Formal Project
Documentation #1

Embedded Linux

Group Members/
Erin Rivera, Mahmoud Jawharji, Ivan D'Alcantara, Fabricio Goncalves, Kevin Han, and Theodore Thueson

Professor/
Chirakkal Easwaran

May 15 2015

Project Description

The project scope is to create a buoy that will hold the raspberry pi, along with other materials necessary to be able to record temperatures at different depths using temperature sensors. The buoy will hold the pi and be connected to multiple temperature sensors at different depths. It will also be connected to a motor that will control our pulley system. Our pulley system is being created as a way for us to elevate and descend our temperature sensors into the water. For example having the first sensor reading the temperature 1 foot below sea level, and the second reading it from 2 feet below sea level and so on. After the pulley lowers the sensor to its location it will remain there for however long is needed, record the data, then be elevated back into its original position. It will be recorded at fixed time intervals of about 1 hour or so. After the data is recorded it will be send the information to the web server. The web server will be able to map out the data on a graph and display it.

Project Goals

The goal of our project is to be able to create a buoy that will record temperature of water. Another one of our goals is to create a design in our buoy with our raspberry pi, temperature sensors, motor, and pulley system where the design will have a fully working waterproof system and be able to monitor and record temperature at multiple depths. The final goal includes being able to have our raspberry pi record the data, save the data, then send the to a web server. Along with creating the web server, we will also need to have the web server receive the data and graph it in a user-friendly form where anyone will be able to monitor our data.

Hardware Components

The hardware for this project is divided into two sections, Buoy Design Parts and Electronic Parts.

A. Buoy Design Parts

The buoy has been built using a plastic case (yellow), which contains a box inside it. All the spaces that are between the box and plastic case are covered with three layers of two types of heavy-duty tapes (black and brown), which hold the whole structure as one solid piece, see Figure 1a & 1b.

Figure 1a

Figure 1b

B. Electronic Parts

The next step is to get all the electronics parts, (Breadboard, Raspberry pi B+, DC & Stepper Motor Hat for Raspberry pi, Stepper Motor, waterproof DS18B20 Digital temperature sensor, and jumper wires), and connect them together, see Figure 2. Note, two holes have been made, one on the side to insert the temperature sensor wire, and the other one in the middle in order to connect the motor to a “Pulley” from the other side. This will help to manage and control the depth of the temperature sensor, in which it will pull it either upward or downward, see Figure 3a & 3b.

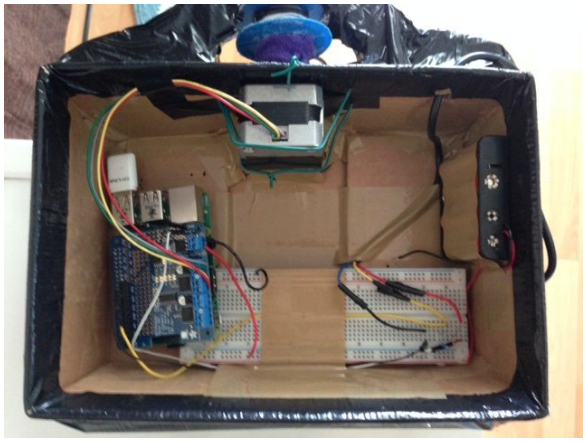

Figure 2

Figure 3a

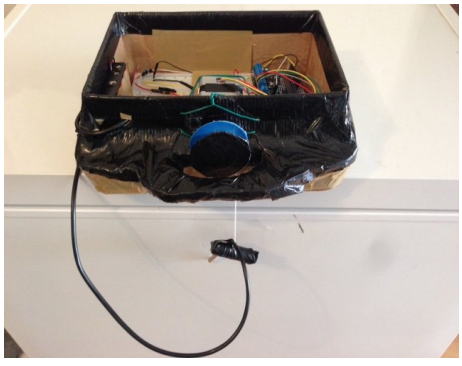

Figure 3b

Software

The software for this project is divided into four parts, Temperature Sensor Code, Stepper Motor Code, Server Communication Code, and Website Code.

A. Temperature Sensor Code

In order to connect the DS18B20 temperature sensor, we followed the same steps as in Figure 4, which from the tutorial on adafriut.com. However, we did not use the extendable wire for the GPIO since we are fully using the GPIO for the Stepper Motor Hat.

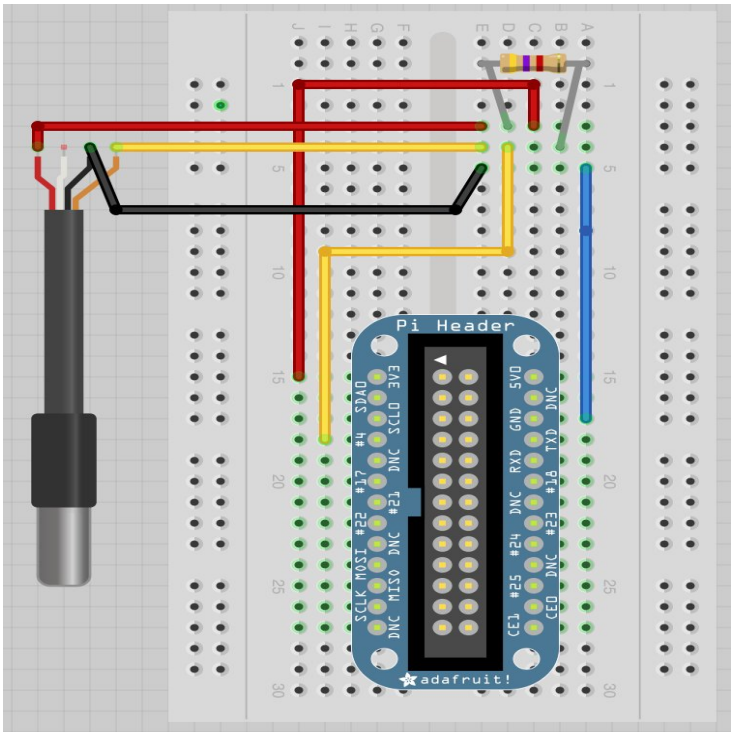

Figure 4

Then we developed the following code (monitor.py), which will capture the temperature and the current time and save them inside the “templog.db” file, see Figure 5.

```
#!/usr/bin/env python

from datetime import datetime
import sqlite3 as mydb
import os
import time
import glob


# main function
# This is where the program starts
def main():

    tempfile = open("/sys/bus/w1/devices/28-000006574c70/w1_Slave")
    tempfile_text = tempfile.read()
    tempfile.close()
    tempC = float(tempfile_text.split("\n")[1].split("t=")[1])/1000
    tempF = tempC*9.0/5.0+32.0

    con = None

    con = mydb.connect('templog.db')
    cur = con.cursor()
    now = datetime.datetime.now()
    cur.execute("INSERT INTO Temp VALUES(?, ?)", (now, tempC))
    con.commit();
    con.close()

if __name__=="__main__":
    main()
```

Figure 5

B. Stepper Motor Code

In order to connect the Stepper Motor, we first need to attach the Stepper & DC Motor Hat, which will make it easier for us to control the Motor from the python script. We followed the tutorial on the Adafruit.com to make the correct connection, see Figure 6.

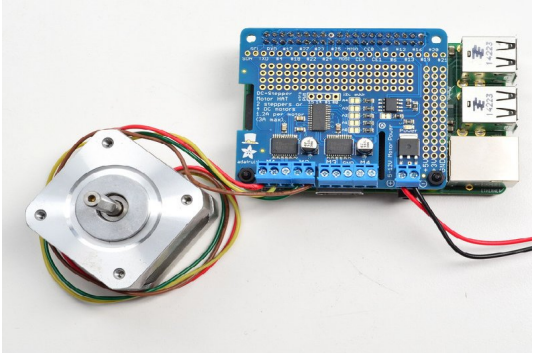

Figure 6

Then, we developed the code for the stepper motor, which will control the movement of the Stepper Motor, FORWARD or BACKWARD. Also, since we have attached pulley on the other side of the stepper motor, the pulley will then control the movement of the string that is attached to the temperature sensor, which will move either UPWARD or DOWNWARD, see Figure 2a & 2b. However, for the purpose of our project, we need the string, or the temperature sensor, to go down on three fixed depths (7 cm, 14 cm, 21 cm). First, the temperature will go down 7 cm and them it will stop, at this point we will call the (monitor.py) file to capture the current temperature and current time and save them on “templog.db”. Then, the temperature will wait for little while it collect the right data and then will move down 7 cm more, which will be at depth 14 cm and then stop. We do the same thing as we did in the first stage, and then continue to 7 cm more to reach at depth 21 cm and then stop. After we collect the data (temperature and current time), we scroll back up to the starting point, in which the temperature sensor will move up 21 cm until it reaches the starting point. Then we wait for fixed time then repeat the same steps to get an update, see Figure 7a and 7b.

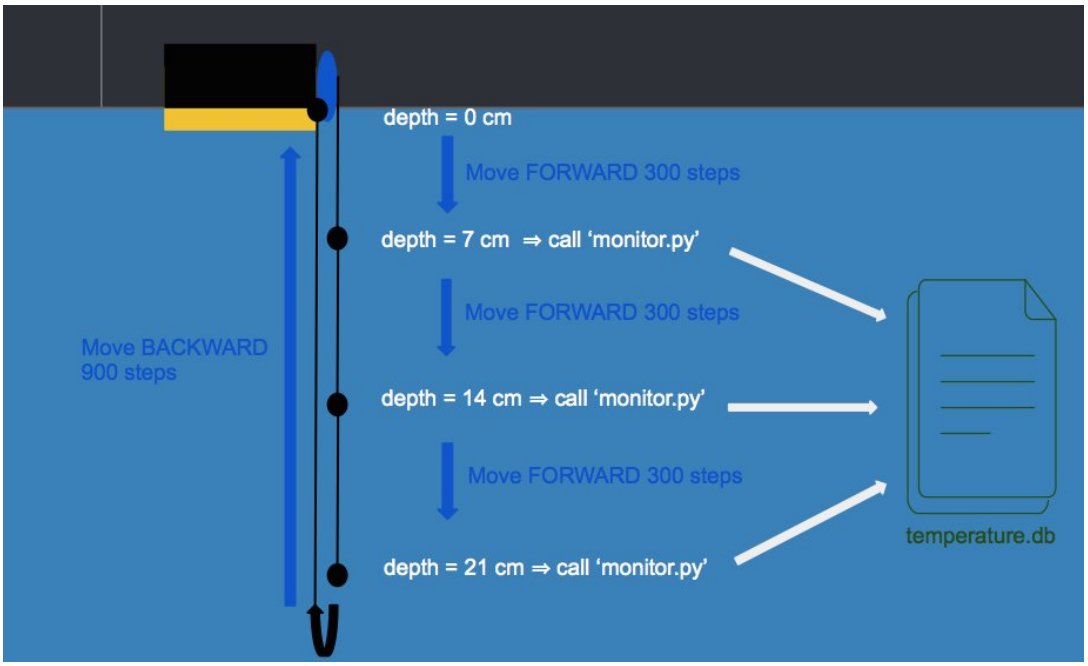

Figure 7a

```
#!/usr/bin/python
from Adafruit_MotorHAT import Adafruit_MotorHAT, Adafruit_DCMotor, Adafruit_StepperMotor

import time
import atexit

# create a default object, no changes to I2C address or frequency
mh = Adafruit_MotorHAT()

# recommended for auto-disabling motors on shutdown!
def turnOffMotors():
    mh.getMotor(1).run(Adafruit_MotorHAT.RELEASE)
    mh.getMotor(2).run(Adafruit_MotorHAT.RELEASE)
    mh.getMotor(3).run(Adafruit_MotorHAT.RELEASE)
    mh.getMotor(4).run(Adafruit_MotorHAT.RELEASE)

atexit.register(turnOffMotors)

myStepper = mh.getStepper(200, 2)   # 200 steps/rev, motor port #1
myStepper.setSpeed(200)         # 30 RPM

while (True):

    print("Double coil steps")
    myStepper.step(300, Adafruit_MotorHAT.BACKWARD,  Adafruit_MotorHAT.DOUBLE)
        # Measure Tempreture on Depth = 7 cm
        # execfile("monitor.py")
        time.sleep(5)
        myStepper.step(300, Adafruit_MotorHAT.BACKWARD,  Adafruit_MotorHAT.DOUBLE)
        # Measure Tempreture on Depth = 14 cm
        # execfile("monitor.py")
        time.sleep(5)
        myStepper.step(300, Adafruit_MotorHAT.BACKWARD,  Adafruit_MotorHAT.DOUBLE)
        # Measure Tempreture on Depth = 21 cm
        # execfile("monitor.py")
        time.sleep(5)
        myStepper.step(900, Adafruit_MotorHAT.FORWARD, Adafruit_MotorHAT.DOUBLE)
        time.sleep(15)
```

Figure 7b

C. Server Communication Code

The data read from the sensor are stored locally, so we need to send this data to the server when an internet connection is available. This component is responsible for checking for internet connection, reading the data from the local database and send it to the server. As soon as an internet connection is available, the component will read all the data stored locally and format it into a defined JSON format. The component then makes a POST request to the web server passing the JSON containing the data. The webserver gets the data and stores in the database sending back a response code.

The first part of this component is to check whether or not exists network connection. The script below does this job.

```
# Purpose: Check wifi connection and execute python program if a connection is available

PYTHON_FILE='/home/pi/ELSpring2015/code/project/send_data.py'

LOG_CONNECTION='/home/pi/ELSpring2015/code/project/log_connection.log'

status=''

ping -c 4 google.com > /dev/null

if [ $? != 0 ]
then
  status="$(date): No network connection available!"
else
  status="$(date): Connection available! Sending data..."
  sudo python "$PYTHON_FILE"
fi

echo "$status" >> "$LOG_CONNECTION"
```

This script was written and saved in ‘usr/local/bin/’ with the following permissions:

```
sudo chmod 775 /usr/local/bin/checkwifi.sh
```

In order to make it running every certain time, we used the crontab editor typing “crontab -e” into the raspberry pi system.

```
*/15 1 * * * /usr/bin/sudo -H /usr/local/bin/checkwifi.sh
```

This allows us to run the script every after 1 hour and 15 minutes. If a connection is establish, it will run the python code “send_data.py”

The python code represents the second part of this component. It works as follow:

- Check rows available in the data table
- Encode the data into a JSON format
- Send JSON to the web server by a POST request

The sqlite3 library is used to access the database file running on the raspberry pi. The program will do nothing if there are no rows in the data table. The functions below manipulates the data from the sqlite.

The getRow() function will check if there is any row in the table, one row is enough to transmit it to the server.

```
# It checks if there is any row in the the data table
def getRow():
        con = db.connect(DB_NAME)

        with con:
        cur = con.cursor()

        cur.execute("""SELECT count(*) FROM data LIMIT 1""")

        count = cur.fetchone()

        return count[0]
```

The function getSensors() returns all serial sensors linked to each data.

```
# It returns the sensors' serials used in the buoy
def getSensors():
        con = db.connect(DB_NAME)

        with con:
        cur = con.cursor()

        cur.execute("""SELECT serial FROM sensor""")

        sensors = cur.fetchall()

        return sensors
```

The getData() function will receive the serial and return all data linked to it.

```
# It returns the data for each sensor attached on the buoy
def getData(serial):
        con = db.connect(DB_NAME)

        with con:
        cur = con.cursor()

        cur.execute("""SELECT time, depth, value FROM data WHERE sensor_id
= (?)""", (serial))

        data = cur.fetchall()

        return data
```

Python dictionaries and lists are used to encode the data from the table into a JSON format. The collections and json libraries were imported to make this job done. The function receives the list with the dictionary and returns its json format.

```
# It defines the JSON format to send the POST request to the server
def defineJSON(object_list):

    objects_list1 = []

    b = collections.OrderedDict()
    b['id'] = RASPBERRY_ID
    b['sensors'] = objects_list

    objects_list1.append(b)

    js = json.dumps(objects_list1[0])

    return js
```

The hhtplib library provided the functions to establish the connection with the web server and send a POST request to it. The function sendData will receive the JSON and send it to the webserver.

```
# It sends the data to the web server
def sendData(json):

    httpServ = httplib.HTTPConnection(WEBSERVER_HOSTNAME)

    httpServ.connect()
```

```
        httpServ.request('POST', WEBSERVER_PATH, json)

        response = httpServ.getresponse()

        httpServ.close()

    return response.read()
```

After the data is successfully transmitted, all rows are deleted.

D. Website Code

The website is used to display the data in form of a chart. It has a backend database with the data sent from the PI by the communication component. The web site gets the data from the database and draw charts for displaying.

The server database was designed and created. We used MySQL Workbench for creating the database model. It was designed in such a way that it is highly extensible when new PIs and sensors are added to the system, see Figure 8.

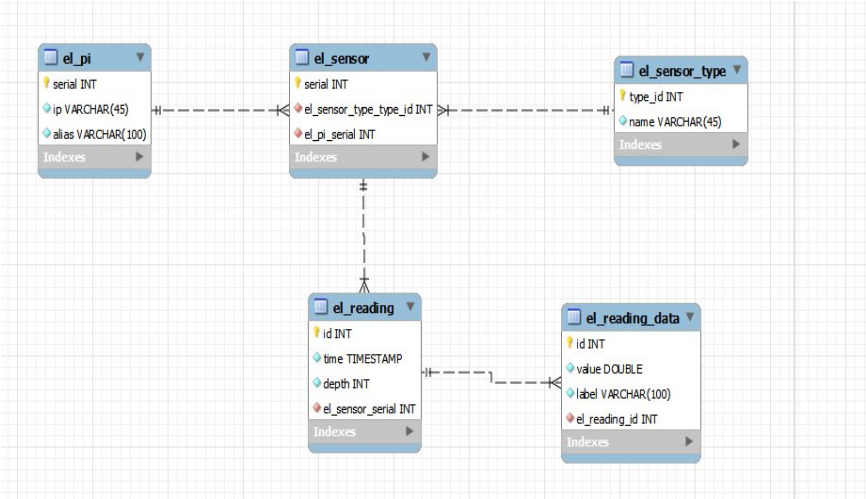

Figure 8

For drawing the charts we are using Google Charts. It is a powerful chart plotting tool by google. It is free, reliable and have many functionalities. The webpage we be also built on top of bootstrap, so that it has a solid css base and helps on the arrangement of the charts in the page, in addition to mechanisms to support mobile devices. We created a simple web page to learn how to use the google charts api, and have a basic filling of the charts arrangement on the website, see Figure 9.

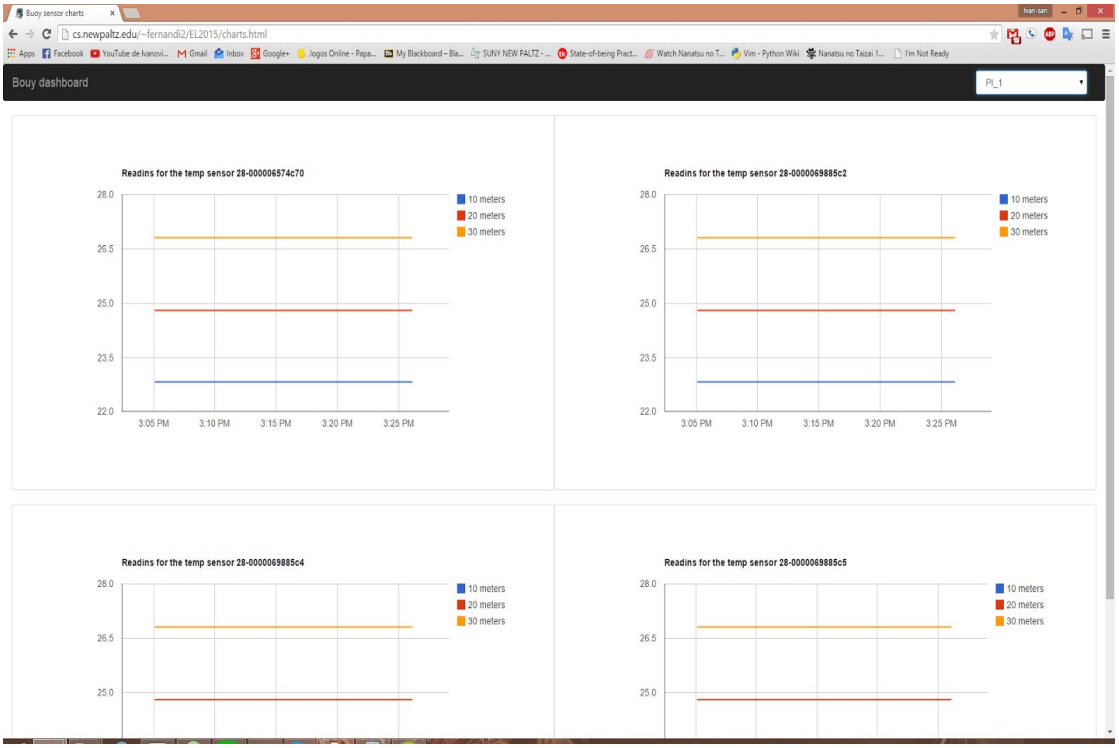

Figure 9

Summary

This project is a prototype that can demonstrate that actual act of a buoy that has a temperature sensor, in which it can move downward or upward, it order to capture the temperature sensor at deferent depths and sends them to the server throughout the nearest network available. After the data is sent to the server, the data will analyzed to be shown on a Google chart representation, which shows the a statistical chart of the data. The electronic parts of the project are; Raspberry Pi B+, Temperature Sensor, Stepper Motor 12V, Steppe Motor Hat for Raspberry Pi B+, Breadboard, and wires. The software of the project contains four parts, Temperature Sensor Code, Stepper Motor Code, Server Communication Code, and Website Code. The motor will stop three times at different depths, 7 cm, 14 cm, 21 cm. At each depth it will capture the temperature measurement and sends it to the temperature.db file. After the Stepper Motor goes back again to where it was, it sends the data that is in the temperature.db to the nearest network connection available. After that, the data is been captured and shows it on the website using PHP and Google charts. The project was tested many times and it has been demonstrated in the class, which fulfills the ultimate goal of the project.

Member Contribution

Fabricio - He will be focusing on programming the raspberry pi to have the temperature sensors read the data and transmit it to a web server. Also making sure to include important details such as temperatures in fahrenheit and celsius, and details about the date and times of when the data is recorded. He will also need to specify which sensor is recording at what time and at what depth according to the pulley system.

Ivan - He will be focusing on programming the web server that will be able to receive data from our raspberry pi. The web server he is creating will also plot the data in a user-friendly graph where anyone could see. He will also record the data on when it was taken and which sensor it was taken from at that depth.

Erin - He will be focusing on hardware and wiring of the parallel sensors. Knowing which ports need to be connected where. Also thinking of the idea of having a similar representation of different sensors being implemented into the design and knowing how to work with them.

Mahmoud - He will be focusing on the architecture of the buoy, and being able to create a design that will house the materials needed to float on top of water. Make sure that the design covers anything that needs to be kept from water. Also having a location on the buoy where we will be able to hang the wires and sensors from the buoy without water entering. He will also need to keep in mind the fact that we will have a motor and pulley system attached to the buoy and figure out the location of where everything needs to go.

Theodore - He will be focusing on the hardware and architecture of being able to implement a pulley system for the different sensors. The idea is to have a system where sensors will be lowered into the water at a specific depth, record the data after a short period of time, then have the pulleys return to their original location after collecting the data. He will need to keep in mind that they will be doing this working with the pi and the stepper motor.

Kevin- Organized group meetings and contributions, Researched and created an implementation plan according to requirements and distributed workload, Tested and supplied various physical components.