Division of Engineering Programs

EMBEDDED LINUX PROJECT REPORT

Spring 2017

May 12th, 2017

Scalable and Robust Infrastructure to Create Building's Heat and Humidity Profile

Student name	Student Major	Grade Received
James Earl	CE	
Richard Paras	CS	

ABSTRACT

Using sensors and microcontroller technology, the team developed programs to utilize a sensor(s) to monitor and track temperature and humidity data. The data is then logged on the Raspberry Pi's local database to be accessed later by a remote web server in order for someone to interface with the stored data whether they want to view historical or current data.

TABLE OF CONTENTS

1. ABOUT THIS PROJECT

Temperature and humidity data can be regarded with high importance in sensitive environments. This project delves into a way to track such data so that decisions can be made on how to maintain such an environment, or collect data purely for research. With our design, you could interface multiple microcontrollers where one operates as a master, and the others operate as slaves. Their functionalities are as follows.

Master	Slave
Run Flask Web Server	**Track sensor**
Request data from slave(s) database	**Log data to database**
Present data in a viable, usable format (graph)	**Facilitate database requests from master**

Figure 1.1 - Device Roles

For our proposed design, it is imperative that there is one and only one master, and at least one slave. For our prototype, we will carry out the project with just one slave, and one master.

2. FUNCTIONAL DESCRIPTION OF DESIGN

2.1 Slave Design -

The slave's functionality is most important to the operation of the project. It polls a sensor and stores the data that is used. For the slave, we need a means of reading the data from the sensor, logging the data, and facilitating a file share so that it's database may be accessed externally (by the master). In order to complete step one, we must select a sensor. For our purposes, we will use the DHT22 temperature and humidity sensor.

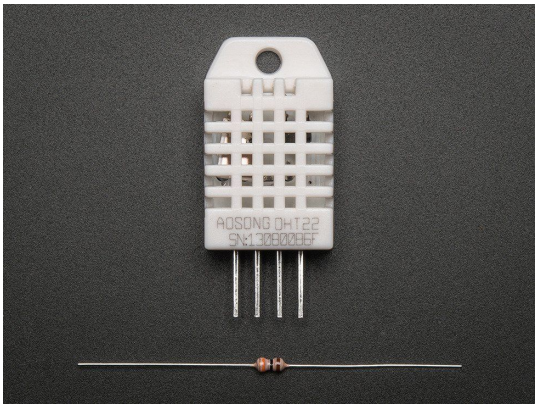

Figure 2.1.1 - DHT22 Temperature and Humidity Sensor

This sensor was very pleasurable to work with because unlike the DS18B20 temperature sensors we worked with earlier, this had a very low latency, and its temperature readings were very much on par like that of the DS18B20 sensors while being less susceptible to influence from anything touching the sensor. To use this sensor, we used the Adafruit library to interface with the sensor using Python.

```
1. git clone https://github.com/adafruit/Adafruit_Python_DHT.git
2. cd Adafruit_Python_DHT
3. sudo apt-get install build-essential python-dev python-openssl
4. sudo python setup.py install
```

Figure 2.1.2 - DHT22 Temperature and Humidity Sensor Library Install

Figure 2.1.3 - DHT22 Temperature and Humidity Sensor Wiring

After setting up the libraries, James created a python file, getData.py that had an embedded function, getData to be called upon that returned humidity data as a percentage, temperature as celsius and fahrenheit, and the current time and date.

Once this was in place, James and Richard both setup a SQL database using SQLite3 to log data. The table in envData.db appears as the following.

line	date	time	tempC	tempF	humperc
1	4/10/2017	9:00	20	68	35

Figure 2.1.4 - SQLite3 Database Table Template

Data is to be logged every half hour for our purposes. After this, the only thing left to do on the slave was make its database mountable elsewhere. To do this, James used a file sharing client called Samba whose operation is similar to that of file sharing on a Windows platform. By

doing this, we were able to make the directory where the database mountable to other devices.

Setting up Samba is as follows...

sudo apt install samba

sudo smbpasswd -a <user_name> //<user_name> is to be substituted for a wanted username.

//You will then be asked to secure the user_name with a password

From here, we must modify the smb.conf file. This is performed as follows...

sudo nano /etc/samba/smb.conf

We then append the share data to the End of the file as follows.

```
[<folder_name>]                              //folder_name represents intended share name
path = /home/<user_name>/<folder_name>       //path where local database is stored
valid users = <user_name>                    //File share user you just created goes here
read only = no                               //Makes directory writeable remotely
```

2.2 Master Design -

The first part of the design that had to be adhered to was mounting the remote database. This is performed as follows…

<put mount command here>

After this, we can now access the database that is actively being logged to to access current and historical data. So from here, Richard wrote a Python functions to read from the database and retrieve the information we need. getDBdate.py retrieves the miscellaneous data that will be used by the web server to make this data useable to a admin user from their web browser.

James configured and programmed the web server/html.Three main pages were set up. A

main “landing” page, a history request, and a history displayed page.

<SCREENSHOTS HERE>

The landing page displayed the “current” information. It displayed the current celsius and humidity percentage along with the current date and time.

On our history request page, we give the user the ability to request up to one day worth of data to be displayed in units of their choice.

From here, the date requested was taken and the data was gathered and assembled into a pygal graph to give a user an accurate visual representation of up to 48 data points that could be gathered over the course of one day.

2.3 - Problems Faced -

The most difficult and time consuming aspect of the project was to use the information and display it using a flask web server. One of the major issues being that the transfer from SQL database to a python variable tuple instead of an integer. Most notable the data began with a (u’). This was resolved by using the built in method row_factory inside the connection class. Another large issue was the group’s unfamiliarity with HTML. If our third partner had come to do the project, tasks could have been better delegated and therefore produced a better product overall.

3. CONCLUSIONS

Going forward, the thing this project could benefit from the most is being more user friendly as far as the UI is concerned. The web page could be all around more elegant and

make better use of graphics.

As far as history display is concerned, I would make the history request a range of dates that averages out the data across 48 points depending on how many days are requested to give the most accurate visual representation for the range of the user's choice. I think it would also be nice for the database to be relocated to the “master” pi and log data from all slaves to one Pi. This would make it easier to develop as far a a final product is concerned because there would only be one database to interface with versus one database per slave.

However, overall, I think we were able to reach a good final product and gave us a good idea and look into how web development and embedded systems can come together.

5. APPENDIX

<Code Attached>