ZigBee Mesh standards and support the unique needs of low-cost, low-power wireless sensor networks. ZigBee is an open global standard built on the IEEE 802.15.4 MAC/PHY standard. ZigBee defines a network layer above the IEEE 802.15.4 layers to support advanced mesh routing capabilities. IEEE 802.15.4 specifies the physical and media access control layers. It is ideal for applications requiring low latency and predictable communication timing [11]. The modules require minimal power (40mA at 3.3V) and provide reliable delivery of data between devices. The modules operate within the ISM 2.4 GHz frequency band and are pin-for-pin compatible with each other [12].

Inter- Controller Communications

This section explains the inter controller communication patterns used for the commands from the user and data to user for both types of interfaces.

Command Word (From User)

The command word has three fields, Experiment time (16 bit), Sample time (16 bit) and Sensor configuration (16 bit) as shown in Figure 2.5. The Experiment Time and Sample Time are entered by the user via the web interface, and the Sensor Configuration is entered by the user in integers from 0 to 31, 0 being all sensors OFF and 31 being all sensors ON. Only 8 bits from the Sensor Configuration word are used for defining the sensor configuration out of which 3 bits are reserved for spare sensors and 5 bits are used for existing sensors as shown in Figure 2.5.

Configuration bits set to 1 correspond to data requests from the noted sensor. For example, a request from the Gyroscope (b_{gy}), Altitude (b_{alt}) and GPS (Global Positioning System) (b_{GPS}) sensors requires a configuration word of 10101 (integer 21). This command turns Pressure (b_p) and Temperature (b_{temp}) sensors OFF.

Data Word: Single User Multi Node Interface (To User)

The data word protocol for the single user multi node interface is comprised of five data fields as shown in Figure 2.6. The sensor ID is a discrete alphabet ID which corresponds to each of the sensors connected through SC. This ID is used in the software algorithms to identify the data source. The values D1 to D4 are the corresponding sensing values from the sensors in float data type. As this data is retrieved by only the authorized user, no encryption is provided.

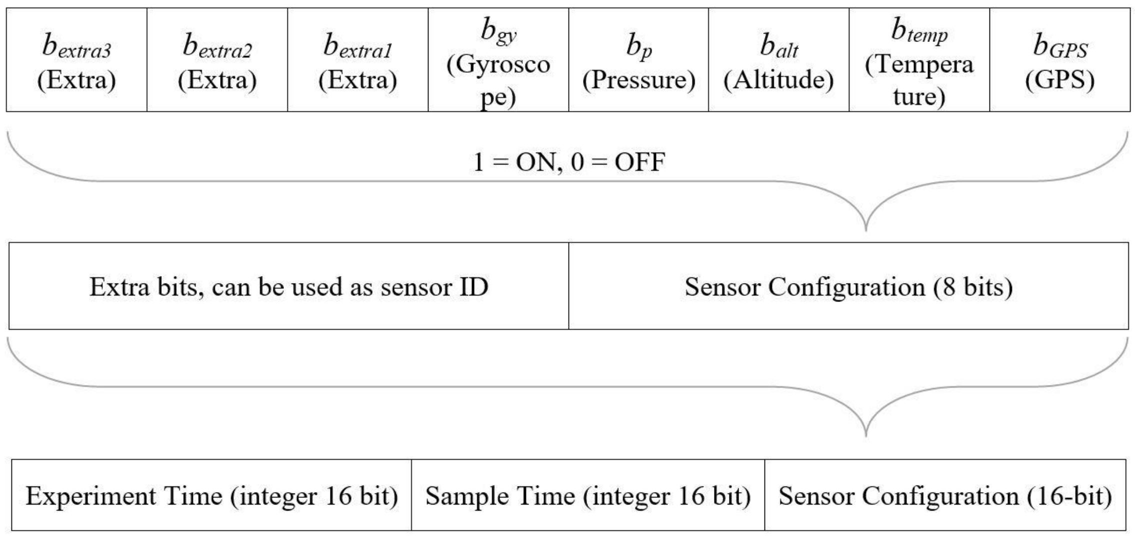


Figure 2.5 Command Word format for communication between Node and Hub. Each data field is separated by commas and ended with a new line character '\n' in transmission

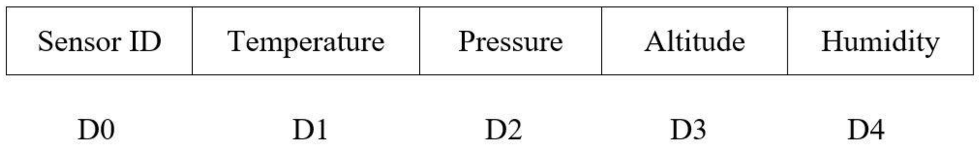


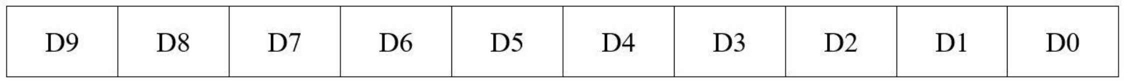
Figure 2.6 Data word format for communication between Node and Hub. Each data field is separated by commas and ended with a new line character '\n' in transmission

Data Word: Multi User Single Node Interface (To User)

The data word for the multi user HMI has 10 fields, each field describes data from each sensor as described in Table 2.1. The data transmitted is encoded in integer step numbers and is decoded at the user end (client side). The step numbers are the quantization level numbers calculated for each data value in software algorithms using the Equation (2.1).

$$Step\ number = \frac{(Max - Min)}{Resolution} \tag{2.1}$$

where, numerator (Max – Min) = Measurement range of the sensor, denominator (Resolution) = Resolution of the sensor.



D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

Figure 2.7 Multi User Single Node Interface: Data Word format. Each field is separated by comma and ended with a new line character '\n' in transmission

Table 2.1 Multi User Single Node Interface: Communication Protocol: Data Word Field Description

Data Field	Data Type	Sensor
D0	Roll	Gyroscope
D1	Pitch	Gyroscope
D2	Heading	Gyroscope
D3	Altitude	Altitude
D4	Pressure	Pressure
D5	Temperature	Temperature
D6	Fix Time	GPS
D7	Latitude	GPS
D8	Longitude	GPS
D9	Altitude	GPS

Conclusion

The chapter gives a background, structure and detailed description of the remote IoT laboratory. These components can be used in many different applications. Chapter 3 and Chapter 4 provide two examples of how the IoT laboratory can be applied/ implemented/ or configured.

CHAPTER III

EXAMPLE IMPLEMENTATION 1: UTCHATTSAT

Introduction

An example implementation of a distributed multi user single node interface is UTChattSat, a ground model small-satellite (CubeSat) for use in classrooms to transform the way students learn space technology. It provides usability, functionality and research from a middle school up to space-expert levels. The hardware and sensor development are explained in the following sections. The GUI operation and live data streaming are explained in the results section of this chapter. The 3D printed housing model of UTChattSat is shown in Figure 3.1 and its 3D explosion is shown in Figure 3.2.

Specifications:

- Usability for middle school level and functionality for experts.
- Manipulability in programs
- Self-starting program for unexpected resets
- Real-time control of Sensor Controller
- Transmission time at 9600 baud rate: 25.5 milliseconds.
- User defined:
    - Experiment Time
    - Sample Time

- Sensor Selection
- Board Selection

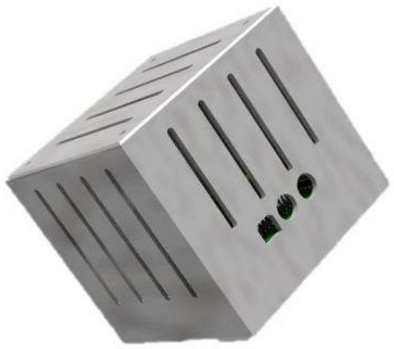


Figure 3.1 UTChattSat: 3D printed model

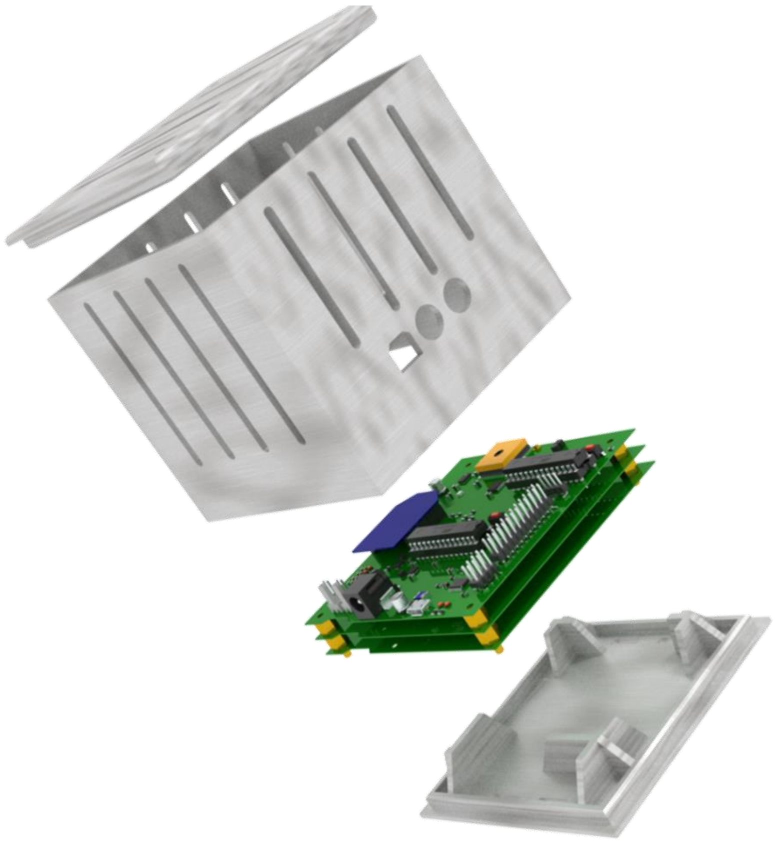


Figure 3.2 UTChattSat: 3D exploded view

Block Diagram

The hardware of UTChattSat is explained in the block diagram in Figure 3.3. It comprises of the Main Controller (MC), the Sensor Controller (SC), Adafruit GPS and 10-DOF (Degrees of Freedom) sensors, XBee and 4.8 VDC rechargeable battery. An example implementation for IoT communication and local computer display are presented in the Results section of this chapter. The connections of UTChattSat are shown in Table 3.1.

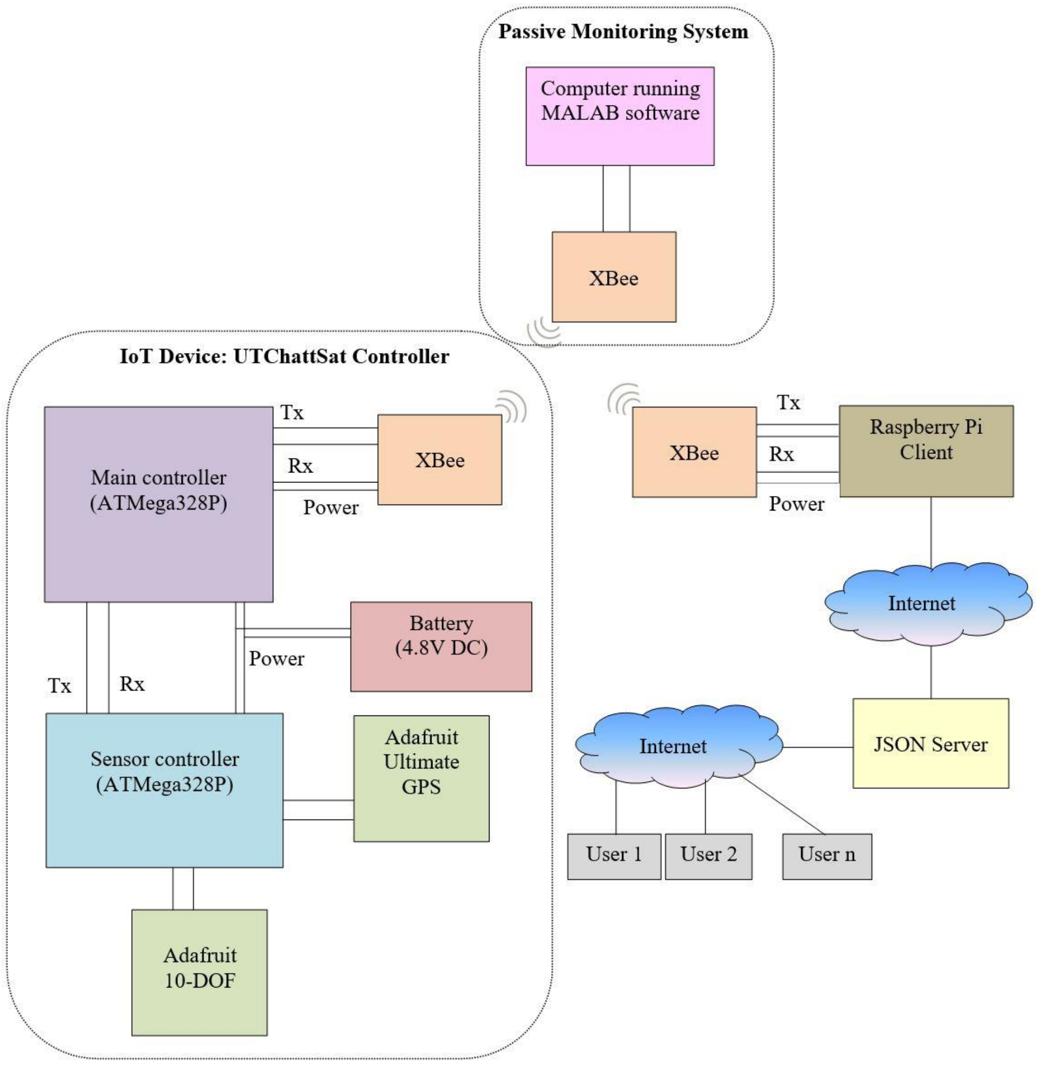


Figure 3.3 UTChattSat: Block Diagram, Interfacing and implemented example of IoT communication and passive monitoring display

Table 3.1 UTChattSat: Connection network for MC, SC, GPS sensor, 10-DOF sensor, XBee, Battery

Arduino Duemilanove (MC)	**Arduino Duelimanove (SC)**	**Adafruit GPS Breakout**	**Adafruit 10-DOF Breakout**	**XBee**	**Battery (4.8 VDC)**
Vin	Vin	Vin	-	Vin	Positive
GND	GND	GND	GND	GND	Negative
Digital 2	Tx (Dig1)	-	-	-	-
Digital 3	Rx (Dig0)	-	-	-	-
-	Digital 2	Rx	-	-	-
-	Digital 3	Tx	-	-	-
-	Analog 4	-	SDA	-	-
-	Analog 5	-	SCL	-	-
-	Digital 4	-	-	-	-
-	3.3 V	-	Vin	-	-
-	-	3.3 Vo	-	-	-
Tx (Dig 1)	-	-	-	DOUT	-
Rx (Dig 0)	-	-	-	DIN	-

The sensor boards selected in this implementation are described in the following sub-sections. These breakout boards were selected in this implementation as they carry the sensors, interfacing controllers and pinouts. Moreover, these boards have libraries that can be used in the software algorithms with the selected microcontroller (ATMega328P) for easy interface.

Adafruit Ultimate GPS

The breakout is built around the MTK3339 chipset, a high-quality GPS module that can track up to 22 satellites on 66 channels, has an excellent high-sensitivity receiver and a built-in antenna. It can sample up to 10 location updates per second for high speed, high sensitivity logging or tracking and requires less than 20 mA current during navigation.

It also comprises of an ultra-low dropout 3.3V regulator, ENABLE pin, a footprint for optional CR1220 coin cell to keep the RTC running and allows warm starts and a tiny bright red LED. The LED blinks at about 1 Hz while it's searching for satellites and blinks once every 15 seconds when a fix is found to conserve power. The 3 MTK3339-based module has the external antenna functionality and the built-in data-logging capability. The Adafruit Ultimate GPS is shown in Figure 3.4. [13]

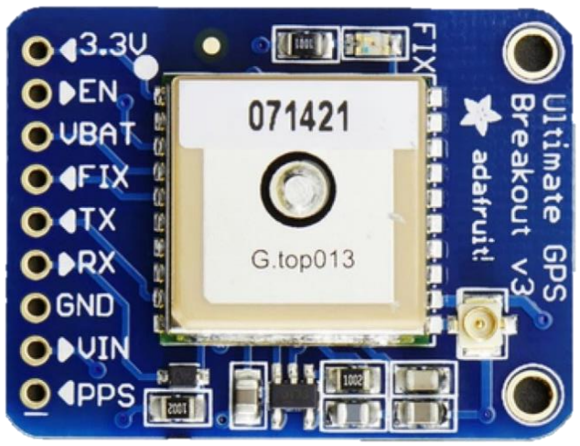


Figure 3.4 Adafruit Ultimate GPS Breakout Board [13]

Adafruit 10-DOF

Adafruit's 10DOF (10 Degrees of Freedom) breakout board allows to capture ten distinct types of motion or orientation related data. The sensor board is shown in Figure 3.5. It comprises of following sensors:[14]

- LSM303DLHC: a 3-axis accelerometer (up to +/-16g) and a 3-axis magnetometer (up to +/-8.1 gauss) on a single die
- L3GD20: a 3-axis gyroscope (up to +/-2000 degrees per second)

- BMP180: A barometric pressure sensor (300-1100 hPa) that can be used to calculate altitude with an onboard temperature sensor.

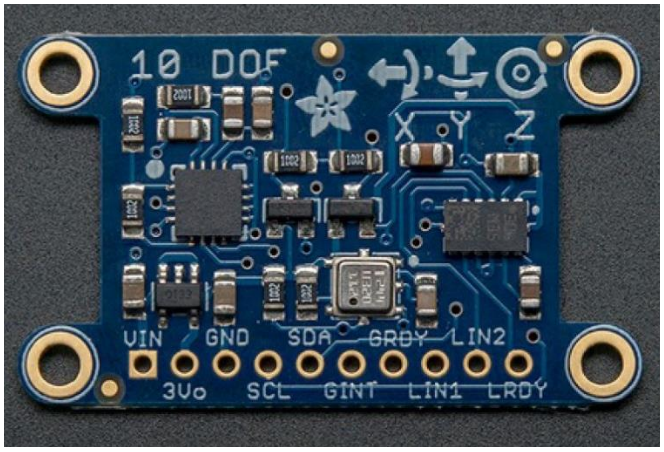


Figure 3.5 Adafruit 10-DOF Breakout Board [14]

Software Development

The UTChattSat software can be segmented into two main blocks: Main Controller (MC) and Sensor Controller (SC). The algorithms for the software of the MC and SC are explained here.

Main Controller Algorithm

The algorithm for MC software flow shown in Figure 3.6 can be explained as follows:

- A: The MC waits for the command word to be received via XBee from the remote controller.
- B: Once the data is available, it parses the command word (configuration word) from the XBee.
- C: The MC transmits the parsed word to the SC.

- D: The serial timeout for the SC is set to 1.5 times of the Sampling Time to avoid data congestion and long delays.
- E: The number of experiment iterations is calculated using Equation (3.1),

$$Experiment\ Iterations = \frac{Experiment\ Time}{Sample\ Time} \tag{3.1}$$

- F: It then waits for serial data coming from the SC.
- G: The MC performs the operation of parsing data stream for the number of Experiment Iterations calculated. Once the counter created to count the iterations reaches the value of Experiment Iterations, the MC stops the data receive-transmit functions and waits for a new command.
- H: Once the data is available, it is parsed and sent to the XBee.
- I: The XBee transmits the data to the User via the remote device. [15-18]

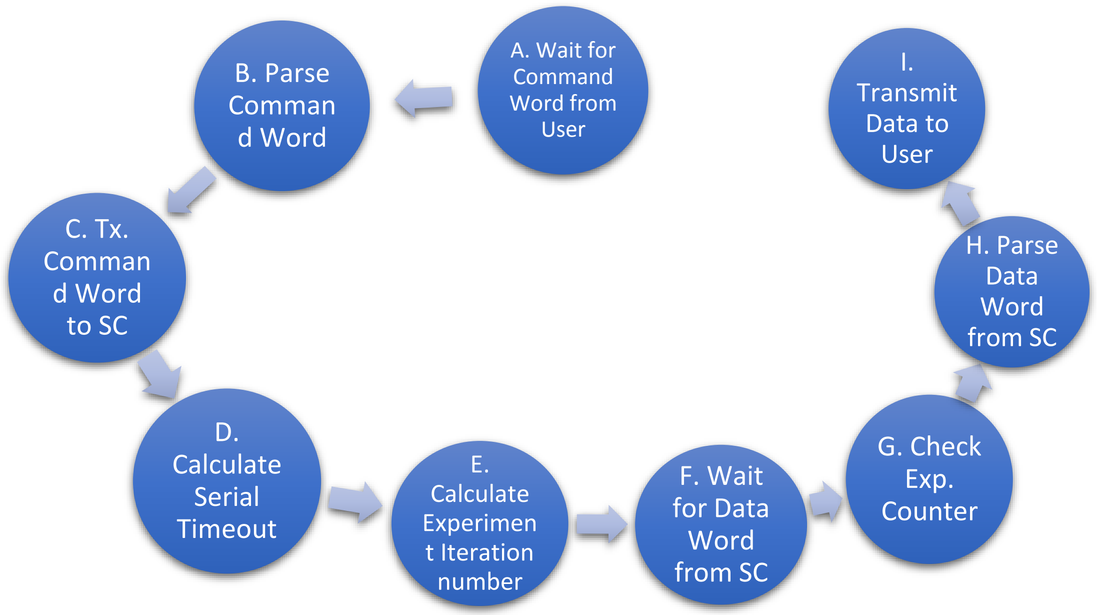

A. Wait for Command Word from User
B. Parse Comman d Word
C. Tx. Comman d Word to SC
D. Calculate Serial Timeout
E. Calculate Experimen t Iteration number
F. Wait for Data Word from SC
G. Check Exp. Counter
H. Parse Data Word from SC
I. Transmit Data to User

Figure 3.6 UTChattSat: Main Controller Software Algorithm

Sensor Controller Algorithm

The algorithm for sensor controller software flow shown in Figure 3.7 can be explained as follows:

- A: All sensors are initialized to a pre-defined baud rate and waits for command word to be available from the MC via serial communication.
- B: When available, the command word is parsed in three variables: Experiment Time, Sample Time and Sensor Configuration.
- C: Sample and Iteration counters are reset. The SC proceeds to calculate the number of experiment iterations by Equation (1).
- D: The SC compares the sample counter to the experiment iterations and if the counter has not reached the iterations value, the *takereading()* function is called and if it has reached the number of iterations, it stops the experiment and waits for serial data form the MC.
- E: *takereading()* subroutine:
    - E1: Decode sensor configuration word and set the sensors ON/OFF as per the word by comparing the value of 1 for bit level comparisons of the configuration word. The Sensor Configuration word is compared for each bit by logical AND function. Two switch case statements are designed, first for invoking sensors and retrieving data and second for setting the data values to zero for corresponding sensor values.
    - E2: Parse readings from sensors that are ON. The sensor values parsed by the microcontroller are converted to quantization levels by using the upper and lower limits of each sensing value and the resolution required by using Equation (3.2).

$$Step\ number = \frac{(Max-Min)}{Resolution} \tag{3.2}$$

Max and Min are the upper and lower limits of each sensor values the sensor can parse correctly. The resolution is the number of bits of resolution required, 16-bit resolution is used here.

- E3: Format the readings into the data word format.
- E4: Transmits the output array to the MC and return to the main program.

- F: After returning the function, a delay of sample time is generated and the experiment counter is incremented. [15-18]

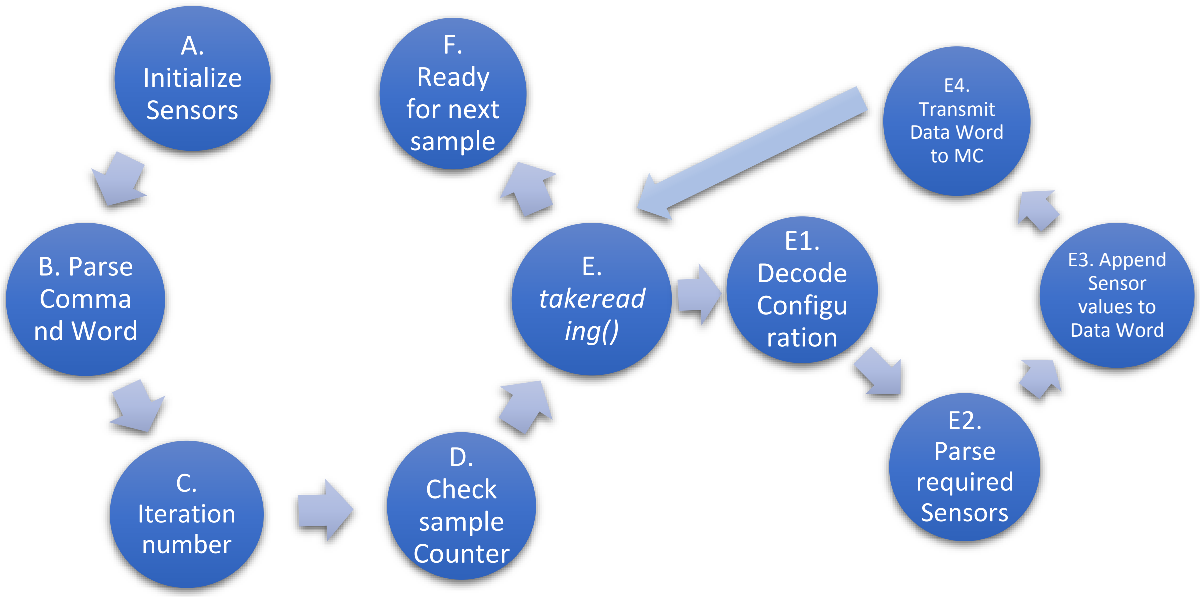


Figure 3.7 UTChattSat: Sensor Controller algorithm

Results

UTChattSat can be interfaced by the user's choice of technology and standards; however, two example interfaces are shown here. The first interface is with a local computer display and the second is through a website setup using a remote server and Raspberry Pi client. These two interfaces are explained independently in the following Passive Monitoring and Remote Monitor and Control sections:

Passive Monitoring System

An XBee connected to a computer within range provides a local interface of the UTChattSat system. The computer runs a MATLAB script that provides user inputs for Experiment Time, Sample Time and Sensor Configuration. The script opens the serial COM port through which the XBee is connected to the computer and enables communication. The command data input by the user is arranged in the command word format and transmitted to UTChattSat. Two figures are created, one for the 3D cube display and the other for plotting data in real time. A snapshot of the figures is shown in Figure 3.11. As the data streams to the computer, the 3D cube and all data plots are updated at the specified sampling rate set by the user. The 3D cube plot is a mirroring of the UTChattSat cube. As the UTChattSat is rotated along an axis in any directions (x, y, z or roll, pitch, heading), the 3D cube plot also rotates live from the gyroscope in the UTChattSat.

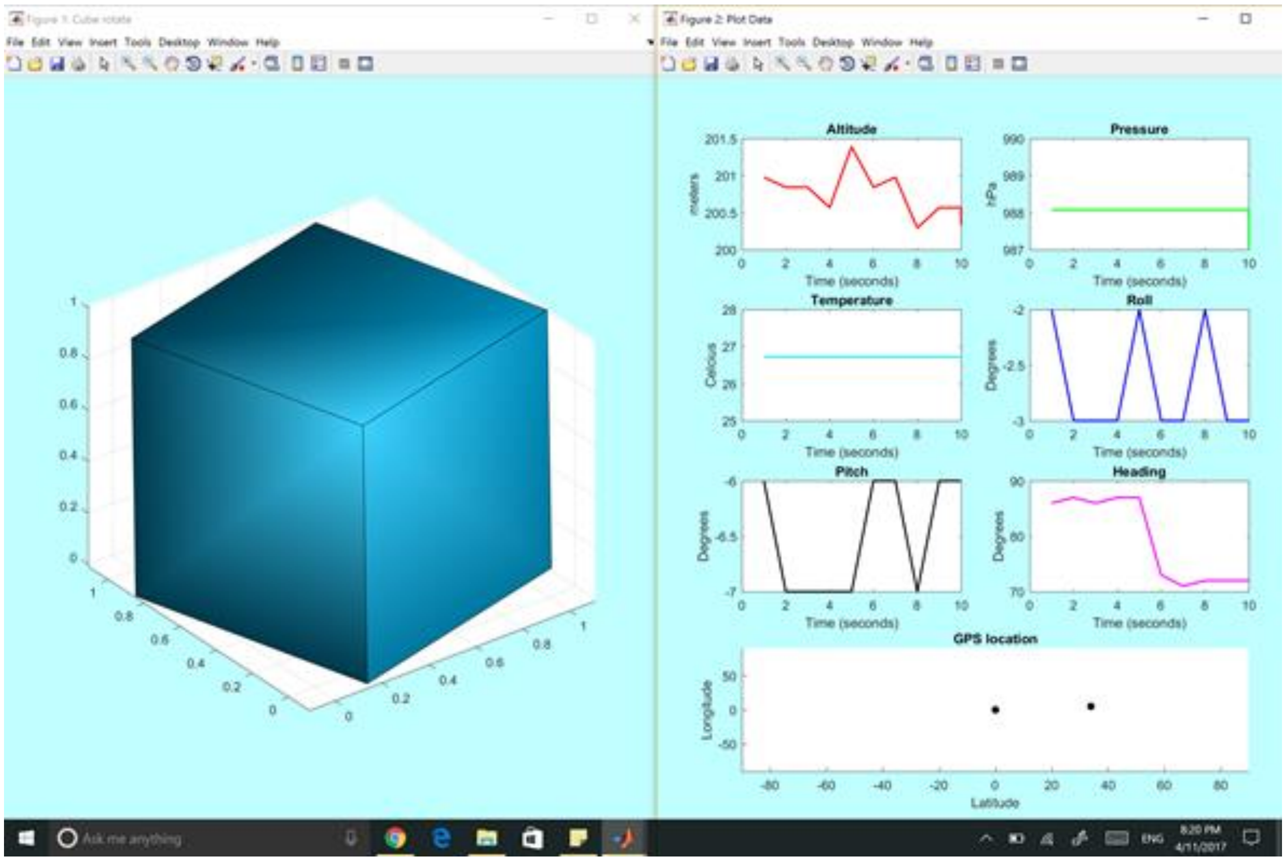
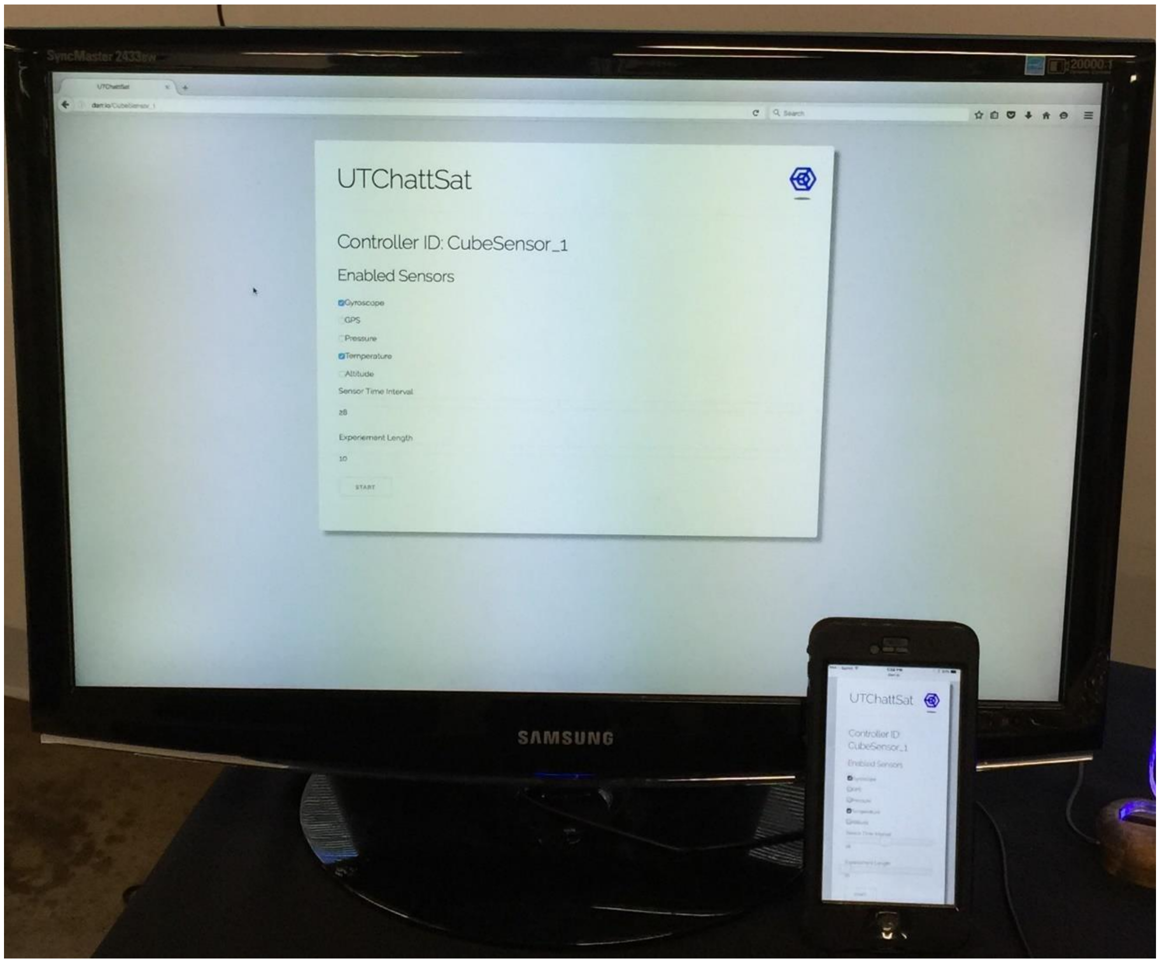


Figure 3.8 UTChattSat: Local Computer display. The left side shows a live display of cube orientation. The right side shows real-time plots for various entities measured at the cube location

Remote Monitor and Control

A server using JSON files can be setup in a computer connected to the Internet. A Raspberry Pi works as the client using a Python script. It receives the command from the user connected to the Internet via JSON communication files. It transmits this command word to UTChattSat via XBee. Note that the Raspberry Pi has an XBee connection and is co-located with the UTChattSat. The Python code then receives the data word and converts the step numbers to the actual data value by multiplying the step values to the received data that is in the form of step numbers.

Data is transmitted to the server using the JSON format and stored in text-based log files. The picture of the webpage is shown in Figure 3.12.



UTChattSat

Controller ID: CubeSensor_1

Enabled Sensors

- [✓] Gyroscope
- [ ] GPS
- [ ] Pressure
- [✓] Temperature
- [ ] Altitude

Sensor Time Interval

28

Experiement Length

10

Figure 3.9 UTChattSat: Webpage for IoT applications. The webpage can be accessed in a computer or phone

Conclusion

This example achieves the multi user HMI by enabling access control to multiple users with a unified hardware via the Internet. This can be used as a kit for education purposes as a student can use this platform to learn: how the controllers communicate with each other, the sensors, modify the hardware or software, or to add functionality to another device. This development kit is also used by local science schools in their ongoing experiments.

CHAPTER IV

EXAMPLE IMPLEMENTATION 2: DISTRIBUTED SENSOR NETWORKS

Introduction

The implementation of the distributed sensors single user multi node interface is used to provide data for evaluation of current building energy modeling techniques. This task requires the development of CPS-based support to assess the deficiencies within the existing building model. The hardware and software development for the CPS and the data collected are explained in succeeding sections.

Specifications:

- Measure Temperature, Pressure, Humidity
- Network covers important places of energy dissipation in the area to be monitored.
- Data collection for 24 hours with time stamp.
- Sample Rate: 4 to 5 samples/minute.
- Data resolution of least two decimal places.
- Data storage in text files separate for each sensor.
- No. of sensors: 8 mounted in EMCS 303.
- Quantities measured: Temperature, Pressure, Humidity
- Altitude is calculated from Pressure and stored.

- Placed at: Door, Ceiling Corner, Right Side Wall, Left Side Wall, Room Center, Vent, Between Windows, Window
- Date: February 2nd 2017

Block Diagram

The hardware of the distributed sensor networks (DSN) is shown in Figure 4.1. It comprises of five Main/Sensor Controllers, eight Sensors and XBees which all together are referred to as Node 1, 2, 3, 4, 5. The Raspberry Pi is designated as the Hub. The inter-connections for each node are shown in Table 4.1 (Node 2, 3, 4), Table 4.2 (Node 1) and Table 4.3 (Node 5). Each node is designated using the letters A through H and are all BME 280. The developed node is shown in Figure 4.2. Each sensor corresponds to a specific position in the laboratory as described in Table 4.4 and shown in Figure 4.3.

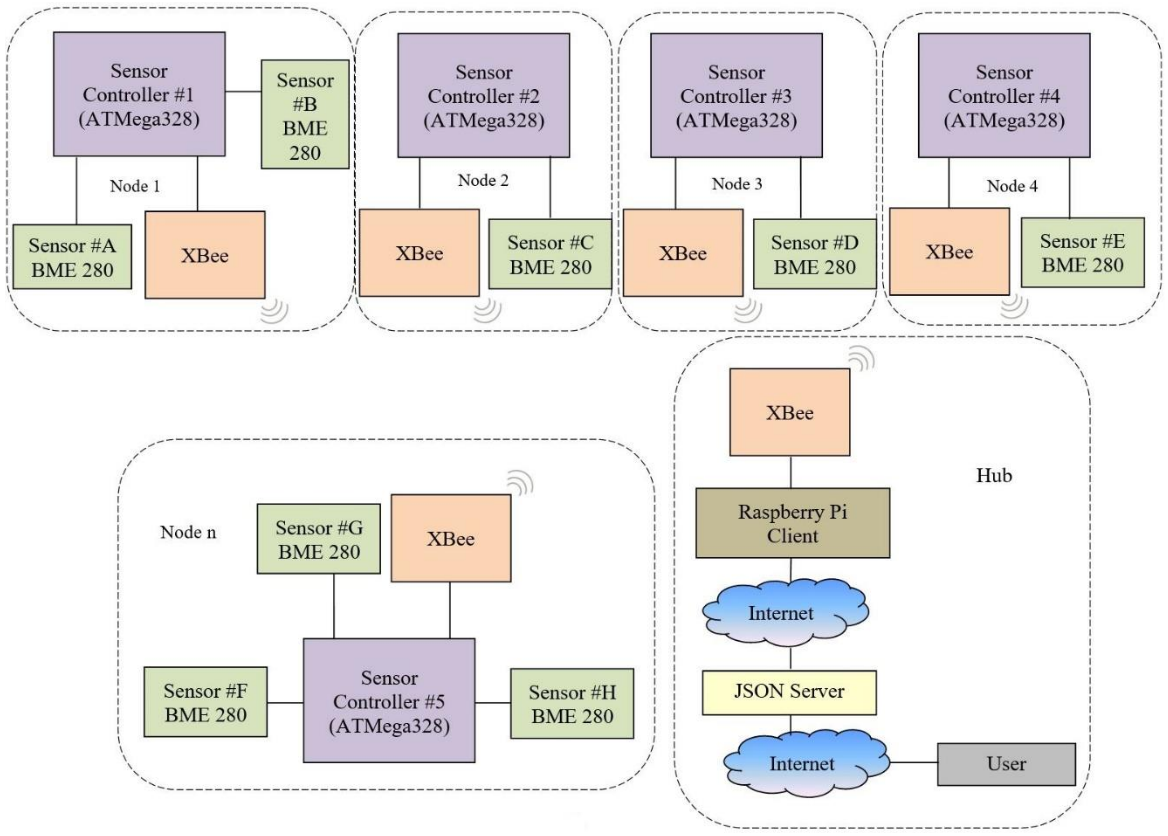


Figure 4.1 DSN: Block Diagram of full network for one laboratory with 8 sensing nodes and 1 hub. Sensors A to H are connected to the sensor controllers respectively

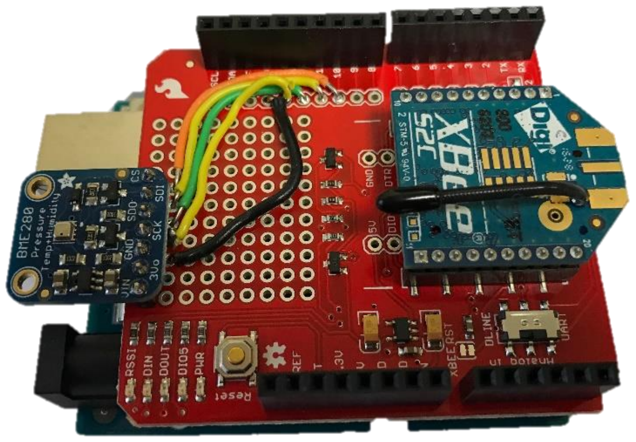


Figure 4.2 DSN: An Arduino UNO Sensor Controller connected to an XBee S2C and a BME 280 Sensor via Sparkfun Arduino Shield modified prototype area (Node Hardware)

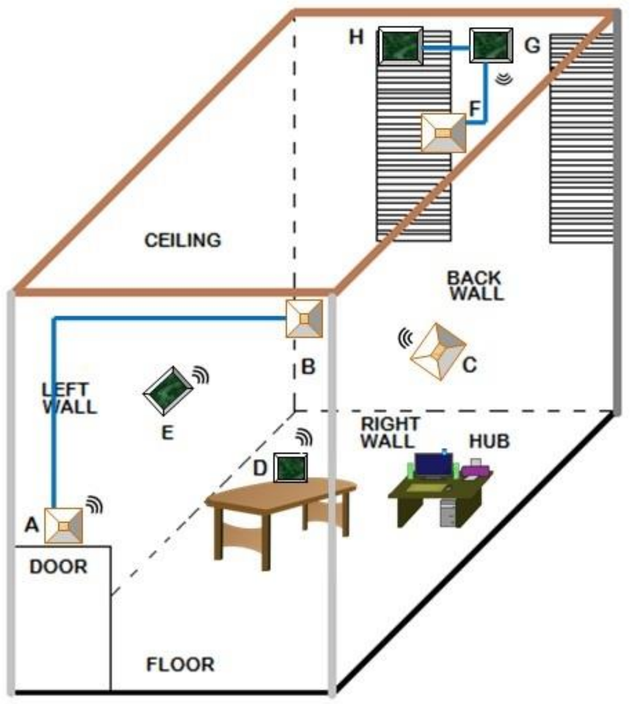


Figure 4.3 3D view of the laboratory with DSN mounted

Table 4.1 DSN: Node connection network. Arduino UNO, Adafruit BME280 and XBee interconnections for Nodes 2, 3 and 4

Arduino UNO # 2, 3, 4	**Adafruit BME 280 (C, D, E)**	**XBee**
Vin	-	Vin
3.3 V	Vin	-
GND	GND	GND
Tx (Dig1)	-	DOUT
Rx (Dig0)	-	DIN
Digital 13	SCK	-
Digital 12	SDO	-
Digital 11	SDI	-
Digital 10	CS	-

Table 4.2 DSN: Node connection network. Arduino UNO, Adafruit BME280 and XBee interconnections for Node 1

Arduino UNO # 1	**Adafruit BME 280 (A)**	**Adafruit BME 280 (B)**	**XBee**
Vin	-	-	Vin
3.3 V	Vin	Vin	-
GND	GND	GND	GND
Tx (Dig1)	-	-	DOUT
Rx (Dig0)	-	-	DIN
Digital 13	SCK	SCK	-
Digital 12	SDO	SDO	-
Digital 11	SDI	SDI	-
Digital 10	CS	-	-
Digital 9	-	CS	-

Table 4.3 DSN: Node connection network. Arduino UNO, Adafruit BME280 and XBee interconnections for Node 5

Arduino UNO # 5	**Adafruit BME 280 (F)**	**Adafruit BME 280 (G)**	**Adafruit BME 280 (H)**	**XBee**
Vin	-	-	-	Vin
3.3 V	Vin	Vin	Vin	-
GND	GND	GND	GND	GND
Tx (Dig1)	-	-	-	DOUT
Rx (Dig0)	-	-	-	DIN
Digital 13	SCK	SCK	SCK	-
Digital 12	SDO	SDO	SDO	-
Digital 11	SDI	SDI	SDI	-
Digital 10	-	CS	-	-
Digital 9	-	-	CS	-
Digital 8	CS	-	-	-

Table 4.4 DSN: Sensor A through H positions in the laboratory

Sensor	**Location**	**Node**
A	At Door	1
B	Ceiling corner	1
C	Right Side Wall	2
D	Room Center	3
E	Left Side Wall	4
F	At Vent	5
G	Between Windows	5
H	At Window	5

The sensor board BME280 is described in the following sub-section. This breakout board was selected in this implementation as they carry the sensors, interfacing controllers and pinouts. Moreover, these boards have libraries that can be used in the software algorithms developed for the Main/Sensor Controller (ATmega328). The BME280 are low cost precision sensor with accuracy adequate for this application.

Adafruit BME280

The BME 280 is a precision sensor, made by Bosch, that measures humidity with ±3%, barometric pressure with ±1 hPa, and temperature with ±1.0°C. As pressure changes with altitude, it can be used as an altimeter with ±1 meter accuracy, with a low altitude noise of 0.25m. The Adafruit BME 280 is assembled using the three sensors (temperature, pressure and humidity), which provides pinout, I2C, and SPI interface. It is shown in Figure 4.4 [19].

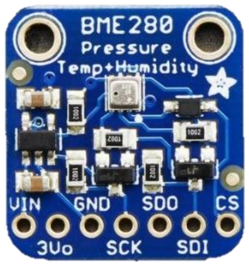


Figure 4.4 Adafruit BME280 sensor

Main/Sensor Controller Software Development

The algorithm for Main/Sensor Controller #1 is shown in Figure 4.5. The same logical flow is used for all main/sensor controllers (M/SC). The software development for M/SC is explained as follows:

- A: The M/SC waits until RPi (Raspberry Pi) broadcasts the sensor IDs.
- B: Once a sensor ID is available, it is parsed.
- C, D: A switch case statement is used to switch through the sensors connected and parse the corresponding sensor. When the sensor ID does not match to the sensors attached, it is neglected and waits for the next sensor ID.
- E: This data is transmitted to Raspberry Pi in float data types in the format of data word for single user multi node interface as explained in Chapter 2.

The M/SC repeats the flow for every code received. The sampling rate is controlled by the Raspberry Pi hub software. Hence, one data reception and transmission takes approximately 13 seconds to complete one cycle including data fetch and store by Raspberry Pi. This state a sampling rate of 4 to 5 samples per minute. This is a two-way communication with the hub as master and the node as slave [15-18].

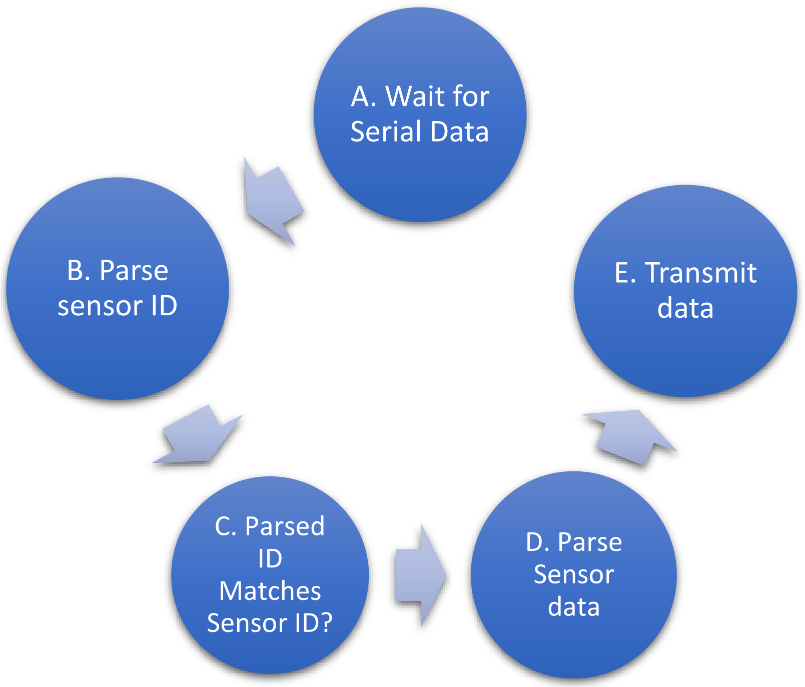


Figure 4.5 DSN: Node Software Development Algorithm

Hub Software Development

A Python program is composed using the Hub Software algorithm. The main outline algorithm is shown in Figure 4.6. The software development of hub is explained as following:

- A: At startup, the Python configures the '/dev/ttyUSB0' port for XBee communication at 9600 baud rate. Then creates and/or open 'Sensor Log' directory at specified path and creates separate files for each sensor with date and timestamp on the file name, writes the start time and date stamp to each log file and initializes the text files.
- B, C: The sensor IDs from 101 to 108 are transmitted and incoming data is read for every code.

- D: The incoming data read is stored in a variable called RxSamp. The size of RxSamp is measured to detect and discard corrupted samples and flush the input buffer of XBee.
- E: The RxSamp is split into array with 5 elements by detecting ‘,’. The first element, the sensor ID is an alphabet from A to H for corresponding sensors.
- F: The sensor ID is used for comparison and mapping to the correct log file for the corresponding sensor. Also, the subroutine runs an error counter whenever corrupted data is received.
- G: The timestamp is compared to 23:59:59 at the end of each code transmission and data reception iteration. Once the timestamp reaches the comparison value, new files are created for logging. At midnight, when new files are created, the total number of errors for that day is written to a new error log file created each day [15-18].

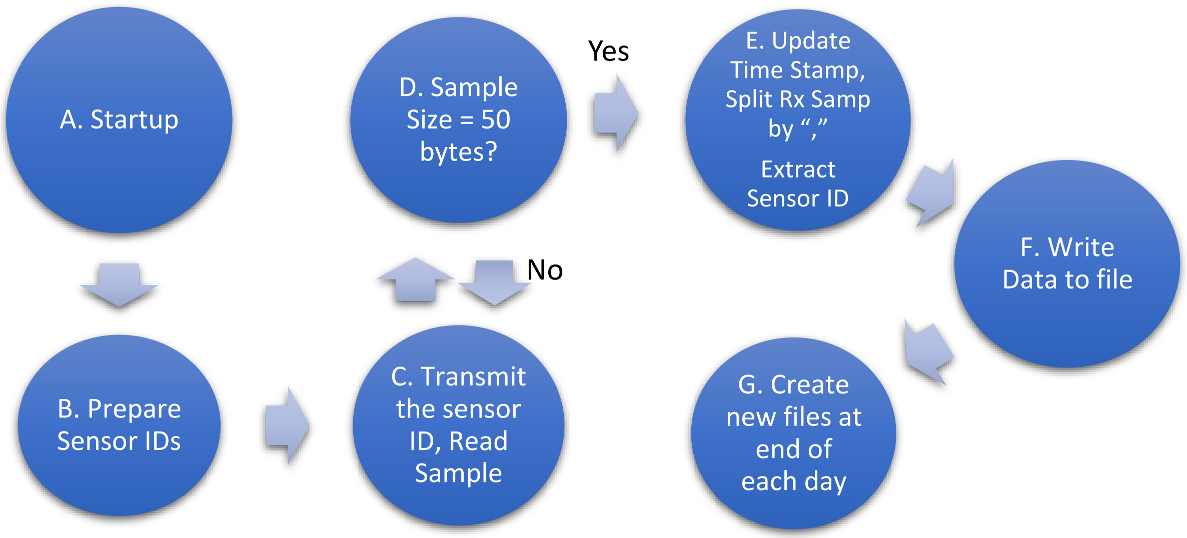

A. Startup
D. Sample Size = 50 bytes?
Yes
E. Update Time Stamp, Split Rx Samp by “,” Extract Sensor ID
F. Write Data to file
No
B. Prepare Sensor IDs
C. Transmit the sensor ID, Read Sample
G. Create new files at end of each day

Figure 4.6 DSN: Hub Software Algorithm

Results

The data collected by the DSN are temperature, pressure and humidity measurements at 8 spots in a laboratory. These data will be used to assess current building energy models. The data is logged for 24 hours and runs continuously for the duration required and stored in separate text files for each sensor named by the Sensor ID, the date and time stamp when it was created. A sample of data shown in following figures is logged on February 2nd, 2017 in the laboratory for 24 hours.

Figure 4.7 shows the temperature plots from data measured by respective sensors. The sensor at the door (A) is mounted exactly above the door such that the energy dissipation due to open/close movement of the door can be determined. The sensor at ceiling corner (B) senses the conditions near the ceiling to determine the dissipation due to elevation of buildings. The sensors at right side wall (C) and left side wall (E) can be used to determine the dissipation due to the walls and adjacent rooms/area. The sensor at room center (D) can determine the average temperature and other conditions in the room, whereas the sensor at the A/C vent (F) senses the input energy to the room. The sensor between windows (G) senses the affect due to outside wall and environment. The sensor at window (H) can be used to determine the possible highest dissipation area in the room. All these sensors can be used to create a energy model of the room and determine the highest and lowest points of the room for power dissipation and energy absorption.

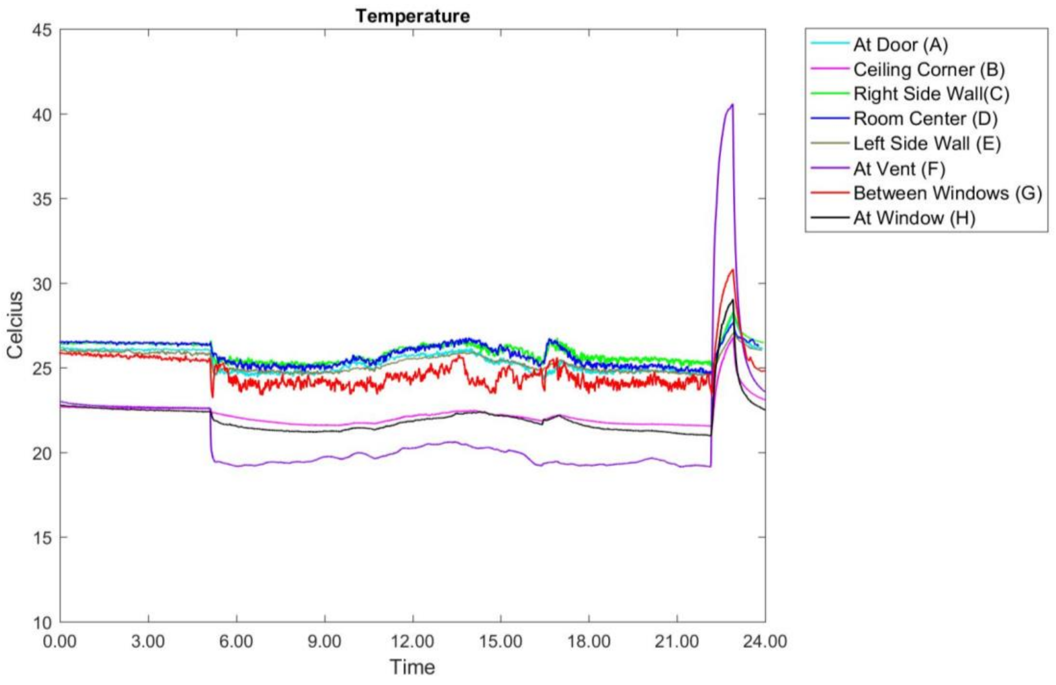


Figure 4.7 Temperature plots from data measured at eight spots across the room on February 2nd, 2017. The placement of the sensors is described in the legend and shown in Figure 4.3

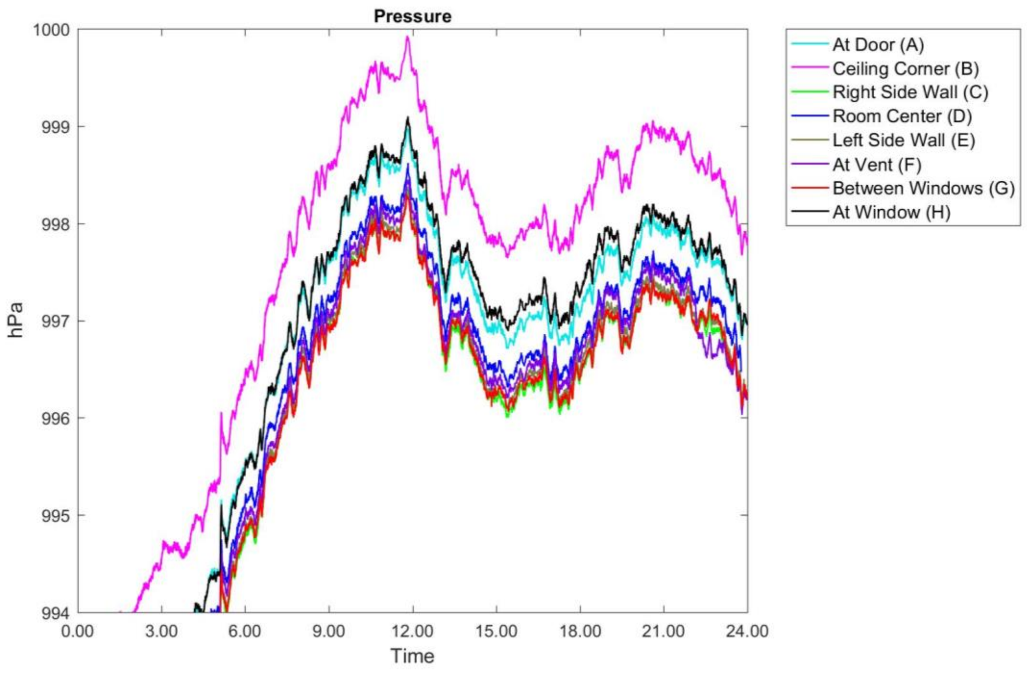


Figure 4.8 Pressure plots from data measured at eight spots across the room on February 2nd, 2017. The placement of the sensors is described in the legend as shown in Figure 4.3

As seen in these data, the temperature at the vent has the most deviation, as it is the area most affected by the temperature change. The Pressure plots can be seen in Figure 4.8. The pressure data has a similar trend for all locations. The offsets may be due to the sensor inaccuracy which is discussed in next chapter. Humidity plots can be seen in Figure 4.9. They are inverse functions of the Temperature plots. Humidity levels at the vent vary indirect proportion to the Temperature variation.

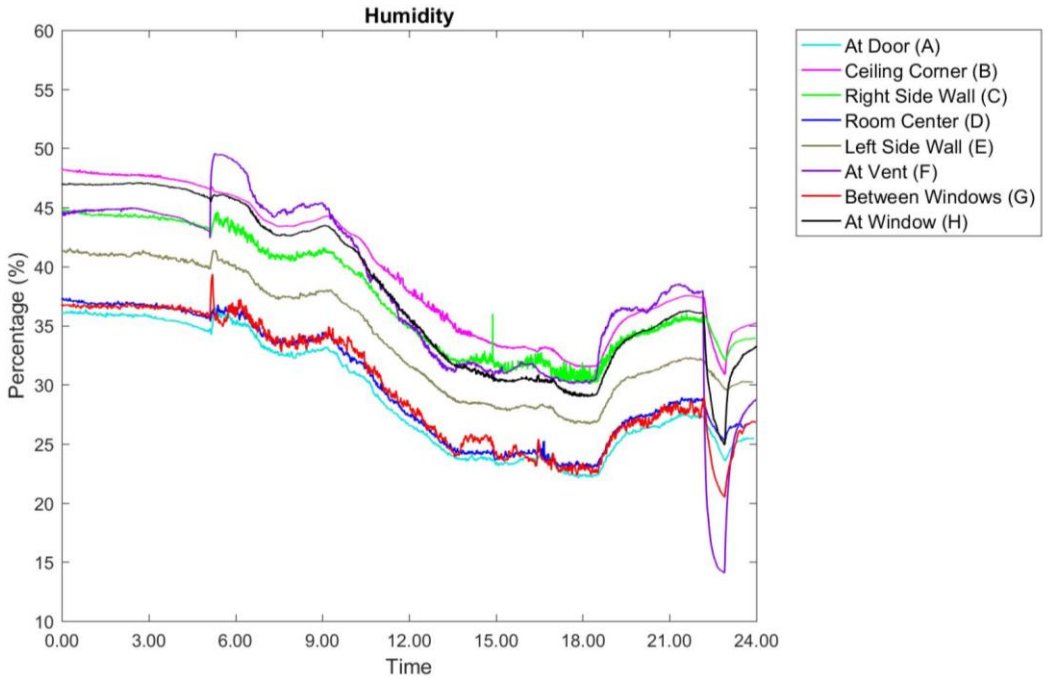


Figure 4.9 Humidity plots from data measured at eight spots across the room on February 2nd, 2017. The placement of the sensors is described in the legend as shown in Figure 4.3

Conclusion

The implementation of the distributed sensors single user multi node interface is used to provide data for evaluation of current building energy modeling techniques. The CPS-based multi node interface is development to assess the existing model. The web-based control of the interface can be used for industrial and professional research for personalized experiments.

CHAPTER V

CALIBRATION

Argumentation

Figure 5.1 shows the data from Figure 4.7, spanning the period of 4 hours from 5:30 - 9:30 am. It shows two different variation groups within the data. Three sensors, i.e. Ceiling Corner (B, $\sigma^2 = 0.64$), At Window (H, $\sigma^2 = 1.41$) and At Vent (F, $\sigma^2 = 1.98$) are much more stable than the rest of the sensors. The three sensors are connected to the main controller and communication device using cables (off board) whereas the other five sensors are mounted on the electronics directly. Hence, the thermal noise of the electronics adds measurement noise and the sensor data may not be accurate. Table 5.1 shows the mean and standard deviation for each sensor.

Table 5.1 February 2nd, 2017 Temperature Data: Mean and Standard Deviation

Sensor	Location	Configuration	Mean	Std. Dev.
Sensor A	At Door	On Board	25.39	0.70
Sensor B	Ceiling Corner	Off Board	22.27	0.80
Sensor C	Right Side Wall	On Board	25.92	0.59
Sensor D	Room Center	On Board	25.78	0.68
Sensor E	Left Side Wall	On Board	25.31	0.56
Sensor F	At Vent	Off Board	21.26	3.61
Sensor G	Between Windows	On Board	24.82	1.14
Sensor H	At Window	Off Board	22.07	1.19

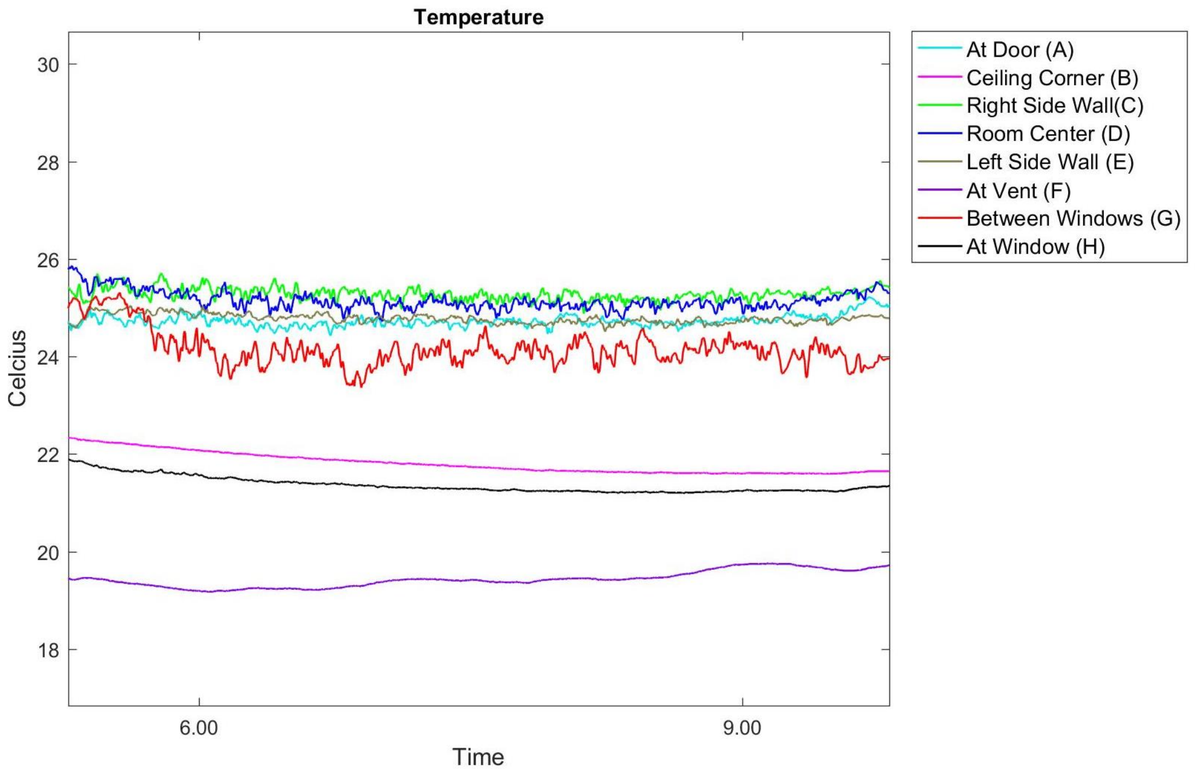


Figure 5.1 Temperature data for February 2nd, 2017 fr0m 6:00 am to 9:00 am. Clear difference can be seen between the sensors that are connected through different hardware configurations

Experiment Details and Results

In order to quantify the bias in these data, a set of calibration experiments were conducted where the sensors were mounted in a controlled environment alongside a K-type thermocouple. Initially, the cold start characteristics of sensors A to H were measured. The sensors were connected to respective microcontrollers powered from the same power supply and all sensors, regardless of on-board or off-board mounting, exhibit increasing temperature immediately following a cold start due to self-heating of sensors and the electronics as shown in Figure 5.2. All sensors appear to asymptotically approach a stable value. Off board sensors show an average initial slope of 0.0583°C per minute whereas on-board sensors show an average initial slope of 0.1452°C per minute. The sensors exponentially approach a stable value.

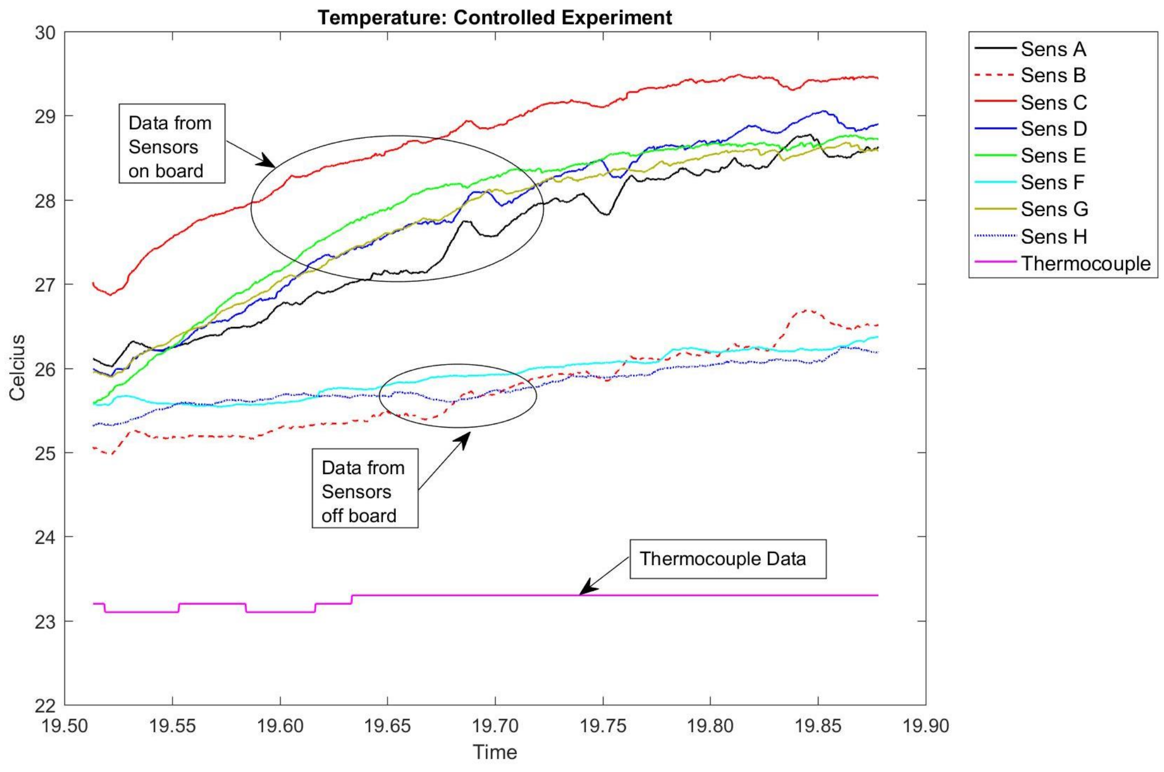


Figure 5.2 Calibration Experiment Results: Test Run. Powered ON without warm up

Two additional calibration experiments were performed as described below:

Experiment 1:

- No. of sensors: 8 placed at a same location in a room.
- Sampling Rate: 1 sample/second.
- Temperature reference device: K-type Thermocouple.

- Quantities measured: Temperature, Pressure, Humidity. Altitude is calculated from Pressure and stored.
- Date: February 10th 2017.
- Data collected for 20 minutes after warm up of 10 minutes and cool down of 5 minutes between each configuration.
- A summary of findings for each sensor is provided for the following configurations:
  1. Sensor A: On Board (Figure 5.3).
     - ➢ Microcontroller#1 connected to Sensor A on board only.
     - ➢ Sensor A On board has a average bias of 4.77°C.
  2. Sensor C: On Board (Figure 5.4).
     - ➢ Microcontroller#2 connected to Sensor C on board only.
     - ➢ Sensor C On board has an avereage bias of 4.85°C.
  3. Sensor A: On Board, Sensor B: Off Board (Figure 5.5).
     - ➢ Microcontroller#1 connected to Sensor A on board and Sensor B off board.
     - ➢ Sensor A On board has a increased bias of 5.34°C when compared to configuration 1.
     - ➢ Sensor B Off board has an avereage bias of 3.58°C.
  4. Sensor F: On Board, Sensor G: Off Board, Sensor H: Off Board (Figure 5.6).
     - ➢ Microcontroller#5 connected to Sensor F on board, Sensor G and Sensor H off board.
     - ➢ Sensor F On board has an avereage bias of 4.71°C
     - ➢ Sensor G Off board has an avereage bias of 3.29°C
     - ➢ Sensor H Off board has an avereage bias of 3.34°C

As seen in Figure 5.3 and 5.4, different microcontrollers connected to individual sensors on board configuration exhibit similar bias in temperature, i.e. 4.77°C for Sensor A and 4.85°C for Sensor C. When off-board Sensor B is included alongside Sensor A as shown in Figure 5.5, the bias in Sensor A increases to 5.34°C. Microcontroller #2 is connected to Sensor G, H and F. The bias for on board Sensor F is 4.71°C and off board Sensors G and H is 3.29°C and 3.34°C respectively. The off board bias values are similar for both microcontrollers [20]. Table 5.2 summarizes the data.

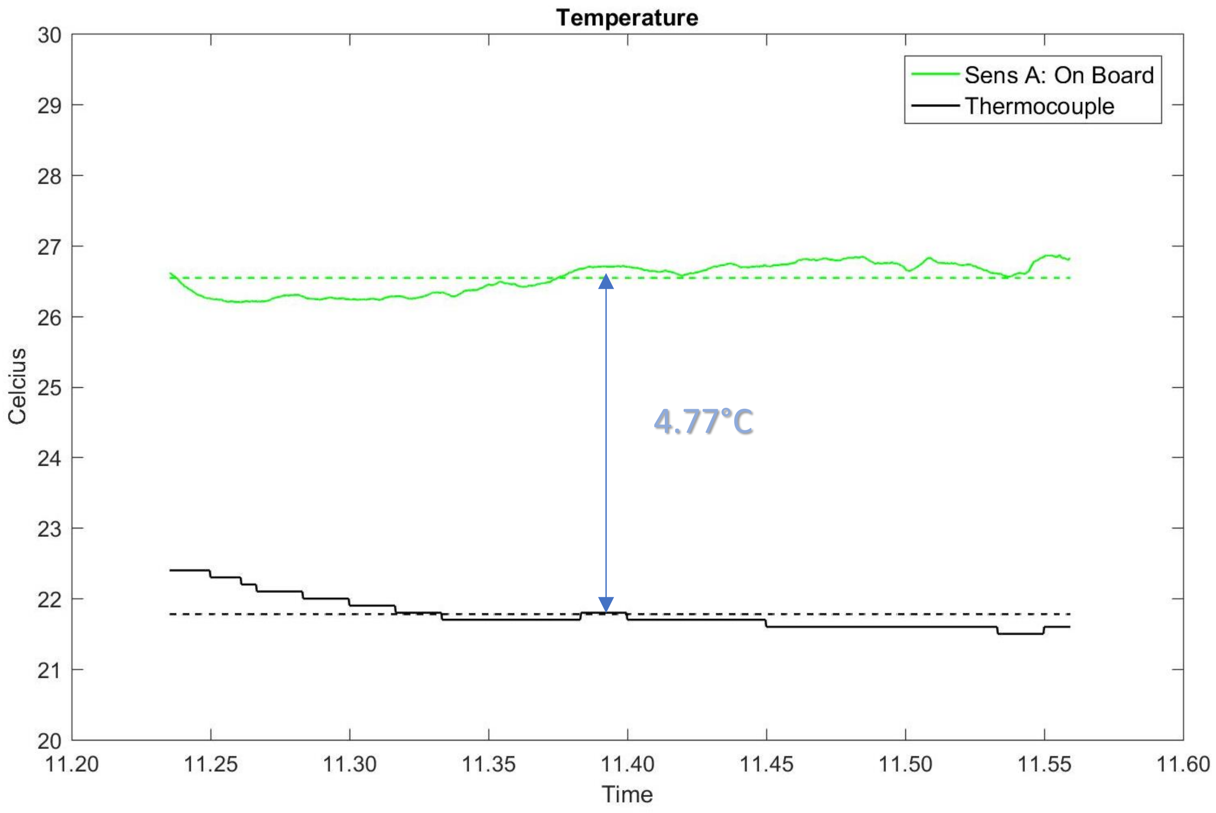


Figure 5.3 Calibration Experiment 1: Microcontroller #1 connected to only Sensor A

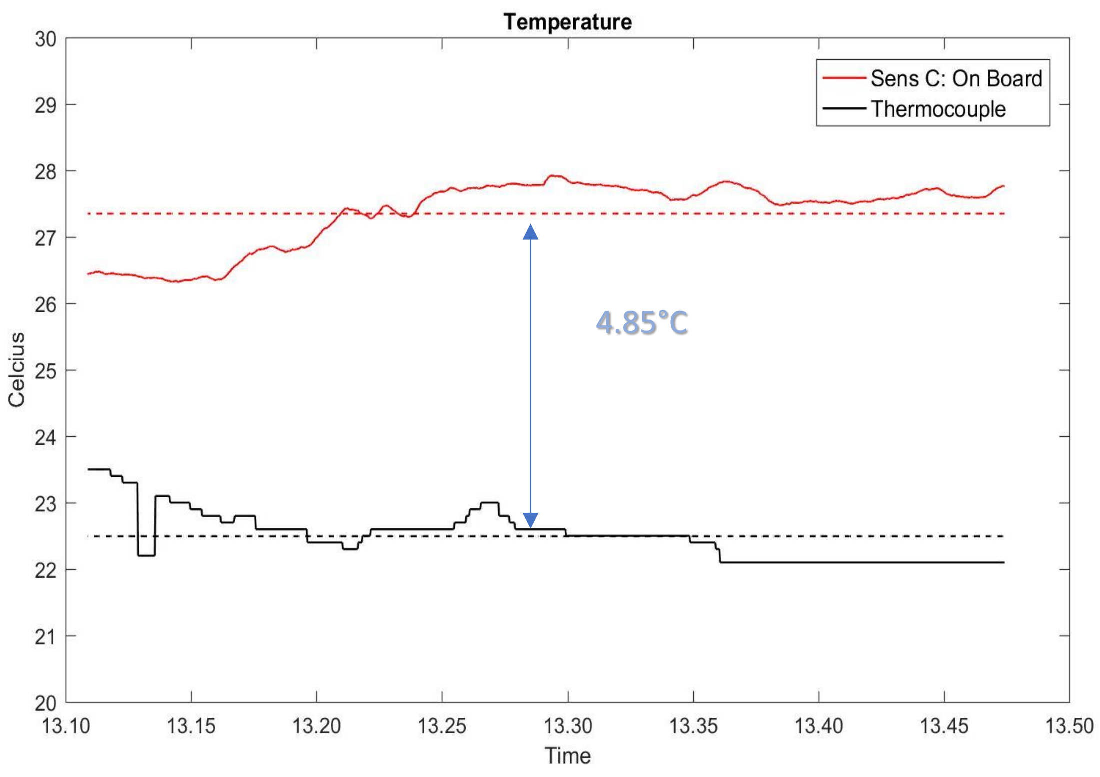


Figure 5.4 Calibration Experiment 1: Microcontroller #2 connected to only Sensor C

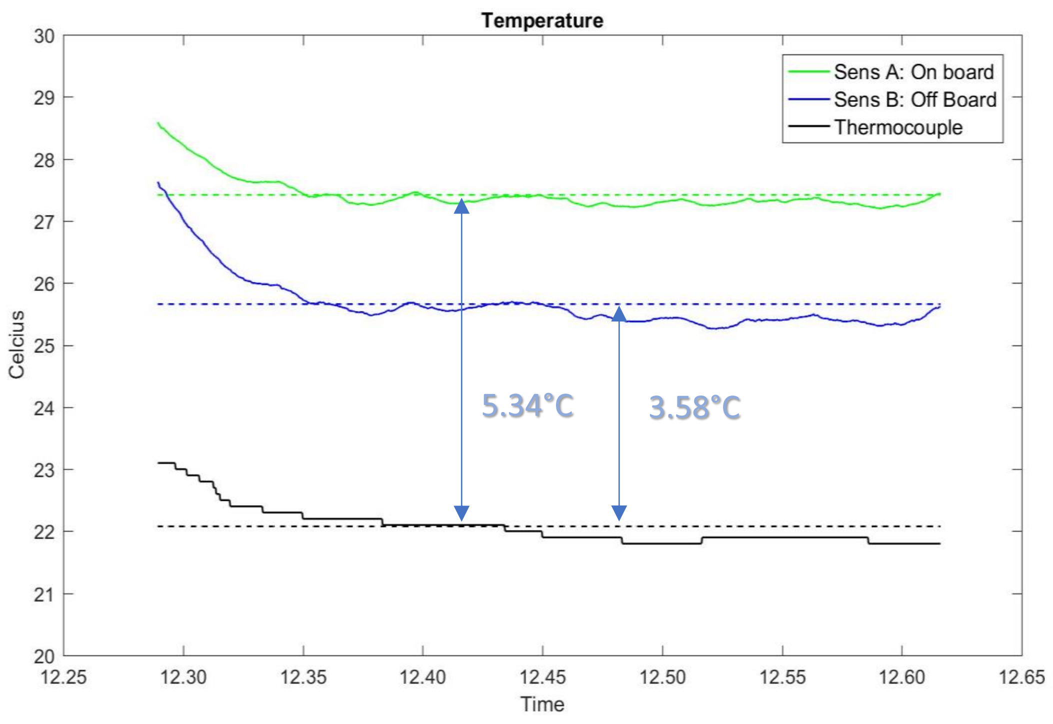


Figure 5.5 Calibration Experiment 1: Microcontroller #1 connected to Sensor A & B

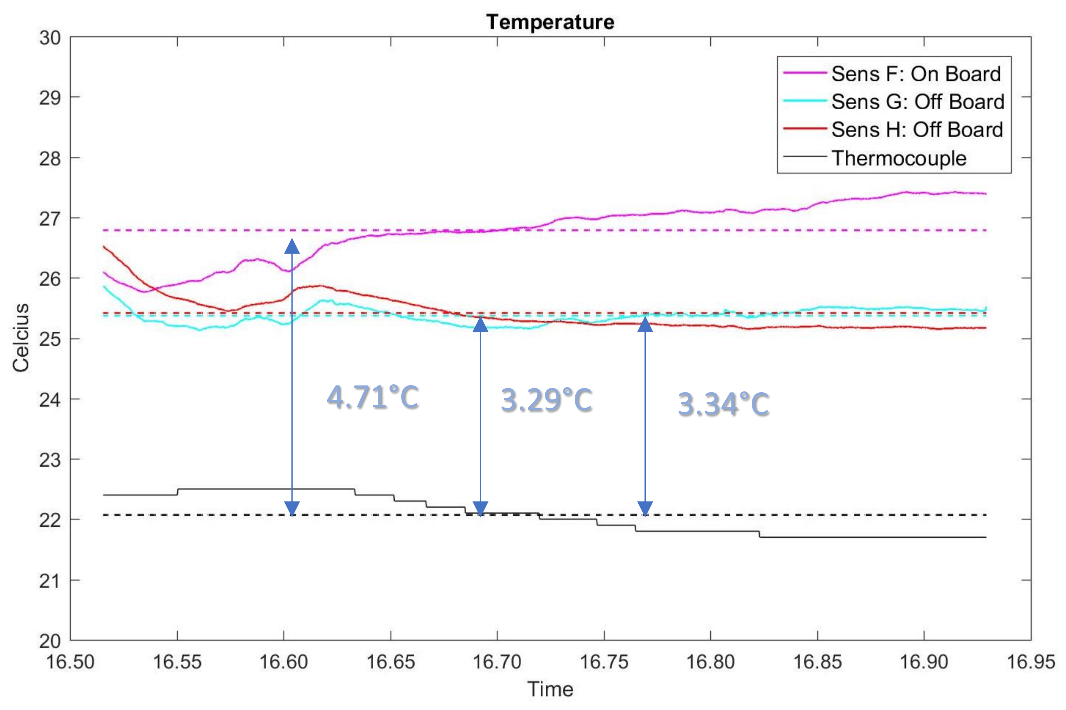


Figure 5.6 Calibration Experiment 1: Microcontroller #2 connected to Sensor F, G & H

Table 5.2 Sensor Calibration Experiment 1: Accuracy and Standard Deviation

Sensor	**Accuracy**	**Std. Dev.**
Sensor A	5.37	0.21
Sensor B	3.58	0.44
Sensor C	4.85	0.49
Sensor D	5.08	0.65
Sensor E	5.28	0.17
Sensor F	4.71	0.47
Sensor G	3.29	0.13
Sensor H	3.34	0.28
Thermocouple	--	0.07

Experiment 2:

- No. of sensors: 6 placed at a same place in a room.
- Sampling Rate: 1 sample/second.
- Temperature reference device: K-type Thermocouple.
- Quantities measured: Temperature, Pressure, Humidity. Altitude is calculated from Pressure and stored.
- Date: May 29th 2017.
- Data collected for 20 minutes after warm up of 10 minutes and cool down of 5 minutes between each configuration.
- A summary of findings for each sensor is provided for the following configurations:

1. Sensor A: On Board (Figure 5.7).
    - ➢ Microcontroller#1 connected to Sensor A on board only.
    - ➢ Sensor A has an average bias of 5.10°C.
2. Sensor A: On Board, Sensor B: Off Board (Figure 5.8).
    - ➢ Microcontroller#1 connected to Sensor A on board and Sensor B off board.
    - ➢ Sensor A On board has a decreased bias of 4.46°C.
    - ➢ Sensor B Off board has a bias of 0.90°C.
3. Sensor A: On Board, Sensor B: Off Board, Sensor C: Off Board (Figure 5.9).
    - ➢ Microcontroller#1 connected to Sensor A on board, Sensor B and Sensor C off board.
    - ➢ Sensor A On board has a decreased bias of 3.33°C.
    - ➢ Sensor B Off board has a decreased bias of 0.49°C.
    - ➢ Sensor C Off board has a negative bias of -0.49°C

4. Sensor G: On Board (Figure 5.10).
    - ➢ Microcontroller#5 connected to Sensor G on board only.
    - ➢ Sensor G On board has a bias of 4.85°C.
5. Sensor G: On Board, Sensor H: Off Board (Figure 5.11).
    - ➢ Microcontroller#5 connected to Sensor G on board and Sensor H off board.
    - ➢ Sensor G On board has a decreased bias of 3.01°C.
    - ➢ Sensor H Off board has a bias of 0.06°C.
6. Sensor G: On Board, Sensor H: Off Board, Sensor F: Off Board (Figure 5.12).
    - ➢ Microcontroller#5 connected to Sensor G on board, Sensor H and Sensor F off board.
    - ➢ Sensor G On board has a increased bias of 5.49°C.
    - ➢ Sensor H Off board has a bias of 1.79°C.
    - ➢ Sensor F Off board has a bias of 2.71°C.

- Table 5.3 summarizes the accuracy and precision data for Experiment 2.

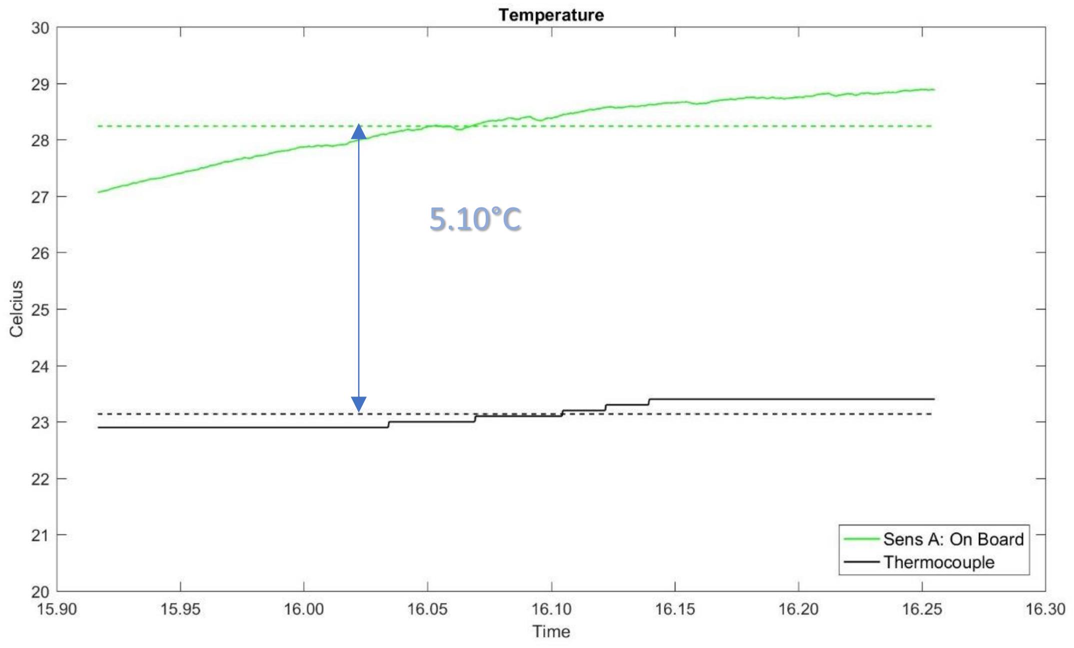


Figure 5.7 Calibration Experiment 2: Microcontroller #1 connected to Sensor A

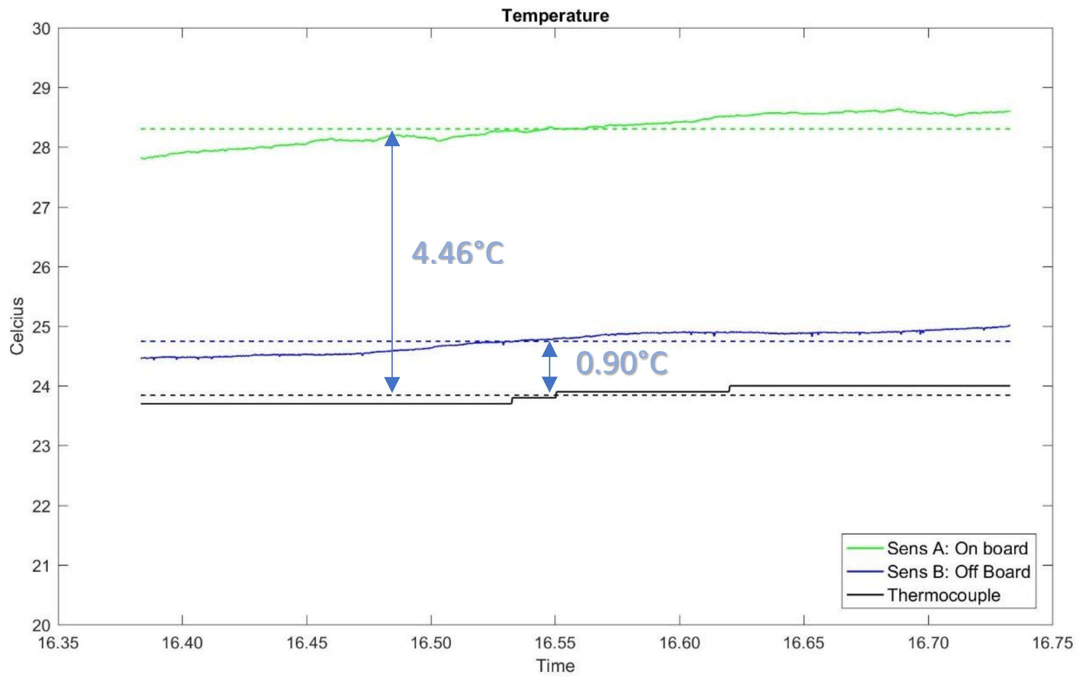


Figure 5.8 Calibration Experiment 2: Microcontroller #1 connected to Sensor A & B

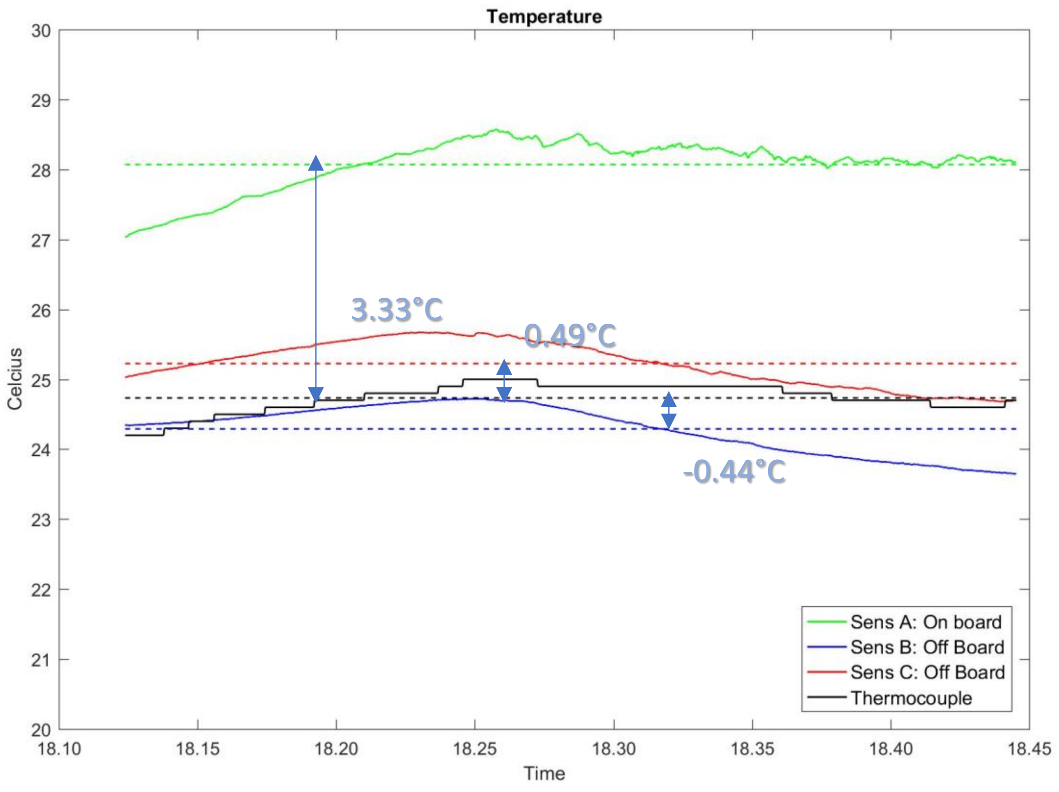


Figure 5.9 Calibration Experiment 2: Microcontroller #1 connected to Sensor A, B & C

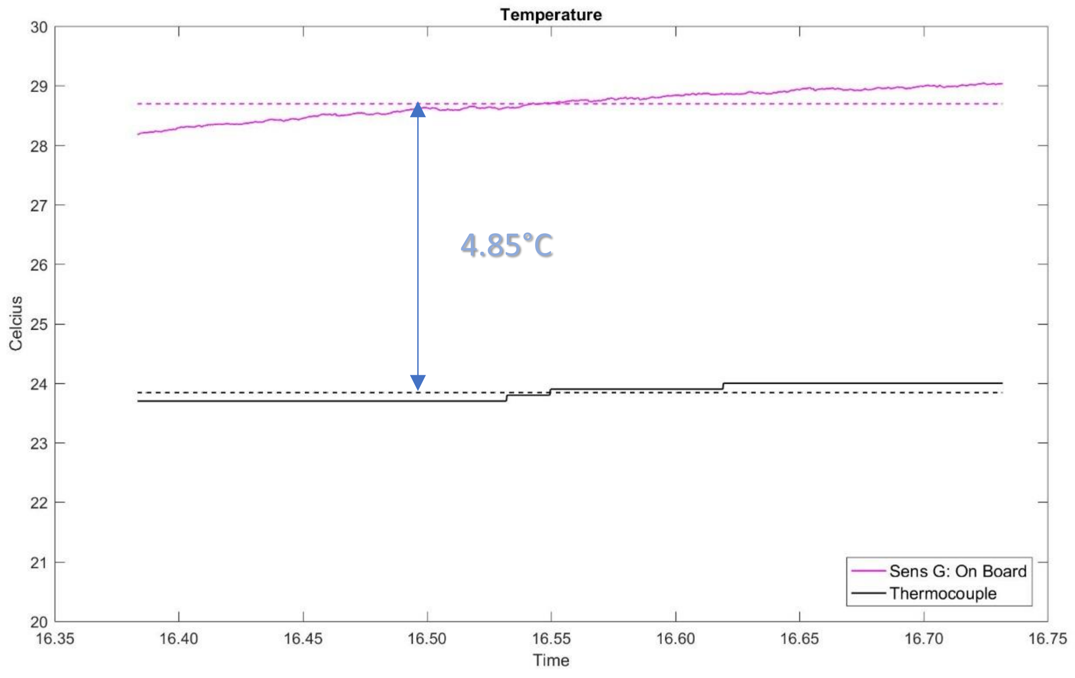


Figure 5.10 Calibration Experiment 2: Microcontroller #5 connected to Sensor G

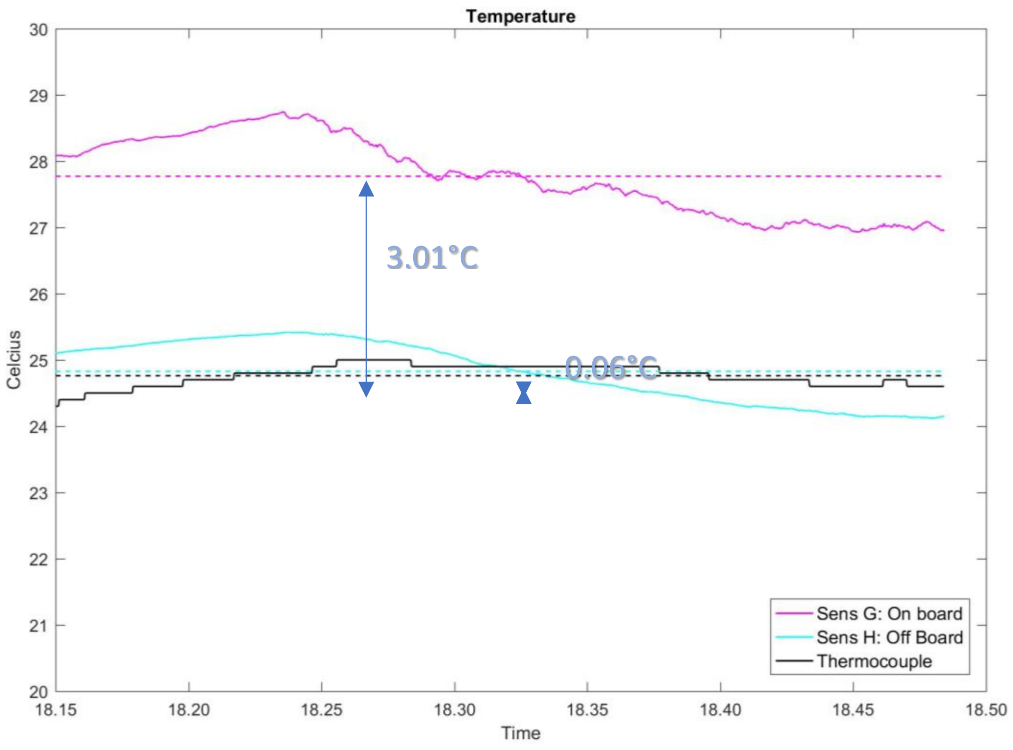


Figure 5.11 Calibration Experiment 2: Microcontroller #5 connected to Sensor G & H

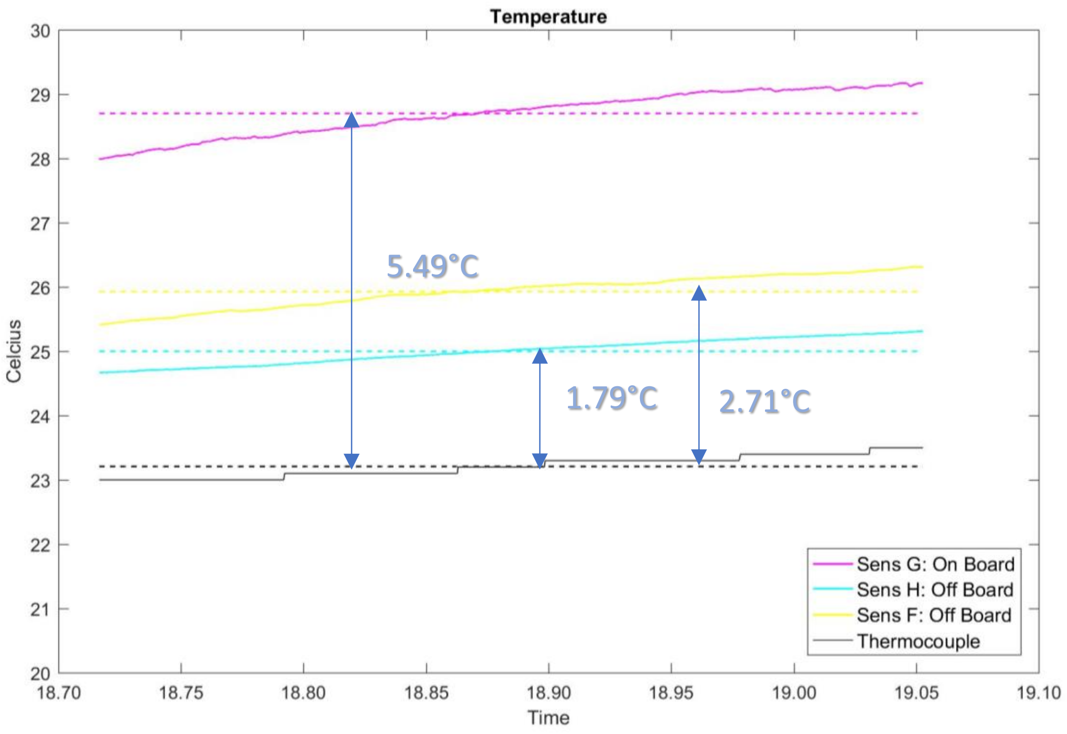


Figure 5.12 Calibration Experiment 2: Microcontroller #5 connected to Sensor G, H and F

Table 5.3 Sensor Calibration Experiment 2: Accuracy and Standard Deviation

Group	Sensor	Configuration	Accuracy	Std. Dev.
1	Sensor A	On Board	5.10	0.51
	Thermocouple	--	--	0.21
2	Sensor A	On Board	4.46	0.23
	Sensor B	Off Board	0.94	0.17
	Thermocouple	--	--	0.13
3	Sensor A	On Board	3.33	0.36
	Sensor B	Off Board	-0.44	0.34
	Sensor C	Off Board	0.49	0.31
	Thermocouple	--	--	0.20
4	Sensor G	On Board	4.85	0.23
	Thermocouple	--	--	0.13
5	Sensor G	On Board	3.01	0.58
	Sensor H	Off Board	0.06	0.45
	Thermocouple	--	--	0.15
6	Sensor G	On Board	5.49	0.34
	Sensor H	Off Board	1.79	0.19
	Sensor F	Off Board	2.71	0.24
	Thermocouple	--	--	0.15

Conclusion

Even though the experiment is run for different sensors in the similar configurations, the specific behavior cannot be determined based on the bias in temperatures. This shows the inconsistency of the sensor accuracy. There is no predicable behavior, but these data can be used to create distributions of the real-time data to estimate uncertainty.

From the derived standard deviations, error bars are plotted for each sensor plot as shown in Figure 5.13 and Figure 5.14 (from 5:30 – 9:30 am). For the sensor data group (with thermal noise) at the top overlaps within the error bars. It can be determined from this behavior that the data sensed is not accurate and the sensor data, with overlapping error bars, has offsets in the measurements.

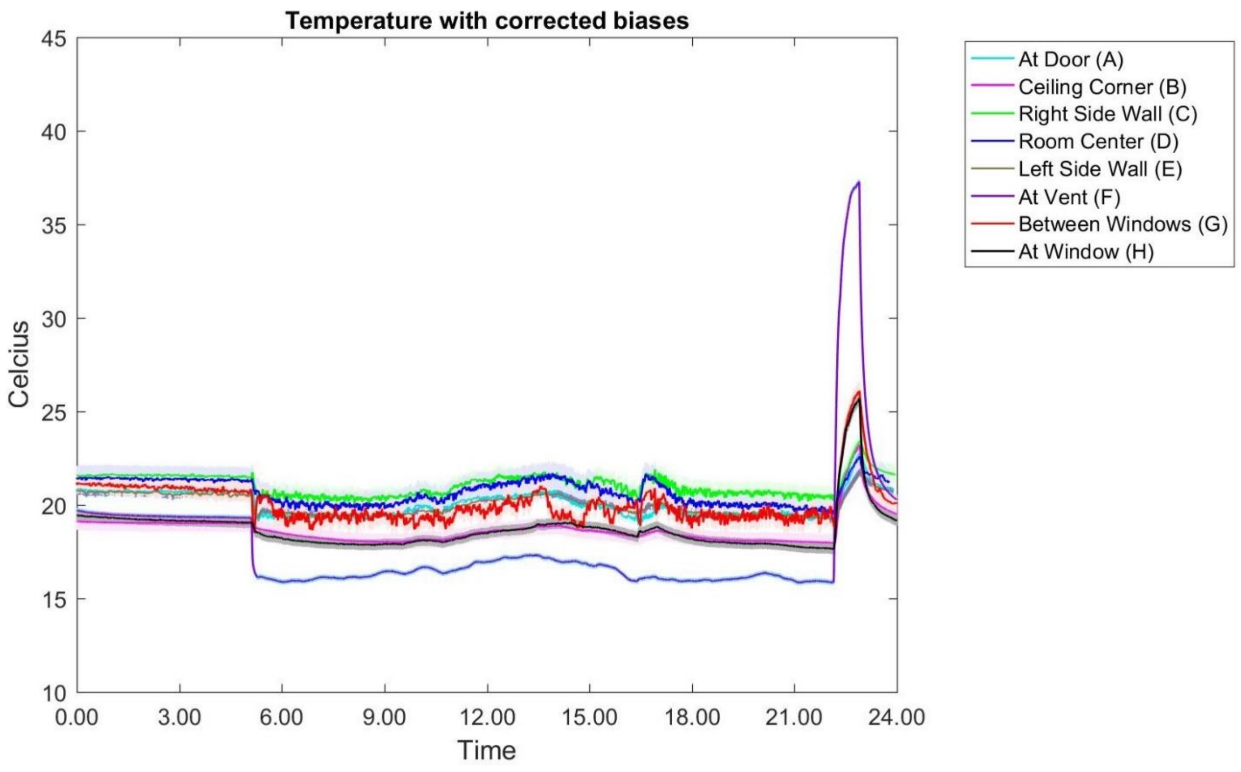


Figure 5.13. Temperature plots from data measured on February 2nd, 2017 with error bars derived from a controlled temperature experiment. The overlapping error bars show the data as same values with offset due to inaccuracy of sensors

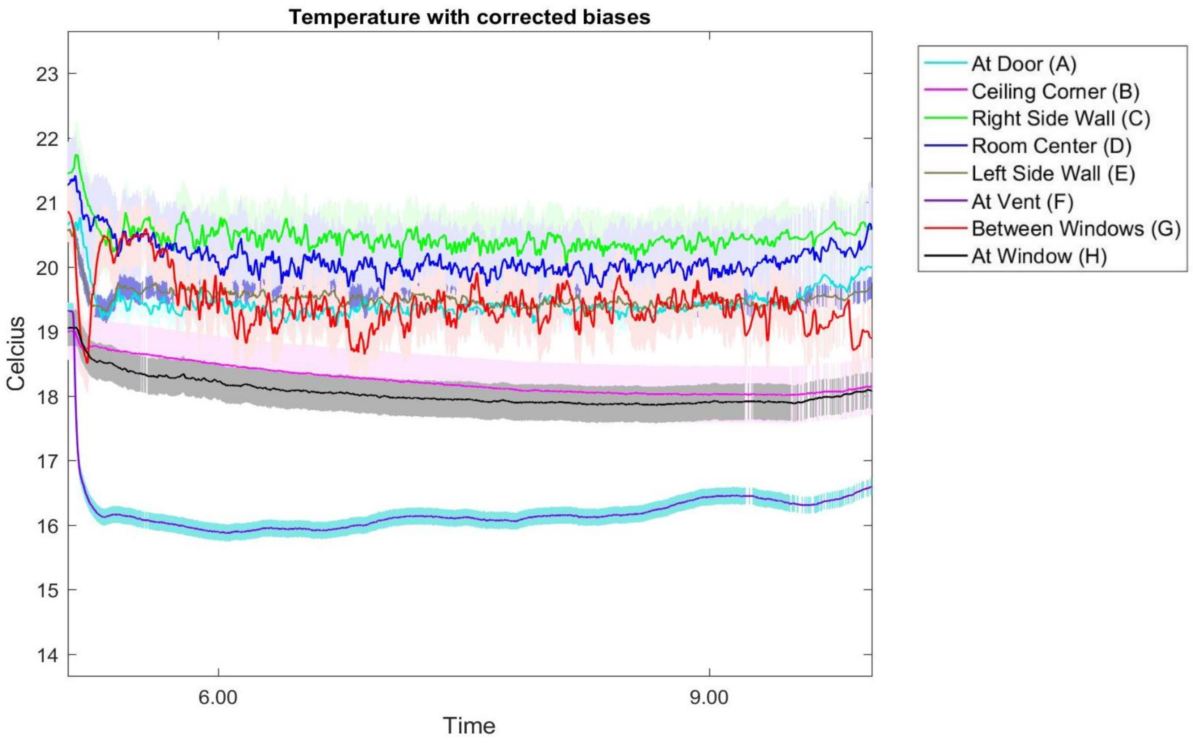


Figure 5.14. Temperature plots from data measured on February 2nd, 2017 with error bars spanning the period of 4 hours from 5:30 to 9:30 am

CHAPTER VI

FUTURE WORKS

Single User Multi Node Interface

Based on the single user multi node interface calibration experiment and data analysis, the currently used sensors, i.e. BME280 can be calibrated via software or hardware to meet the accuracy and precision requirements. This can be achieved if the slope correction technique is established.

Also, high precision sensors can be used as an alternative to meet the accuracy and precision requirements. If separate sensors for measuring temperature, pressure and humidity are used, high accuracy and precision sensors are available. Some work has been performed in selection of sensors:

Table 6.1 Comparison of present and future high precision sensors

Parameter	Sensor	Accuracy	Precision
Temperature, Humidity, Pressure	BME280	±1°C, ± 3%, ± 1 hPa	-
Temperature	MCP9809	+ 0.25°C	+0.0625°C
Pressure	MPL3115A2	± 0.4 kPa	1.5 Pa
Humidity	SHT31-D	± 2%	0.01%

Multi User Single Node Interface

The Multi User Single Node Interface can be expanded by development of multiple nodes that are similar to a single node developed here. These nodes can be at same place or at different places operating tree or star topology with the hub. Different ID’s will need to be provided for each node for their identification. Each node can have different types and numbers of sensors/actuators or electronics attached as per the user requirements.

CHAPTER VII

CONCLUSION

A remote Internet of Things (IoT) laboratory has been developed and used in teaching and research of Cyber Physical Systems (CPS) while successfully demonstrating the concept and technology of the laboratory in two applications. First the laboratory is configured as a Multi User Single Node Interface where multiple remote users can access the unified remote sensing hardware with single point measurements. The example implementation of this system is the UTChattSat that utilizes real-time communication and web-based user control to create a distributed multi-user interface. Second, the laboratory is configured for a Single User Multi Node Interface where a single user can access multiple remote sensing nodes. The example implementation of this system develops the distributed sensor networks that consists of interconnected, real time, and smart system with embedded sensors and processors. This network is used to provide data for assessment of current building energy models.

As a conclusion, experiments can be controlled and monitored across the world using authorized access. Multi-dimensional uses of the laboratory support a wide range of research. Single User Interface provides remote access to industrial and professional research for a personalized experiment. Example: Smart Buildings Challenge at UTC. Multi User Interface provides distributed user measurement/data collection techniques for educational research. Example: Local Science Schools, UTChattSat.

REFERENCES

[1] D. Hodson, "Time for action: Science education for an alternative future," *International Journal of Science Education,* vol. 25, no. 6, 2003.

[2] J. D. Gobert and B. C. Buckley, "Introduction to model-based teaching and learning in science education," *International Journal of Science Education,* vol. 22, no. 9, 2000.

[3] R. Giffinger, "Smart cities - Ranking of European medium-sized cities," Vienna: Centre of Regional Science, 2007.

[4] S. Musa, "Smart City Roadmap," [Online] Available: https://www.academia.edu/21181336/Smart_City_Roadmap

[5] S. Bhardwaj and A. Kole, "Review and Study of Internet of Things: It's the Future," presented at the International Conference on Intelligent Control Power and Instrumentation (ICICPI), 2016.

[6] R. Baheti and H. Gill, "Cyber-physical Systems," The Impact of Control Technology, IEEE Control Systems Society, 2011.

[7] National Science Foundation, "Cyber Physical Systems," Directorate for Computer & Information Science & Engineering Vision Statment, 2015. [Online] Available: https://www.nsf.gov/funding/pgm_summ.jsp?pims_id=503286

[8] N. Komninos, "What makes cities intelligent?," Smart Cities: Governing, Modelling and Analysing the Transition: Taylor & Francis, 2013.

[9] "Building a Smart City + Equitable City," New York City Mayor's Office of Technology & Innovation: Department of Citywide Administrative Services, 2015.

[10] J. Soldatos, "IoT vs. M2M, CPS, WoT...: Are these terms synonyms?," LinkedIn, 2015. [Online] Available: https://www.linkedin.com/pulse/iot-vs-m2m-cps-wot-terms-synonyms-john-soldatos

[11] *Digi XBee Hardware*, 2017. [Online] Available: https://www.digi.com/lp/xbee/hardware

[12] "Product Manual v1.xEx - 802.15.4 Protocol," IEEE 802.15.4 RF Modules, Digi International, 2009.

[13] L. Ada, "Adafruit Ultimate GPS," Adafruit Industries, learn.adafruit.com, 2016.

[14] K. Townsender, "Adafruit 10-DOF IMU Breakout," Adafruit Industries, learn.adafruit.com, 2015.

[15] A. Dennis, *Raspberry Pi Home Automation with Arduino*, First Edition, USA: Packt Publishing, 2013.

[16] P. Justo and E. Gertz, S. Ellis and B. Jepson, *Atmospheric Monitoring with Arduino*, First Edition, USA: O'Reilly Media Inc., 2013.

[17] E. Gertz and P. Justo, S. Ellis and B. Jepson, Eds. *Environmental Monitoring with Arduino,* First Edition, USA: O'Reilly Media Inc., 2012.

[18] R. Faludi, *Building Wireless Sensor Networks*, First Edition, USA: O'Reilly Media Inc., 2011.

[19] L. Ada, "Adafruit BME280 Humidity + Barometric Pressure + Temperature Sensor Breakout," Adafruit Industries, learn.adafruit.com, 2015.

[20] H. Stark and J. Woods, *Probability and Random Processes with Applications to Signal Processing*, Third Edition. USA: Prentice Hall Inc., 2002.

APPENDIX A
SOFTWARE

UTChattSat Main Controller Software:

The software for Main Controller of UTChattSat is as given below. The algorithm is shown in Figure A.1.

```
#include <SoftwareSerial.h>
#include <SD.h>
 SoftwareSerial sensorcont(4,5);
  void setup() {
   Serial.begin(9600);
   sensorcont.begin(9600);
   while(!Serial);
 }
 void loop() {
 int SensorData[10], SensorConfig[3];
 int sensor_SampTime = 0, ExpIter, ExpCounter;
 while(!Serial.available());
 if (Serial.available()){
     for (int i = 0; i<3 ; i++){
       SensorConfig[i] = Serial.parseInt(); // Read Configuration Word form XBee, Digital 0
pin
     }
     for (int i = 0; i<3 ; i++){
       sensorcont.print(SensorConfig[i]);   // Write Configuration Word to SC, Digital 5 pin
       if (i < 2 ) sensorcont.print(",");
       else sensorcont.println(" ");
     }
   sensor_SampTime = SensorConfig[1] * 1500;
   sensorcont.setTimeout(sensor_SampTime);
   ExpIter = trunc(SensorConfig[0]/SensorConfig[1]);
   while(!sensorcont.available());
   for(ExpCounter = 0; ExpCounter<ExpIter; ExpCounter++){
     int j, k;
     for (j = 0; j<10 ; j++){
         SensorData[j] = sensorcont.parseInt();      // Read Data Word form SC, Digital 4 pin
       }
     j=0;
    for (k = 0; k<10 ; k++){
       Serial.print(SensorData[k]);               // Write Data Word to XBee, Digital 1 pin
       if (k < 9 ) Serial.print(",");
       else Serial.println(" ");
     }
     k=0;
   }
```

```
        }
      }
********************************************************************************
```

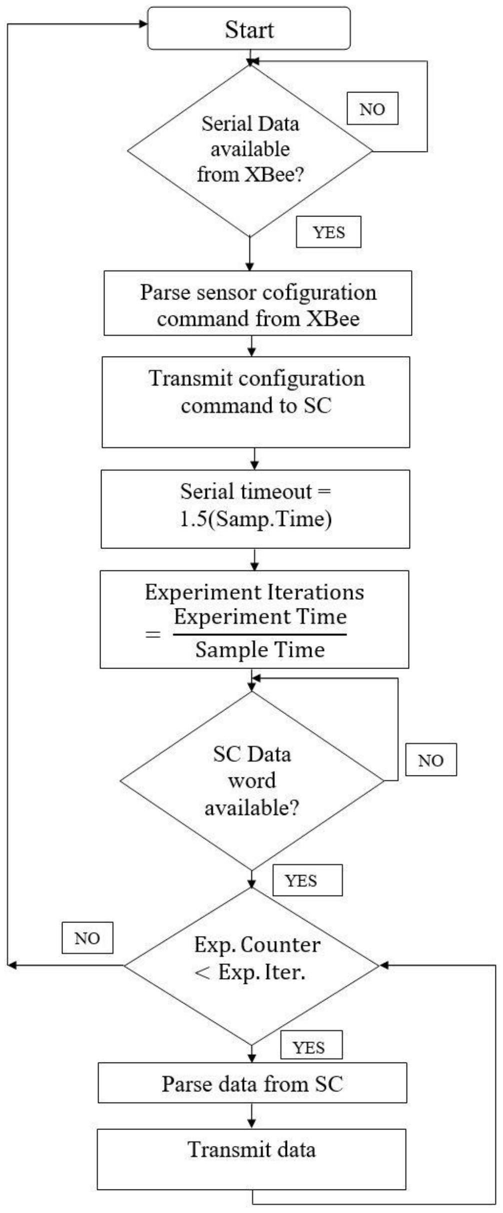

Start
Serial Data available from XBee?
NO
YES
Parse sensor cofiguration command from XBee
Transmit configuration command to SC
Serial timeout = 1.5(Samp.Time)
Experiment Iterations = Experiment Time / Sample Time
SC Data word available?
NO
YES
Exp. Counter < Exp. Iter.
NO
YES
Parse data from SC
Transmit data

Figure A.1 UTChattSat: Main Controller Software Algorithm

UTChattSat Sensor Controller Software:

The software for Sensor Controller of UTChattSat is as given below. The algorithm is shown in Figure A.2 and A.3.

```
#include "MsTimer2.h"
//#include <TimerOne.h>
#include <SoftwareSerial.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM303_U.h>
#include <Adafruit_BMP085_U.h>
#include <Adafruit_L3GD20_U.h>
#include <Adafruit_10DOF.h>
#include <Adafruit_GPS.h>
SoftwareSerial mySerial(3, 2);
Adafruit_GPS GPS(&mySerial);
#define GPSECHO  true
/* Assign a unique ID to the sensors */
Adafruit_10DOF                dof   = Adafruit_10DOF();
Adafruit_LSM303_Accel_Unified accel = Adafruit_LSM303_Accel_Unified(30301);
Adafruit_LSM303_Mag_Unified   mag   = Adafruit_LSM303_Mag_Unified(30302);
Adafruit_BMP085_Unified       bmp   = Adafruit_BMP085_Unified(18001);
sensors_event_t accel_event;
sensors_event_t mag_event;
sensors_vec_t   orientation;
sensors_event_t bmp_event;
float seaLevelPressure = SENSORS_PRESSURE_SEALEVELHPA;
boolean usingInterrupt = false;
float alt, temp1,prs1,roll,pitch,head,alt1;
int ExpTime, SampTime,ExpTime_milli, SampTime_milli, SensorConfig;
uint32_t timer = millis();
int fixtime, latitude, longitude, GPS_alt;
bool gyro_flag=false, GPS_flag=false;
int GetData[10];
// Function declaration
void useInterrupt(boolean);
void initSensors();
void gyro();
int temperature();
int pressure();
int altitude();
```

```
void GPS_parse();
void takeReading();

void setup() {
  Serial.begin(9600);                    // Serial Comm Baud Rate
  GPS.begin(9600);                        // GPS Baud Rate
  GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);   // RMC and GGA
ON
  GPS.sendCommand(PMTK_SET_NMEA_UPDATE_10HZ);     // NMEA Update rate:
10 Hz
  GPS.sendCommand(PMTK_API_SET_FIX_CTL_5HZ);      // Fix Update rate: 5 Hz
  initSensors();                          // Initialise the sensors
  useInterrupt(true);
  delay(900);
  mySerial.println(PMTK_Q_RELEASE);        // Initialize the GPS
}

SIGNAL(TIMER0_COMPA_vect) {
  char c = GPS.read();            // GPS read data
  return;
}

void loop()
{
  int ExpCounter;                //keeps track of how many samples have been collected
  int ExpIter;                  //total number of samples to be taken
  while(!Serial.available());
  if(Serial.available())
  {
    ExpCounter = 0;              // Reset the counters for a new transmission request
    ExpIter = 0;
    ExpTime = Serial.parseInt();  // Save Experiment Time
    SampTime = Serial.parseInt(); // Save Sample Time
    SensorConfig = Serial.parseInt(); // Save Sensor Configuration
    ExpTime_milli = ExpTime * 1000;
    SampTime_milli = SampTime * 1000;
    ExpIter = trunc(ExpTime/SampTime);

  }
  if (ExpCounter < ExpIter){        // Check for the current transmission complete
      for(ExpCounter = 0; ExpCounter<ExpIter; ExpCounter++)       // Run through for
Experiment time
      {
        takeReading();            // collect and transmit readings as per Sensor Configuration
         //char c = Serial.read();
        //while (c != '$');
```

```
                delay(SampTime_milli);          // stay here for Sample time
               }
             }
          }


         void takeReading(){
           bool CompVal = 1;                                // Compare Value = '1' for checking whether
sensor is ON
           int j;
           for (int j = 0; j < 8; j++)   // Roll through all sensors(8); 0 = Gyro, = Alt, 2 = Press, 3 =
Temp, 4 = GPS
           {                                          // 5,6,7 = Extra sensor 1,2,3 respectively
             if (CompVal && bitRead(SensorConfig, j))        // Check sensor j state (1 = ON, 0 =
OFF)
             {
               switch (j)
               {
                case 0 :                                  // Gyroscope ON
                   gyro();                                // Call function to get the Gyroscope data
                   if(gyro_flag){
                     GetData[0] = orientation.roll;
                     GetData[1] = orientation.pitch;
                     GetData[2] = orientation.heading;
                   }
                   else{
                     GetData[0] = 0;
                     GetData[1] = 0;
                     GetData[2] = 0;
                   }
                   gyro_flag = false;
                   break;
                case 1 :
                   GetData[3] = altitude();                 // Altitude ON. Call function to get Altitude
data
                   break;
                case 2 :
                   GetData[4] = pressure();
                   break;
                case 3 :
                   GetData[5] = temperature();            // Temperature ON. Call function to get
Temperature data
                   break;
                case 4 :
                   useInterrupt(true);                   // GPS Interrupt enabled
                   GPS_parse();                          // GPS ON. Call function to get GPS data
```

```
        if(GPS_flag){
          GetData[6] = fixtime;
          GetData[7] = latitude;
          GetData[8] = longitude;
          GetData[9] = GPS_alt;
        }
        else{
          GetData[6] = 0;
          GetData[7] = 0;
          GetData[8] = 0;
          GetData[9] = 0;
        }
        GPS_flag = false;
        useInterrupt(false);              // GPS Interrupt disabled
        break;
      case 5 :                         // Extra Sensor 1 ON
        break;
      case 6 :                         // Extra Sensor 2 ON
        break;
      case 7 :                         // Extra Sensor 3 ON
        break;
      default:                         // Default State. Do Nothing.
        break;
    }
  }
  else                                // Transmit '0' for the sensors that are turned OFF
  {
    switch (j)
    {
      case 0 :                         // Gyroscope OFF
        GetData[0] = 0;
        GetData[1] = 0;
        GetData[2] = 0;
        break;
      case 1 :                         // Altitude OFF
        GetData[3] = 0;
        break;
      case 2 :                         // Pressure OFF
        GetData[4] = 0;
        break;
      case 3 :                         // Temperature OFF
        GetData[5] = 0;
        break;
      case 4 :                         // GPS OFF
        GetData[6] = 0;
        GetData[7] = 0;
```

```
          GetData[8] = 0;
          GetData[9] = 0;
          break;
        case 5 :                        // Extra Sensor 1 OFF
          break;
        case 6 :                        // Extra Sensor 2 OFF
          break;
        case 7 :                        // Extra Sensor 3 OFF
          break;
        default:                        // Default State. Do Nothing.
          break;
       }
     }
    }
  //Serial.print("$");Serial.print(",");

    for (int i=0; i<10 ; i++){

      Serial.print(GetData[i]);                // Print the data stream, each field seperated by
commas
      if (i < 9 ) Serial.print(",");
      else Serial.println(" ");

    }

  //  while(!Serial.available());
   // Serial.println(" ");           // Enter a new line for every transmission
    return;
  }

  void gyro()
  {
    int gyro_data_c[3];

     /* Calculate pitch and roll from the raw accelerometer data */
    accel.getEvent(&accel_event);
    if (dof.accelGetOrientation(&accel_event, &orientation) )
    {                                        // 'orientation' should have valid .roll and .pitch fields
      roll = (orientation.roll/2000)*(65536);          // Step number for 16 bits
     // gyro_data_c[0] = roll;
        gyro_data_c[0] = orientation.roll;

      pitch = (orientation.pitch/2000)*(65536);        // Step number for 16 bits
      //gyro_data_c[1] = pitch;
      gyro_data_c[1] = orientation.pitch;
  }
   /* Calculate the heading using the magnetometer */
  mag.getEvent(&mag_event);
  if (dof.magGetOrientation(SENSOR_AXIS_Z, &mag_event, &orientation) )
  {
     /* 'orientation' should have valid .heading data now */
    head = (orientation.heading/2000)*(65536);       // Step number for 16 bits
    //gyro_data_c[2] = head;
    gyro_data_c[2] = orientation.heading;
  }
  gyro_flag = true;
  return;
}

int temperature(){
  int temp_data_c;
  bmp.getEvent(&bmp_event);
  float temperature;
  if (bmp_event.pressure)
  {
    /* Get ambient temperature in C */
    float temperature;
    bmp.getTemperature(&temperature);
    temp1 = (temperature/90)*256;                    // Step number for 8 bits
    temp_data_c= temp1;
   }
return temp_data_c;
}

int pressure(){
  int prs_data_c;
  bmp.getEvent(&bmp_event);
  float temperature;
   if (bmp_event.pressure)
  {
    /* Get pressure in hPa */
   prs1 = (bmp_event.pressure/1100)*256;             // Step number for 8 bits
   prs_data_c = prs1;

   return prs_data_c;
  }
}

int altitude(){
  int alt_data_c;
  bmp.getEvent(&bmp_event);
```

```
           float temperature;
           if (bmp_event.pressure)   // Calculate the altitude using the barometric pressure sensor */
           {
             float temperature;
             bmp.getTemperature(&temperature);   // Get ambient temperature in C
             alt = bmp.pressureToAltitude(seaLevelPressure,          // Convert atmospheric pressure,
SLP and temp to altitude    */
                                        bmp_event.pressure,
                                        temperature);
             alt1 = (alt/9000)*65536;                      // Step number for 16 bits
             alt_data_c = alt1;

             return alt_data_c;;
           }

         }

         void GPS_parse(){
           float GPS_data_raw[4];

           if (! usingInterrupt) {// read data from the GPS in the 'main loop'
           }
          if (GPS.newNMEAreceived()) {
            if (!GPS.parse(GPS.lastNMEA())) {
           }
             // approximately every 0.2 seconds or so, print out the current stats
           if (millis() - timer > 200) {
            //GPS_data_raw[0]=
(int(GPS.hour)*10000)+(int(GPS.minute)*100)+int(GPS.seconds);
            GPS_data_raw[0] = GPS.hour;
            fixtime = GPS_data_raw[0];
            if (GPS.fix) {
              GPS_data_raw[1]= (GPS.latitudeDegrees/90)*65536;                    // Step number for
16 bits
              latitude = GPS_data_raw[1];
              GPS_data_raw[2]=(GPS.longitudeDegrees/90)*65536;                    // Step number
for 16 bits
              longitude = GPS_data_raw[2];

              GPS_data_raw[3] = GPS.altitude;
              GPS_alt = GPS_data_raw[3];
            }
           }
          }
          GPS_flag =true;
          return ;
```

```
}

void useInterrupt(boolean v) {
  if (v) {
    OCR0A = 0xAF;
    TIMSK0 |= _BV(OCIE0A);
    usingInterrupt = true;
  } else {
    TIMSK0 &= ~_BV(OCIE0A);
    usingInterrupt = false;
  }
  return;
}

void initSensors()
{
  if(!accel.begin())
  {
    /* There was a problem detecting the LSM303 ... check your connections */
    Serial.println(F("Ooops, no LSM303 detected ... Check your wiring!"));
    while(1);
  }
  if(!mag.begin())
  {
    /* There was a problem detecting the LSM303 ... check your connections */
    Serial.println("Ooops, no LSM303 detected ... Check your wiring!");
    while(1);
  }
  if(!bmp.begin())
  {
    /* There was a problem detecting the BMP180 ... check your connections */
    Serial.println("Ooops, no BMP180 detected ... Check your wiring!");
    while(1);
  }
  return;
}
```

****
*****

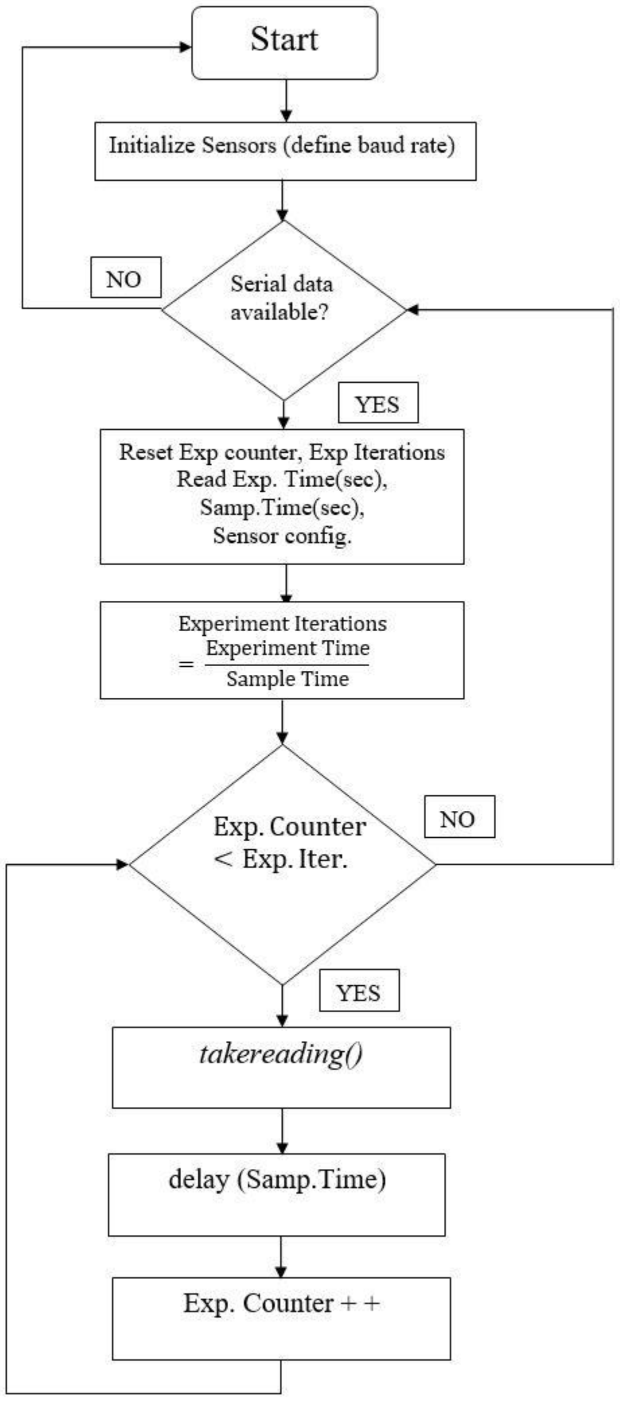

Start
Initialize Sensors (define baud rate)
NO
Serial data available?
YES
Reset Exp counter, Exp Iterations
Read Exp. Time(sec),
Samp.Time(sec),
Sensor config.
Experiment Iterations = Experiment Time / Sample Time
Exp. Counter < Exp. Iter.
NO
YES
takereading()
delay (Samp.Time)
Exp. Counter + +

Figure A.2 UTChattSat: Sensor Controller algorithm

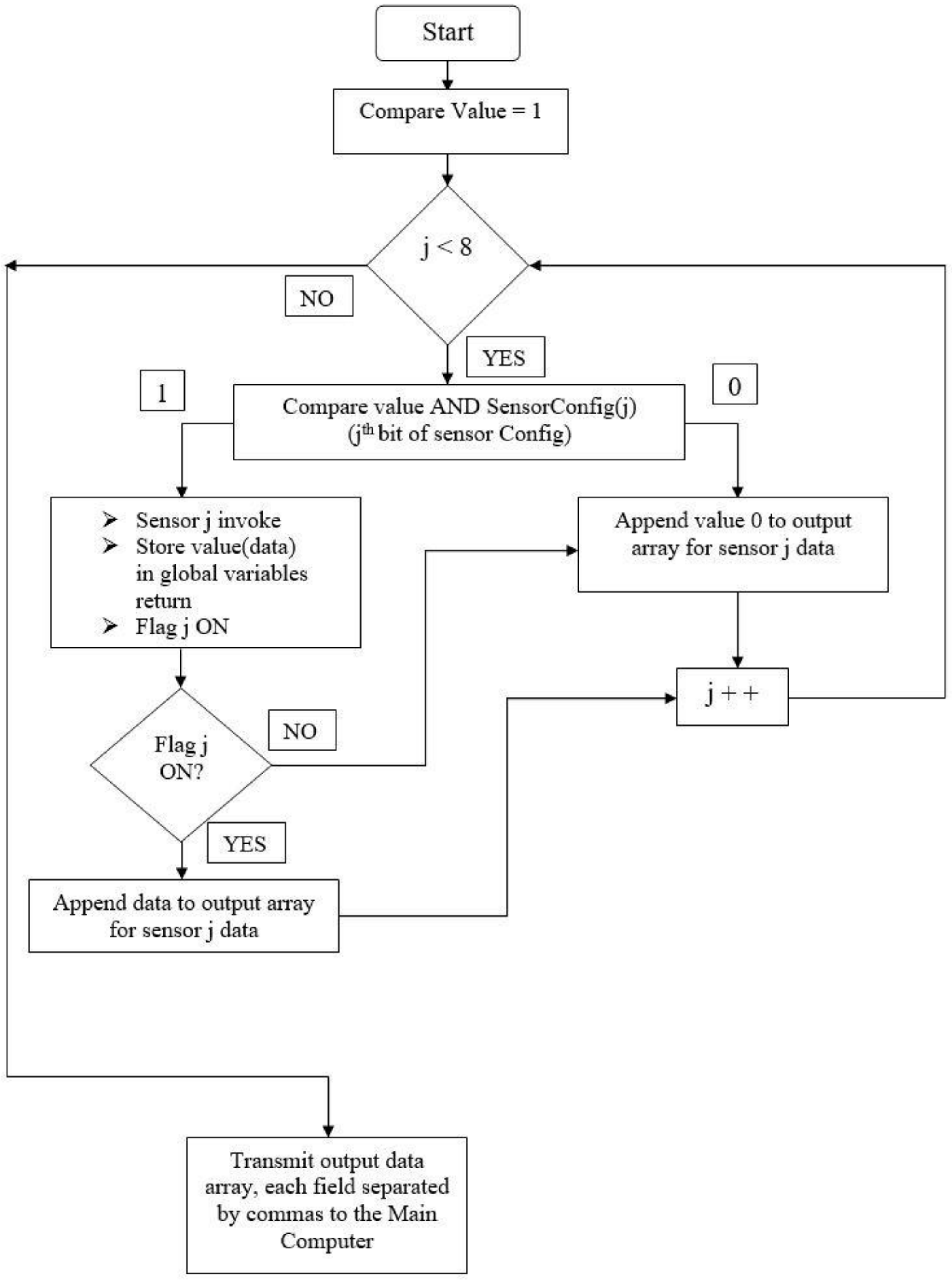

Start
Compare Value = 1
j < 8
NO
YES
1
Compare value AND SensorConfig(j) (jth bit of sensor Config)
0
Sensor j invoke
Store value(data) in global variables return
Flag j ON
Append value 0 to output array for sensor j data
j + +
Flag j ON?
NO
YES
Append data to output array for sensor j data
Transmit output data array, each field separated by commas to the Main Computer

Figure A.3 UTChattSat: Sensor Controller algorithm subroutine *takereading()*

DSN Node Software for Sensor Controller 1 with sensors A and B:

The software for DSN Node Sensor Controller 1 with sensors A and B is as given below.

The algorithm is shown in Figure A.4.

```
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#define BME_SCK 13
#define BME_MISO 12
#define BME_MOSI 11
#define BME_CS_A 10
#define BME_CS_B 9
#define SEALEVELPRESSURE_HPA (1013.25)
Adafruit_BME280 bme_A(BME_CS_A); // hardware SPI
Adafruit_BME280 bme_B(BME_CS_B); // hardware SPI

void setup() {
  Serial.begin(9600);
  bme_A.begin();
  bme_B.begin();
  if (!bme_A.begin()) {
    Serial.println("Could not find a valid BME280 sensor, check wiring!");
    while (1);
  }
  if (!bme_B.begin()) {
    Serial.println("Could not find a valid BME280 sensor, check wiring!");
    while (1);
  }
}

void loop() {
  int code = 0;
  float  CV_A = -4.5, CV_B = -2.7;
  while(!Serial.available());
    if (Serial.available()){
    code = Serial.parseInt();
    switch(code)
    {
      case 101:
      // Start parsing sensor A
        Serial.print("A,");
        Serial.print((bme_A.readTemperature())+CV_A);
        Serial.print(",");
        Serial.print((bme_A.readPressure() / 100.0F));
        Serial.print(",");
        Serial.print((bme_A.readAltitude(SEALEVELPRESSURE_HPA)));
        Serial.print(",");
        Serial.println((bme_A.readHumidity()));
        break;
      case 102:
        // Start parsing sensor B
        Serial.print("B,");
        Serial.print((bme_B.readTemperature())+CV_B);
        Serial.print(",");
        Serial.print((bme_B.readPressure() / 100.0F));
        Serial.print(",");
        Serial.print((bme_B.readAltitude(SEALEVELPRESSURE_HPA)));
        Serial.print(",");
        Serial.println((bme_B.readHumidity()));
        break;
    }
  }
```
****

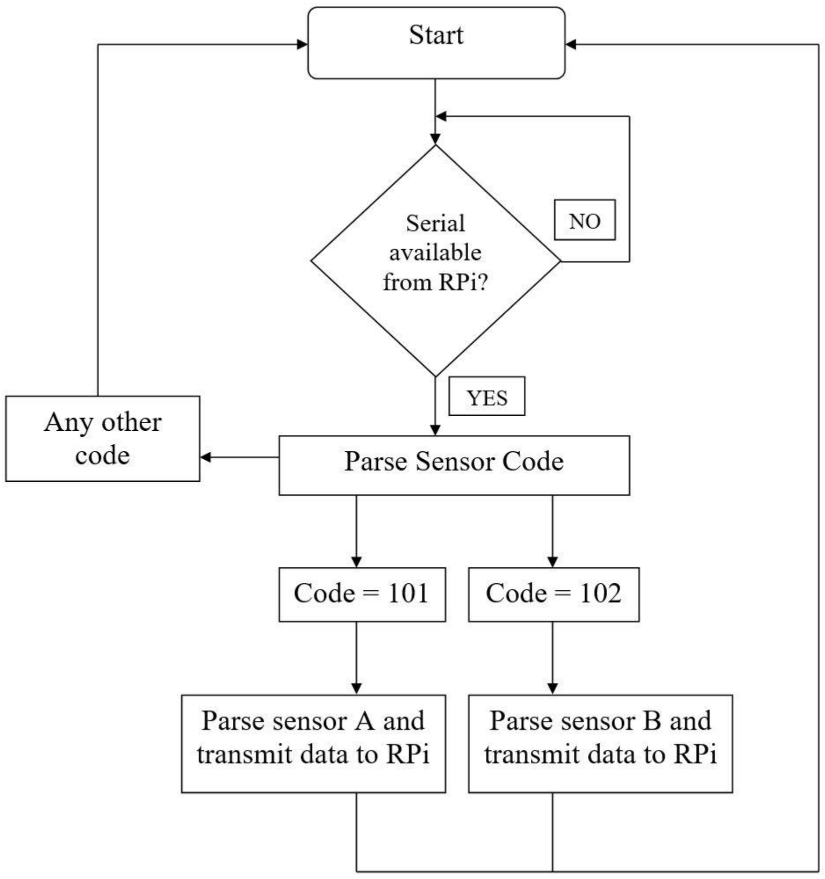

Start
Serial available from RPi?
NO
YES
Any other code
Parse Sensor Code
Code = 101
Code = 102
Parse sensor A and transmit data to RPi
Parse sensor B and transmit data to RPi

Figure A.4 DSN: Node Software Development Algorithm for Sensor Controller 1 with Sensor A and B

DSN Hub Software:

The software for DSN Hub is as given below. The algorithm is shown in Figure A.5, A.6, A.7 and A.8.

```
##  libraries       ##
import os.path, serial, time, datetime, socket, sys, threading, linecache

time_start = time.time()
time_start_string = datetime.datetime.fromtimestamp(time_start).strftime('-
%Y%m%d%H%M%S')
```

```
ser = serial.Serial('/dev/ttyUSB0', 9600)
ser.timeout = 1
    ##files
Dir_SDL = '/home/pi/Desktop/Sync/SensorLog'
SDL_name_A = '/Sens_A_' + time_start_string + '.txt'
SDL_name_B = '/Sens_B_' + time_start_string + '.txt'
SDL_name_C = '/Sens_C_' + time_start_string + '.txt'
SDL_name_D = '/Sens_D_' + time_start_string + '.txt'
SDL_name_E = '/Sens_E_' + time_start_string + '.txt'
SDL_name_F = '/Sens_F_' + time_start_string + '.txt'
SDL_name_G = '/Sens_G_' + time_start_string + '.txt'
SDL_name_H = '/Sens_H_' + time_start_string + '.txt'
Err_log = '/Err_' + time_start_string + '.txt'

##  functions       ##

def Startup():
    time_start = time.time()
    time_start_string = datetime.datetime.fromtimestamp(time_start).strftime('-
%Y%m%d%H%M%S')
    print 'Running Startup() at' + time_start_string
    if not os.path.exists(Dir_SDL):
        os.makedirs(Dir_SDL)
    log = open(Dir_SDL + SDL_name_A, 'w')
    log.write('Sensor A Data log \n\nStarted: ' + time_start_string + '\n Time \t\t Temp \t Press \t
Alt \t Hum\n')
    log.close()

    log = open(Dir_SDL + SDL_name_B, 'w')
    log.write('Sensor B Data log \n\nStarted: ' + time_start_string + '\n Time \t\t Temp \t Press \t
Alt \t Hum\n')
    log.close()

    log = open(Dir_SDL + SDL_name_C, 'w')
    log.write('Sensor C Data log \n\nStarted: ' + time_start_string + '\n Time \t\t Temp \t Press \t
Alt \t Hum\n')
    log.close()

    log = open(Dir_SDL + SDL_name_D, 'w')
    log.write('Sensor D Data log \n\nStarted: ' + time_start_string + '\n Time \t\t Temp \t Press \t
Alt \t Hum\n')
    log.close()

    log = open(Dir_SDL + SDL_name_E, 'w')
    log.write('Sensor E Data log \n\nStarted: ' + time_start_string + '\n Time \t\t Temp \t Press \t
Alt \t Hum\n')
```

```
    log.close()

    log = open(Dir_SDL + SDL_name_F, 'w')
    log.write('Sensor F Data log \n\nStarted: ' + time_start_string + '\n Time \t\t Temp \t Press \t
Alt \t Hum\n')
    log.close()

    log = open(Dir_SDL + SDL_name_G, 'w')
    log.write('Sensor G Data log \n\nStarted: ' + time_start_string + '\n Time \t\t Temp \t Press \t
Alt \t Hum\n')
    log.close()

    log = open(Dir_SDL + SDL_name_H, 'w')
    log.write('Sensor H Data log \n\nStarted: ' + time_start_string + '\n Time \t\t Temp \t Press \t
Alt \t Hum\n')
    log.close()

def RPi2_SensorLog():
    i = 0
    err = 0
    while True:
        code = 100
        j = 0
        for j in range(1,9):
            code = 100 + j;
            ser.write(str(code))
            i = i+1
            time.sleep(0.25)
            print "Sample:" + str(i) + " : " + str(code) + "\n"
            RXSamp = ser.readline()
            print RXSamp
            sze_RXSamp = sys.getsizeof(RXSamp)
            print sze_RXSamp
            if sze_RXSamp != 50:
                print RXSamp
                err = err+1
                RXSamp = "0,0,0,0,0"
                print err
                ser.flushInput()

            SampDataList = RXSamp.split(',')
            SplitVar = SampDataList[4].split()
            SampDataList[4] = SplitVar[0]
            time_sample = time.time()
            time_sample_string =
datetime.datetime.fromtimestamp(time.time()).strftime('%H:%M:%S')
```

```
compare_time = "23:59:59"
SensorID = SampDataList[0]

print SampDataList

if SensorID == 'A':
    Sen_A_data = SampDataList
    with open(Dir_SDL + SDL_name_A, 'a') as log:
        log.write(time_sample_string + '\t')
        log.write(Sen_A_data[1] + '\t')
        log.write(Sen_A_data[2] + '\t')
        log.write(Sen_A_data[3] + '\t')
        log.write(Sen_A_data[4] + '\n')
    log.close()

elif SensorID == 'B':
    Sen_B_data = SampDataList
    with open(Dir_SDL + SDL_name_B, 'a') as log:
        log.write(time_sample_string + '\t')
        log.write(Sen_B_data[1] + '\t')
        log.write(Sen_B_data[2] + '\t')
        log.write(Sen_B_data[3] + '\t')
        log.write(Sen_B_data[4] + '\n')
    log.close()

elif SensorID == 'C':
    Sen_C_data = SampDataList
    with open(Dir_SDL + SDL_name_C, 'a') as log:
        log.write(time_sample_string + '\t')
        log.write(Sen_C_data[1] + '\t')
        log.write(Sen_C_data[2] + '\t')
        log.write(Sen_C_data[3] + '\t')
        log.write(Sen_C_data[4] + '\n')
    log.close()

elif SensorID == 'D':
    Sen_D_data = SampDataList
    with open(Dir_SDL + SDL_name_D, 'a') as log:
        log.write(time_sample_string + '\t')
        log.write(Sen_D_data[1] + '\t')
        log.write(Sen_D_data[2] + '\t')
        log.write(Sen_D_data[3] + '\t')
        log.write(Sen_D_data[4] + '\n')
    log.close()

elif SensorID == 'E':
    Sen_E_data = SampDataList
    with open(Dir_SDL + SDL_name_E, 'a') as log:
        log.write(time_sample_string + '\t')
        log.write(Sen_E_data[1] + '\t')
        log.write(Sen_E_data[2] + '\t')
        log.write(Sen_E_data[3] + '\t')
        log.write(Sen_E_data[4] + '\n')
    log.close()

elif SensorID == 'F':
    Sen_F_data = SampDataList
    with open(Dir_SDL + SDL_name_F, 'a') as log:
        log.write(time_sample_string + '\t')
        log.write(Sen_F_data[1] + '\t')
        log.write(Sen_F_data[2] + '\t')
        log.write(Sen_F_data[3] + '\t')
        log.write(Sen_F_data[4] + '\n')
    log.close()

elif SensorID == 'G':
    Sen_G_data = SampDataList
    with open(Dir_SDL + SDL_name_G, 'a') as log:
        log.write(time_sample_string + '\t')
        log.write(Sen_G_data[1] + '\t')
        log.write(Sen_G_data[2] + '\t')
        log.write(Sen_G_data[3] + '\t')
        log.write(Sen_G_data[4] + '\n')
    log.close()

elif SensorID == 'H':
    Sen_H_data = SampDataList
    with open(Dir_SDL + SDL_name_H, 'a') as log:
        log.write(time_sample_string + '\t')
        log.write(Sen_H_data[1] + '\t')
        log.write(Sen_H_data[2] + '\t')
        log.write(Sen_H_data[3] + '\t')
        log.write(Sen_H_data[4] + '\n')
    log.close()

if time_sample_string == compare_time:
    with open(Dir_SDL + Err_log, 'a') as log:
        log.write("Total numbers of errors: " + str(err))
    log.close()
    return
time.sleep(0.5)
```

```
##  main       ##
Startup()

time_start = time.time()
time_start_string = datetime.datetime.fromtimestamp(time_start).strftime('-
%Y%m%d%H%M%S')

while True:
   time_start = time.time()
   time_start_string = datetime.datetime.fromtimestamp(time_start).strftime('-
%Y%m%d%H%M%S')
   SDL_name_A = '/Sens_A_' + time_start_string + '.txt'
   SDL_name_B = '/Sens_B_' + time_start_string + '.txt'
   SDL_name_C = '/Sens_C_' + time_start_string + '.txt'
   SDL_name_D = '/Sens_D_' + time_start_string + '.txt'
   SDL_name_E = '/Sens_E_' + time_start_string + '.txt'
   SDL_name_F = '/Sens_F_' + time_start_string + '.txt'
   SDL_name_G = '/Sens_G_' + time_start_string + '.txt'
   SDL_name_H = '/Sens_H_' + time_start_string + '.txt'
   Err_log = '/Err_' + time_start_string + '.txt'

   print 'Running Polling() at' + time_start_string
   log = open(Dir_SDL + SDL_name_A, 'w')
   log.write('Sensor A Data log \n\nStarted: ' + time_start_string + '\n Time \t\t Temp \t Press \t
Alt \t Hum\n')
   log.close()

   log = open(Dir_SDL + SDL_name_B, 'w')
   log.write('Sensor B Data log \n\nstarted: ' + time_start_string + '\n Time \t\t Temp \t Press \t
Alt \t Hum\n')
   log.close()

   log = open(Dir_SDL + SDL_name_C, 'w')
   log.write('Sensor C Data log \n\nStarted: ' + time_start_string + '\n Time \t\t Temp \t Press \t
Alt \t Hum\n')
   log.close()

   log = open(Dir_SDL + SDL_name_D, 'w')
   log.write('Sensor D Data log \n\nStarted: ' + time_start_string + '\n Time \t\t Temp \t Press \t
Alt \t Hum\n')
   log.close()

   log = open(Dir_SDL + SDL_name_E, 'w')
   log.write('Sensor E Data log \n\nStarted: ' + time_start_string + '\n Time \t\t Temp \t Press \t
Alt \t Hum\n')
```

```
    log.close()

    log = open(Dir_SDL + SDL_name_F, 'w')
    log.write('Sensor F Data log \n\nStarted: ' + time_start_string + '\n Time \t\t Temp \t Press \t
Alt \t Hum\n')
    log.close()

    log = open(Dir_SDL + SDL_name_G, 'w')
    log.write('Sensor G Data log \n\nStarted: ' + time_start_string + '\n Time \t\t Temp \t Press \t
Alt \t Hum\n')
    log.close()

    log = open(Dir_SDL + SDL_name_H, 'w')
    log.write('Sensor H Data log \n\nStarted: ' + time_start_string + '\n Time \t\t Temp \t Press \t
Alt \t Hum\n')
    log.close()

    RPi2_SensorLog()
    time.sleep(3)

********************************************************************************
*****
```

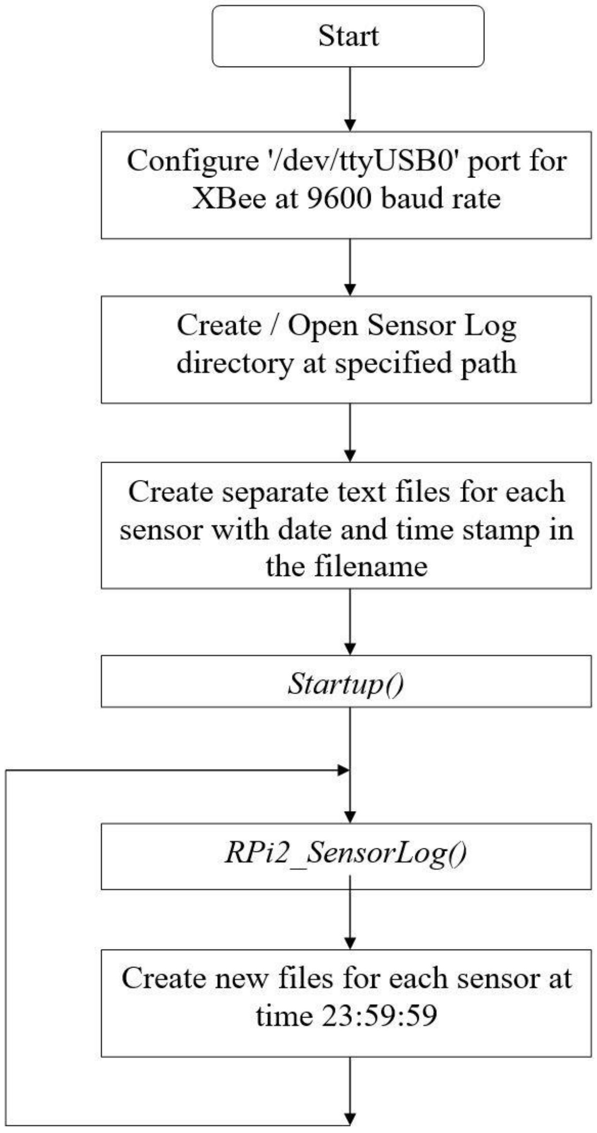

Start
Configure '/dev/ttyUSB0' port for XBee at 9600 baud rate
Create / Open Sensor Log directory at specified path
Create separate text files for each sensor with date and time stamp in the filename
Startup()
RPi2_SensorLog()
Create new files for each sensor at time 23:59:59

Figure A.5 DSN: Hub Software Algorithm main flow

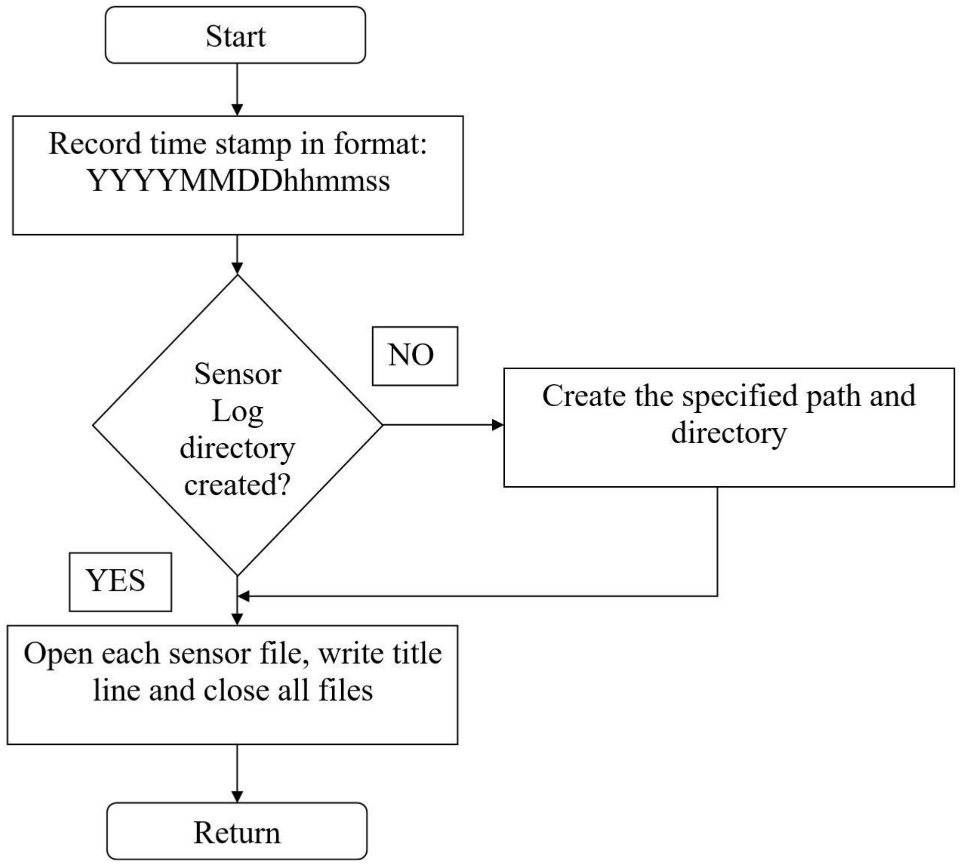

Start
Record time stamp in format: YYYYMMDDhhmmss
Sensor Log directory created?
NO
Create the specified path and directory
YES
Open each sensor file, write title line and close all files
Return

Figure A.6 DSN: *Startup()* subroutine

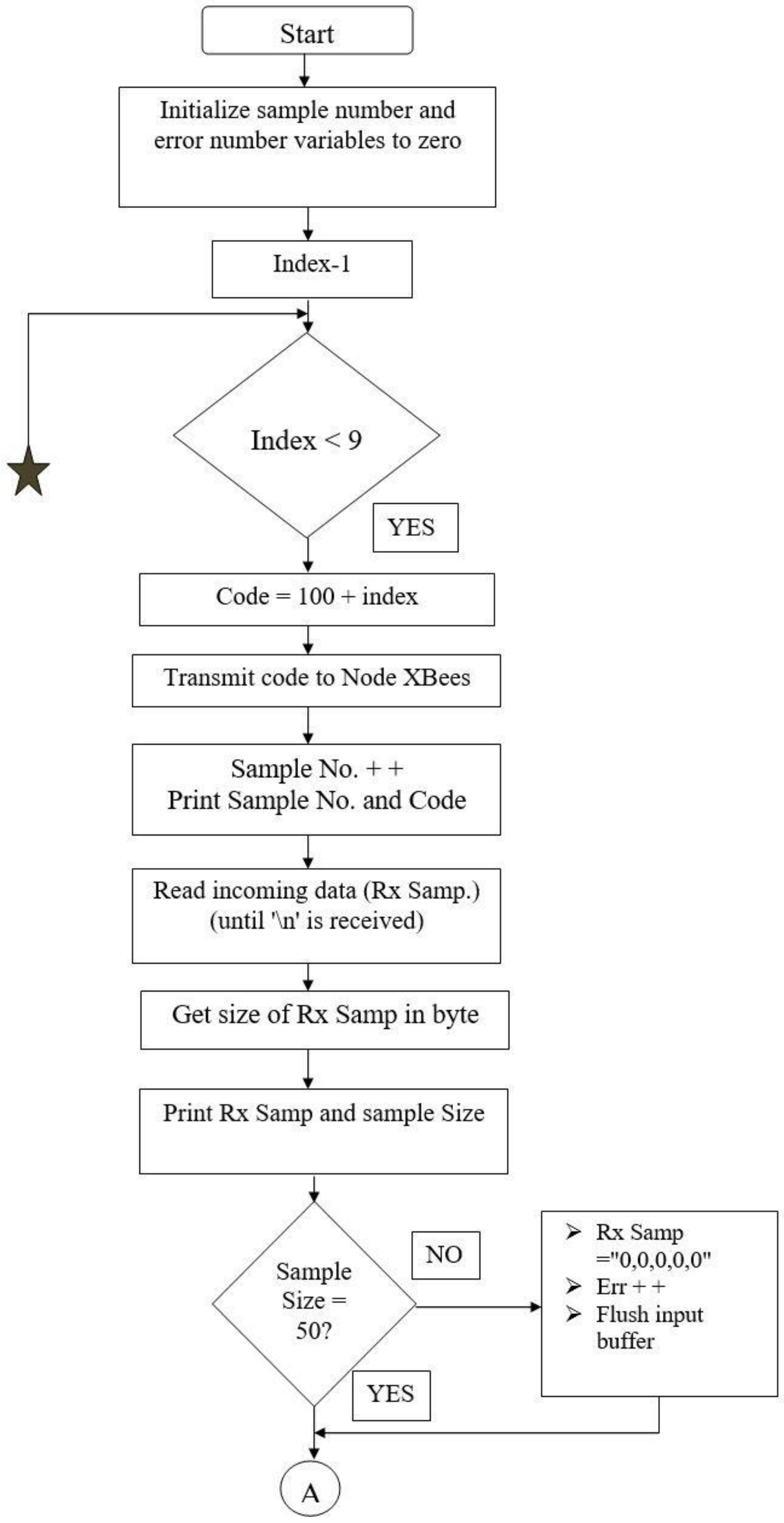

Start
Initialize sample number and error number variables to zero
Index-1
Index < 9
YES
Code = 100 + index
Transmit code to Node XBees
Sample No. + +
Print Sample No. and Code
Read incoming data (Rx Samp.) (until '\n' is received)
Get size of Rx Samp in byte
Print Rx Samp and sample Size
Sample Size = 50?
NO
Rx Samp ="0,0,0,0,0"
Err + +
Flush input buffer
YES
A

Figure A.7 DSN: *RPi2_SensorLog()* subroutine part-1

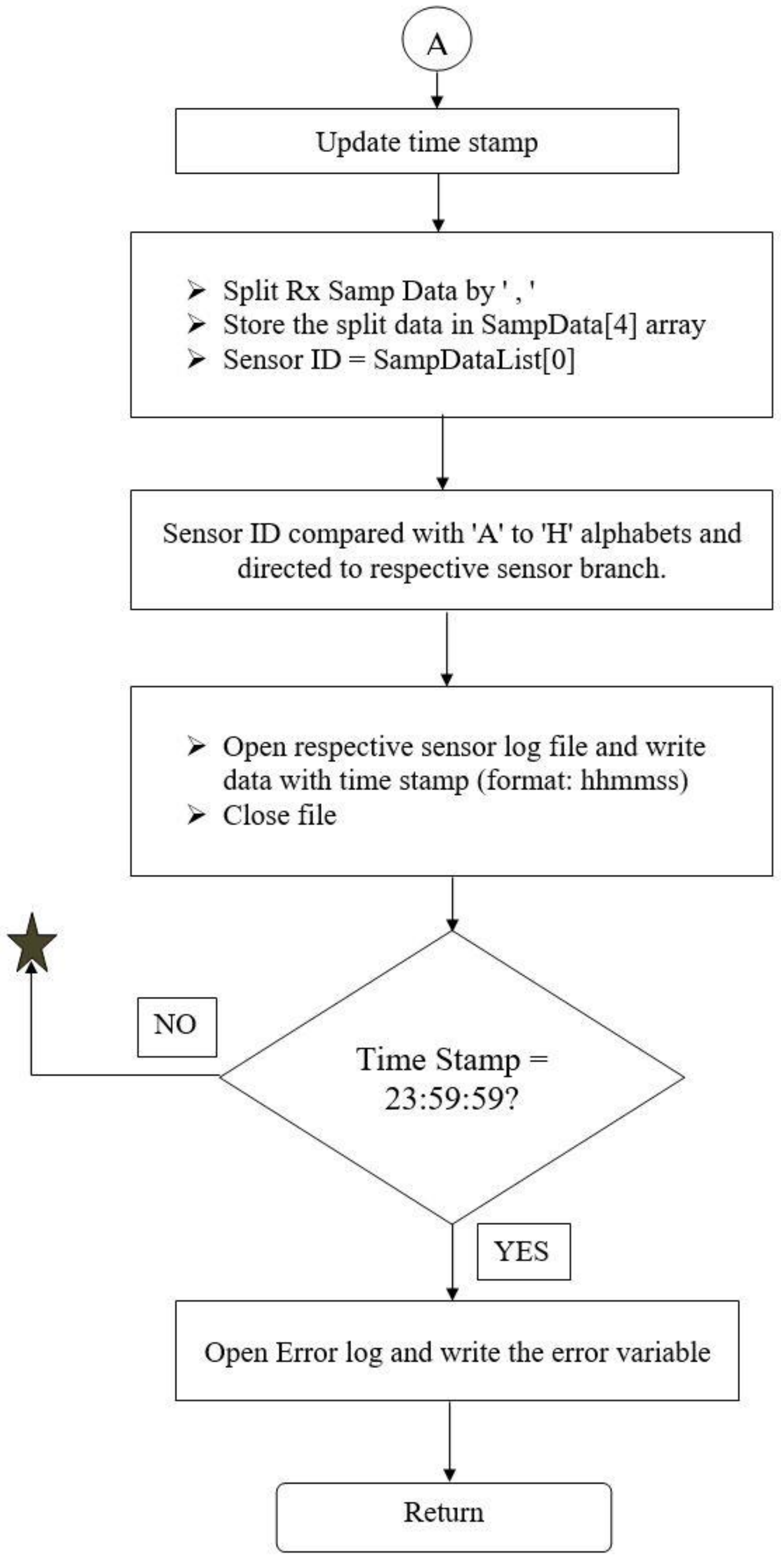

A
Update time stamp
➢ Split Rx Samp Data by ' , '
➢ Store the split data in SampData[4] array
➢ Sensor ID = SampDataList[0]
Sensor ID compared with 'A' to 'H' alphabets and directed to respective sensor branch.
➢ Open respective sensor log file and write data with time stamp (format: hhmmss)
➢ Close file
NO
Time Stamp = 23:59:59?
YES
Open Error log and write the error variable
Return

Figure A.8 DSN: *RPi2_SensorLog()* subroutine part-2

VITA

Amee Patel was born in Gujarat, India, to the parents Sharadaben and Maheshkumar Patel. She is the first of the two children, a younger brother. She attended L.J. Primary School and continued to Devashish Sankul in Ahmedabad, Gujarat. After the 10th grade, she pursued Diploma of Engineering in electronics and Communication from Government Girls Polytechnic in Ahmedabad, Gujarat and graduated in June 2010. She completed her Bachelors of Engineering in June 2014 in Electronics and Communication from Gujarat Technological University, Ahmedabad. Meanwhile, she worked for one year and five months for the Institute of Plasma Research as a Technical Assistant and then continued pursuing her Bachelor's Degree. Amee got admitted in University of Tennessee at Chattanooga in August 2015 and accepted a graduate research assistantship in January 2016. She graduated with a Masters of Science degree in Electrical Engineering in August 2017. Amee is working with La-Z-Boy Residential as Electrical Engineer in the Research and development department.