Distributed Home Environment Monitoring System

KEYANG YU, Florida International University, United States

As a widely applied smart home management device, Raspberry Pi has been deployed in various smart facilities. Most smart thermometers or motion sensors have limit capabilities for customization, and most of them are close sourced, which make it difficult for users to add additional functions. This work proposed a simple, low-cost, and highly user-tunable home environment monitoring platform based on Raspberry Pi and 2 digital sensors - DHT11 and HC-SR501. The system comes with distributed data collection, storage and visualization, which is easily expandable.

Additional Key Words and Phrases: IoT, Smart Home, Data Visualization

ACM Reference Format:
Keyang Yu. 2018. Distributed Home Environment Monitoring System. 1, 1 (December 2018), 4 pages. https://doi.org/10.1145/1122445.1122456

1 OVERALL DESIGNING

The designing can be devided into three main phases:

- Connect temperature humidity sensor and infrared motion sensor to the GPIO slots on Raspberry Pi, and collect the real-time data. Prepare for sending the data though Socket to an Windows platform.
- Build a MySQL database to save the temperature and humidity data. The server side application receives data from Socket and decode them for MySQL.
- Visualize the data from MySQL through HTML5 webpage, supports auto refreshing for showing the latest data change.

The Socket communication on the client side (Pi) is pretty much similar as our assignment 2. Each message contains the temperature, humidity and timestamp on the Pi (which actually is the UTC time), arranged in a fixed format, and encoded into bitstream to send to our Windows device in local area network.

On our Windows device, we struggled pretty much on importing a proper Python driver for accessing our MySQL database and in vain. So We considered using Maven, which is able to automatically fetch required packages for a Java application. Thus, we designed a Java server to receive message from our Pi client.

The server application will first decode the bitstream, and read the temperature, humidity, and timestamp data into memory by conventional format. Then it will insert the data into MySQL database for further processing.

To visualize our data, we considered an open-source project named Chart.js. Which is able to plot several types of diagrams according to data from databases or even txt files. We made several modifications to the original project to fit our project. Which includes webpage auto refreshing, plotting multiple lines in one chart and so on.

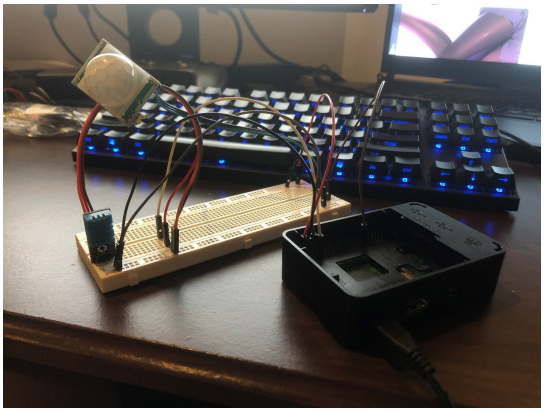

Fig. 1. Hardware Overview

2 DESIGN BREAKDOWN

2.1 Raspberry Pi

2.1.1 Hardware. We first connected the DHT11 (temperature humidity sensor) and HC-SR501 (infrared motion sensor) onto a bread board. The blue component on the bottom left part of Figure 1 is DHT11, and the white hemi-sphere on the top left part of Figure 1 is HC-SR501.

Figure 2 and Figure 3 showed the connection details for both sensor modules to the Pi. The DHT11 uses 3.3V as Vcc, and HC-SR501 uses 5V as Vcc, so we connect pin 1 (black wire) and pin 2 (red wire) separately to the breadboard as two different power source for our sensors. And we use pin 6 (gray wire) as ground for both sensor. Then, we connected GPIO4 (pin 7, white wire) to DHT11, and GPIO5 (pin 29, brown wire) to HC-SR501 for data exchanging.

2.1.2 Software. In the datasheet of both sensor module, the manufacturer provided the rule for handshake and serial data format. We can easily control the GPIO slots in Python applications.

We take DHT11, the temperature sensor for example. The GPIO object in the code provides several operations to the assigned slot as described in the comments (Figure 4). Since the temperature and humidity data was sent through 40 bit serial data, we need to decode them into integers for easier processing.

In Figure 5, the serial data was first segmented (line 43 to 54), and then decoded (line 56 to 68). The first 8 digit of the serial data is the integer part of humidity, 9th to 16th digit is the decimal part of humidity. The 17th to 24th digit is the integer part of temperature, and 25th to 32nd digit is the decimal part of temperature. Since the serial data is possibly contains errors in some digits due to level interference, the 33rd to 40th digit is checksum for previous digits. If

Author's address: Keyang Yu, kyu009@fiu.edu, Florida International University, 11200 SW 8th Street, Miami, Florida, United States.

XXXX-XXXX/2018/12-ART $15.00
https://doi.org/10.1145/1122445.1122456

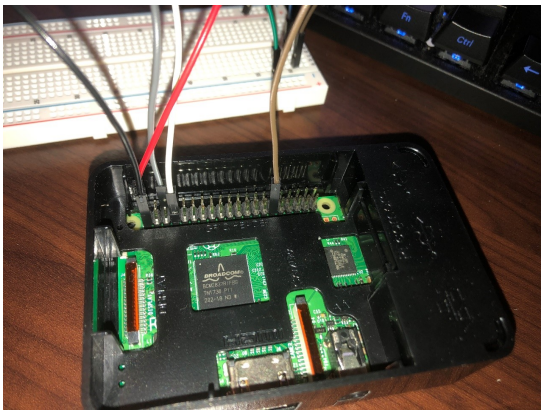

Fig. 2. GPIO Connections

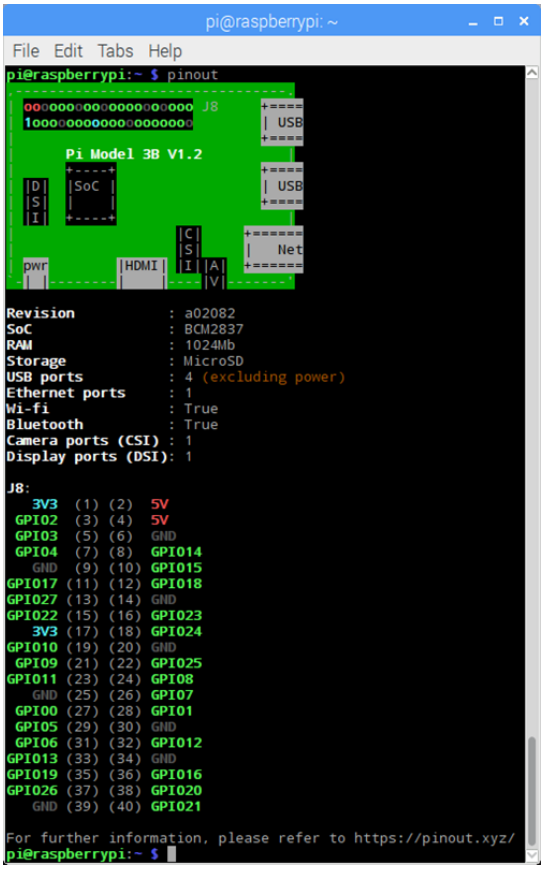

Fig. 3. GPIO Pins for Raspberry Pi

the checksum is not corresponding to the previous data, this whole serial data will be discarded (line 66). The serial message is showed in Figure 6.

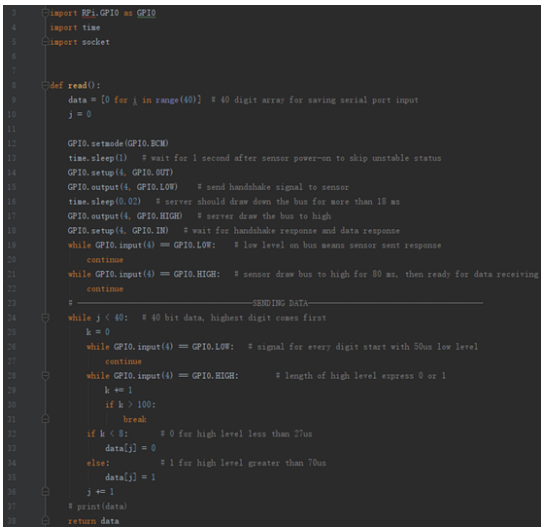

Fig. 4. Handshake and data reading for DHT11

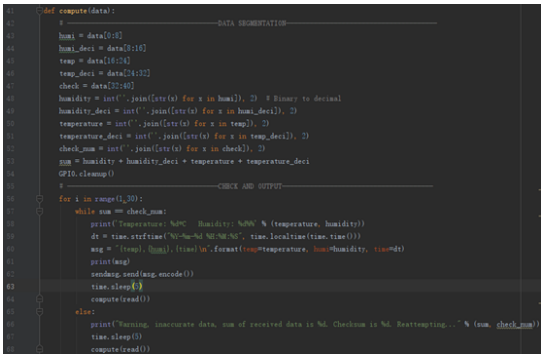

Fig. 5. Data decoding for DHT11

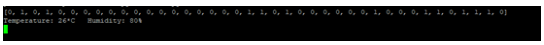

Fig. 6. Data sample of DHT11

2.2 Database

As introduced in Section 1, we developed a Java application acts as server to receive the data from Socket and insert data into MySQL database. To import correct driver packages for accessing MySQL, we first need to create a pom file for Maven. Then the required packages will be automatically imported.

When inserting into MySQL, we need to decode the message into three different data by using comma as separator (Figure 10 line 21). The format of MySQL table is shown in Figure 12. The first row is an auto incrementing ID for just identifying. The second and third row are the integer part of temperature and humidity. The reason why we didn't consider the decimal part will be demonstrated in next section, design tradeoffs. And the fourth row is the timestamp

Fig. 7. Maven pom

```
1  package com.gky666.server;
2
3  import java.io.*;
4  import java.net.ServerSocket;
5  import java.net.Socket;
6  import java.text.ParseException;
7  import java.text.SimpleDateFormat;
```

Fig. 8. Package List

```
1  package com.gky666.server;
2
3  import java.sql.*;  //java package for accessing MySQL
4  import java.util.Date;  //java package for date
```

Fig. 9. Package List

```
17          InputStream is = socket.getInputStream();
18          BufferedReader in = new BufferedReader(new InputStreamReader(is));
19          String info;
20          while ((info = in.readLine()) != null) {
21              String[] contents = info.split(",");
```

Fig. 10. Data Decoding

```
69  public void write(Connection conn, Integer temp, Integer humility, Date timeStamp) throws SQLExce
70      System.out.println("Data inserting...");
71      String sql;
72      sql = "INSERT INTO os_final(temp,humi,time_stamp) VALUES (?,?,?)";
73
74      try {
75          PreparedStatement ps = conn.prepareStatement(sql);
76          ps.setInt(1, temp);
77          ps.setInt(2, humility);
78          ps.setTimestamp(3, new Timestamp(timeStamp.getTime()));
79          ps.executeUpdate();
80          System.out.println("Successfully inserted.");
81      }
82      catch (SQLException e) {
83          e.printStackTrace();
84      }
85  }
86
87  public void writeDbContent(Connection conn , Integer temp, Integer humility, Date timeStamp){
88      try{
89          write(conn, temp, humility, timeStamp);
90      }
91      catch (SQLException e){
92          e.printStackTrace();
93      }
94      finally {
95          try{
96              conn.close();
97          }
98          catch (SQLException e1){
99              e1.printStackTrace();
100         }
101     }
102 }
```

Fig. 11. Data Inserting

created on the Pi, which is actually the UTC time, in YYYY-MM-DD HH-MM-SS format. As shown in Figure 10 and Figure 11, we packed write method into writeDbContent method, mainly in order to follow the encapsulation.

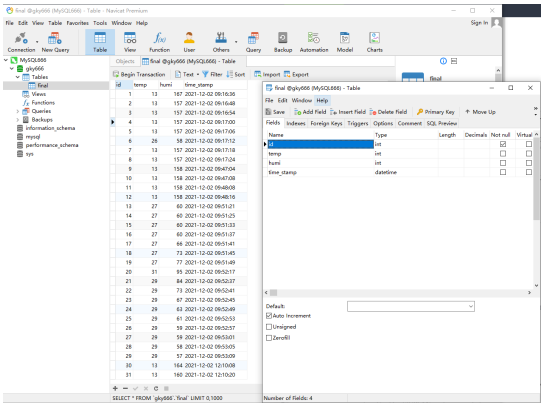

Fig. 12. Table Overview

Fig. 13. Data Fetching

Fig. 14. Page Refreshing

2.3 Chart Rendering

In chart rendering part, we designed a PHP method to fetch data from assigned MySQL table.

Since the Pi client keeps sending messages and the database keeps adding data into the table, the timeliness of our chart is essential. So we refresh the web page every 3 seconds.

Since we want to show the temperature and humidity in one chart, which is easier to see the data in one particular time point. We decided to use timestamp as a common X-axis, and plot two separate lines. The implementation is shown in Figure 15.

3 DESIGN TRADE-OFFS

There are several sacrifices in this project. Most of them are due to the complexity of the whole system, and the lack of knowledge in several particular disciplines like distributed system managing and database managing.

In the Raspberry Pi section, most of our tentative plan has been achieved. The DHT11 and HC-SR501 sensor works correctly, and Socket communication is stable. However, we failed to connect the motion sensor to Homeassisstant, which means the Wemo switch cannot be triggered by the motion sensor.

In the MySQL section, as we introduced before, we waived on using Python to access the database, mainly due to the lack of knowledge on Python packages. However, the Mavon and Java

Fig. 15. Chart Plotting

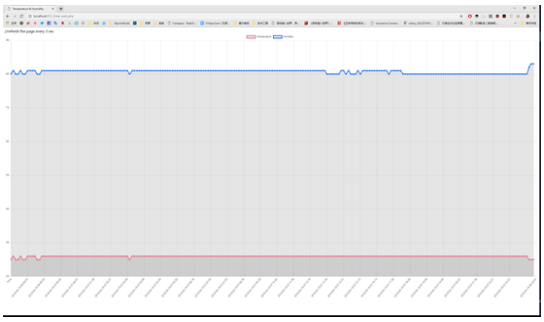

Fig. 16. Visualization Results

application runs pretty well, the connection between Java server and Python client is stable.

We discarded the decimal part of our temperature and humidity data since the digit of decimal is usually different among different readings. However, we changed our method by using info.split method, which we can use comma for separator instead of read some digit. Thus this problem was eliminated, and we will add decimals into our data in future version.

Since we applied several custom setting on our router, Raspberry Pi, and MySQL database, our project is difficult to reproduce in other network environments, such as in the classroom. We will try to improve the universality of our system. The Raspberry Pi has some unknown problems which cause the failure on accessing it from Internet. The problem may be due to the unexpected modification to critical files by some services.

On the hardware side, we found that the readings from DHT11 sensor have a high error rate, as shown in Figure 17. The DHT11 sensor was connected directly to the GPIO slot on the Raspberry Pi, and the VCC, GND may drifted from the designated value. Further more, the wire connecting the sensor and the Pi may also have higher resistance than expected, which may cause insufficient pull-up voltage. To improve the performance, we may need to properly ground the Pi, and use shorter wires to lower the resistance.

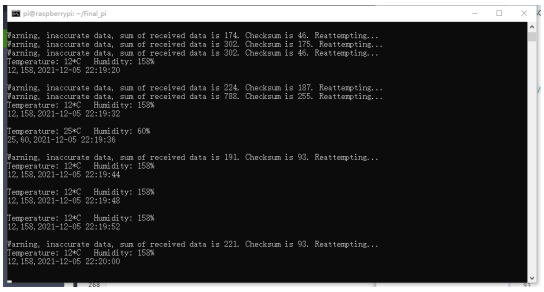

Fig. 17. Error Readings

4 POSSIBLE IMPROVEMENTS AND EXTENSION

This project is mainly to complete some basic work for the design of advanced IoT projects. We built a set of distributed devices to collect, store and displaying data. Unlike using existing platforms like Homeassisstant, which may exist several restrictions on operating devices or services, we can extend the job to more complex environments. What's more , the sensor we chose is more flexible and atomic comparing with IoT devices on market. Which means the extendibility and robustness is higher.

Since we haven't mastered the skill of building a long-term connection between Java application and MySQL database, in order to prevent unexpected connection failure, we open a connection when we need to insert data, and close that connection right after we finished inserting. This method has a some limitation, mostly on the time resource trumpet. In this situation, we may not be able to deal with messages coming in a relatively high frequency (1 message per second or so), for the time cost for connecting is unstable, and these operations are pretty costly. This problem could be fixed by importing database connection pool, such as Mybatis, which is a famous Java persistence framework for database management.