

# Home Security System Using Raspberry Pi

A Design Project Report

Presented to the School of Electrical and Computer Engineering of Cornell University  
in Partial Fulfillment of the Requirements for the Degree of  
Master of Engineering, Electrical and Computer Engineering

Submitted by

Jingyi Wang (jw2527), Haodong Ping (hp394), Manquan Fang (mf734)

Meng Field Advisor: Bruce Land, Joe Skovira

Degree Date: December, 2019

# **Abstract**

Master of Engineering Program

School of Electrical and Computer Engineering

Cornell University

Design Project Report

**Project Title:** Home Security System Using Raspberry Pi

**Author:** Jingyi Wang, Haodong Ping, Manquan Fang

**Abstract:**

This project is designed to develop a home security system based on Raspberry Pi which realizes the function of providing residence with security and alarm information through remote sensing. To complete this project, team members use temperature and humidity sensor and PIR sensor to detect the temperature, humidity and motion information of a house, and then report any potential unusual or dangerous situations to the owner. In order to ensure that the owner can get the potential unusual or dangerous situations in time, security notifications will be by email, and all information will be by WiFi to a commodity wireless router. The final products are robust, packaged, and powered by mains. The goal of this project is to replace the current DLINK security system with a system that does not need to go out to the cloud for information sharing and control.

## **Executive Summary**

This home security system is aimed to use temperature and humidity sensor and PIR motion sensor to detect the temperature, humidity and motion information of a house. If it detects any potential unusual or dangerous situations, it should report to the owner via email. What's more, it is packaged within a required size and powered by mains. Users can start this system from PC via SSH.

In order to complete this project, our team members should do several things. For software design, we should get data from sensors. Then we should check if anything unusual is detected by sensors and send out notification via emails. We should also set our equipment to automatically work when powering by mains. For hardware design, we should connect PIR motion sensor and temperature and humidity sensor directly to Raspberry Pi. For mechanical design, we should integrate hardware part and software part, then package this equipment within a required size.

After integrating software part and hardware part and packaging it, we plug it and make it work for one week. The result proves that this system can work successfully.

## Distribution of Work

For the report, individual distribution is shown as below:

	Introduction	Design	Issues	Results	Future Work	Conclusion
Haodong Ping		√	√	√		
Manquan Fang		√	√			
Jingyi Wang	√	√	√	√	√	√

For the project, individual distribution is shown as below:

	Software Design	Hardware Design	Mechanical Design	Poster
Haodong Ping	√			
Manquan Fang			√	√
Jingyi Wang		√		

1. Introduction.....	6
2. Design.....	7
2.1 Software Design.....	7
2.1.1 Sensor.....	7
2.1.2 Email Sending.....	9
2.1.3 Main Program.....	10
2.2 Hardware Design.....	11
2.2.1 PIR Sensor.....	11
2.2.2 Temperature and Humidity Sensor.....	13
2.2.3 Connect Sensors to Raspberry Pi.....	14
2.3 Mechanical Design.....	18
3. Issues.....	29
3.1 Software Part Issues.....	29
3.2 Hardware Part Issues.....	30
3.3 Mechanical Part Issues.....	31
4. Results.....	33
5. Future Work.....	37
6. Conclusion.....	37
Appendix.....	35

# 1. Introduction

Nowadays, people attach much more importance to home security. Home security is related to both the security hardware in place on a property and personal security practices. Security hardware includes doors, locks, alarm systems, lighting, motion detectors, security camera systems, etc. that are installed on a property. However, current home security devices need to update the personal data to the cloud, which brings users the risk of losing privacy. What's more, since all data are processed in the cloud server, it is inconvenient for users to change control conditions.

In this project, team members use Raspberry Pi as the server to replace the cloud server in order to solve the data privacy problem. Raspberry Pi is a credit card-sized single-board computer and a very powerful control unit. Users can connect the sensors to the Pi, and design programs to make it deal with those data. When Raspberry Pi detect some abnormal situation, like the temperature is too low and the movement of strangers, it can send email to users directly. Raspberry Pi is also very cheap. It costs 35 dollars to buy a Raspberry Pi 3 and only costs 5 dollars to buy a Raspberry Pi 0. As a result, we use Raspberry Pi 3 to design and then transfer from Raspberry Pi 3 to Raspberry Pi 0.

In the designed home security system, all the sensors, including PIR sensor, temperature and humidity sensor, will be connected to Raspberry Pi directly and Raspberry Pi will be worked as server to process all the data collected by sensors. When Raspberry Pi detects something wrong, it can send a warning email to user directly instead of going out to the cloud for

information sharing and control.

## 2. Design

### 2.1 Software Design

In our project, the system is based on Raspberry Pi, all the sensors will be connected to Pi directly and Pi will be worked as server to process all the data collected by sensors. When Pi detect something wrong, it can send a warning email to user directly. The whole system design of the solution is as shown below.

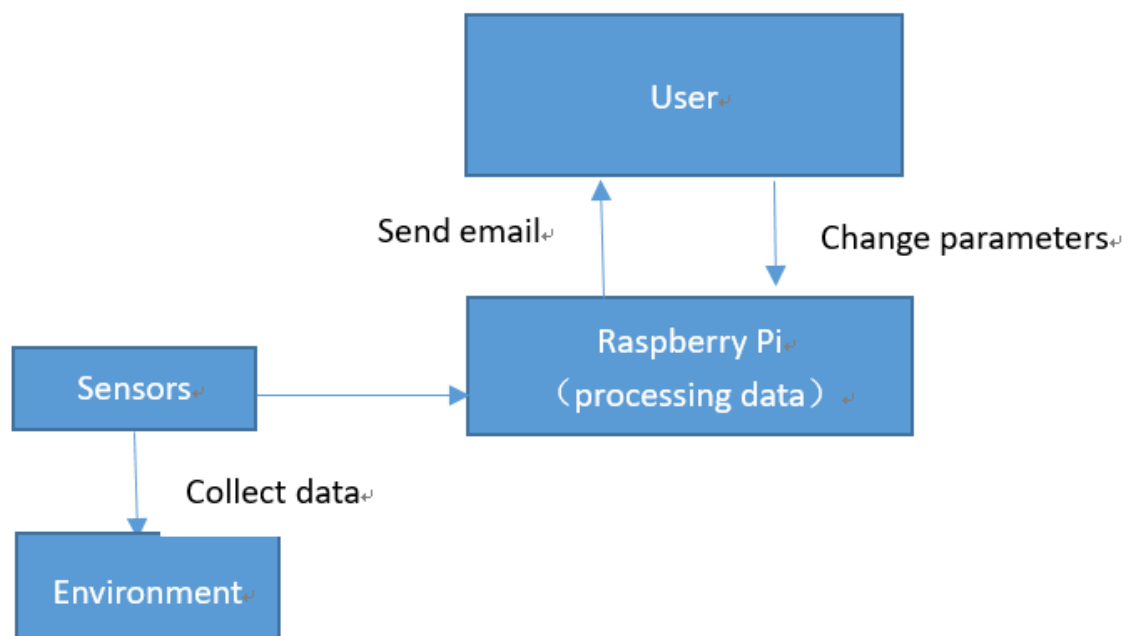


FIG 1 Pin map of HC-SR501 PIR motion sensor

#### 2.1.1 Sensor

In our project, there are two sensors we need to handle, the PIR sensor and the temperature

and humidity detection sensor. In the programming part, we apply the idea of object-oriented programming and regard every sensor as an object. In their constructor, the parameter we need to allocate a PIN number and initialize the corresponding IO state. There is also a method used to run itself to detect environment parameters and the programming part is not very difficult since we have the whole document of those sensors. Take the PIR class for example, we can use the following code to initialize this class:

```
p = pir.pir(PIN_NUM)
```

Then a PIR sensor object was created in our program. And it is almost the same code for the DHT11 sensor. To make those sensors work, we could call the detecting function, it requires no parameters and it can return the data we want. The detecting function is the most important part in the class, for the PIR sensor, we just read the voltage from GPIO, for the DHT11, its communication protocol is 1-wire protocol, it can read data from only one GPIO. In this protocol, low level 50us followed by a high level of 26-28us represents 0 and low level 50us followed by a high level of 70us represents 1. So we got the basic timing sequence, the idle state of the bus is high. The host pulls down the bus and waits for DHT11 to respond. The host pulls down the bus more than 18 milliseconds to ensure that DHT11 can detect the starting signal. After DHT11 receives the start signal of the host, it waits for the end of the start signal of the host, and then sends 80us low-level response signal. After the start signal of the host is sent, the response signal of DHT11 is read after waiting for 20-40us. After the host sends the start signal, we can switch it to the input mode. We implement this part by reading the official document of DHT11 [8] instead of using the standard library, because we find it reads a lot of invalid data and it would be more flexible to implement it by



ourselves, we will introduce this part with more detail in Issue section. For the PIR sensor, the detecting function returns a Boolean value and for the DHT11, it returns an array with only two float number, in which the first one is the temperature and the second one is the humidity. To make those sensors work continuously, we should call the detecting function in a loop. And the example code of PIR are list below:

```
while True :  
    p.detect()
```

From the code we can see that we only need only three lines of code to make the sensor work, so it brings a lot of flexibility to do the unit test and makes whole program maintainable.

### 2.1.2 Email Sending

It is pretty easy to implement the email sending part using the standard library in Python, but there is still a lot of lines of code. To make our program cleaner, we also regard it as an object. In the constructor, the program will initialize the customer's email address. The example code are list below.

```
se = sendemail.sendEmail(the customer's email address)
```

Then we have an email sending class in our program. When we need to send an email, we can invoke the method called mail, this function contains 7 parameters, which is the current PIR state, current temperature and humidity, the maximum and the minimum temperature the customer allowed, and the maximum and minimum humidity limitations.

In the main loop, we can get those current value by the sensors and get the limitations from the configuration file. Finally, this function returns a Boolean value indicting if we have sent the email successfully. After we have all those parameters, we can send the email, the example code is list below:

```
ret = se.mail(temp,min_temp, max_temp, humid, min_hum, max_hum, pir_detect)
```

From the code, we can easily send the email to the customer and customer get can get the email in about 20 seconds. We can print the value of “ret” if we hope to test the current network condition.

### **2.1.3 Main Program**

In the main program, there are mainly two part, the initialize part and the main loop for collecting the data from the environment and analysis the data to decide if we need to send an alerting email.

The initialize part initialize build all the models of the sensors and email sending part using their own constructor, initialize some flag value and get the parameters from the configuration file.

The main loop is an infinite loop and in the loop, we call the detecting function of the sensor object and collect the data. According to those data, we update the corresponding flag value and decide to send the email or not according to those value. Every time after the system send the alerting email, the system will reset all the flag value and sleep for about 5 minutes

to avoid sending redundant emails.

## **2.2 Hardware Design**

In our project, all the sensors will be connected to Raspberry Pi directly and Raspberry Pi will be worked as the server to process all the data collected by sensors. When Raspberry Pi detects something wrong, it can send a warning email to users directly. As a result, the central part of hardware design is to choose proper sensors and connect them to Raspberry Pi directly.

The main function of our project is to detect and report the unusual temperature, humidity and motion information of the house. In order to detect the temperature, humidity and motion information of a house, we use PIR motion sensor and temperature and humidity sensor, and connect them to the Raspberry Pi through GPIO.

### **2.2.1 PIR Sensor**

PIR sensors, often referred to as, "Passive Infrared" sensors, enable you to sense motion. Everything emits a small amount of infrared radiation, and the hotter something is, the more radiation is emitted. As a result, PIR sensors work by detecting movement of infrared radiation emitted from warm objects.

In this project, we choose to use HC-SR501 PIR motion sensor because it is not expensive and easy to use. The inner of HC-SR501 PIR motion sensor is shown as below:

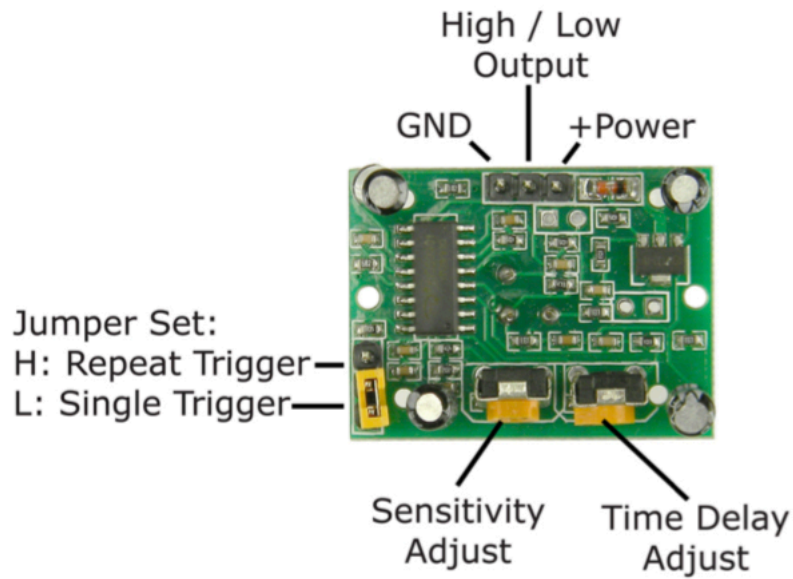


FIG 2 HC-SR501 PIR motion sensor

It has three pins: GND, +power and high/low output. At idle, when no motion has been detected, the digital out will remain low. However, when motion is detected, the digital out will pulse high (3.3V) and we'll use our Raspberry Pi to sense this. We use the sensor to detect the motion of people.

The pin map of HC-SR501 PIR motion sensor is shown as below:

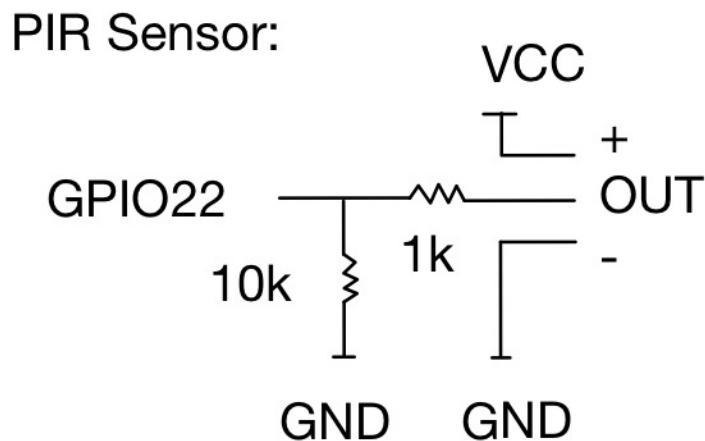


FIG 3 Pin map of HC-SR501 PIR motion sensor

### 2.2.2 Temperature and Humidity Sensor

In our project, we choose to use DHT11 temperature and humidity sensor in order to obtain the humidity and temperature information of the house. DHT 11 temperature and humidity sensor is a little module that provides digital temperature and humidity reading.

DHT11 temperature and humidity sensor uses a humidity sensor and a thermistor to measure the surrounding air, and sends a digital signal to the specific data pin. We choose to use DHT11 temperature and humidity sensor since it only needs one wire for the data signal. The pin map of DHT11 temperature and humidity sensor is shown as below.

## Temperature and Humidity Sensor:

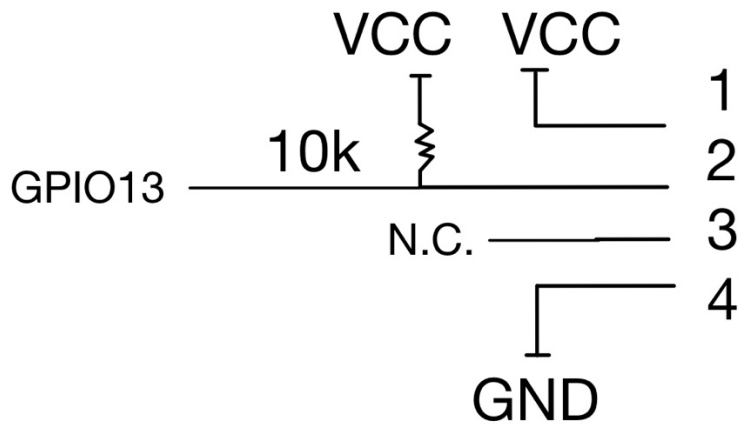










































FIG 4 Pin map of DHT11 humidity and temperature sensor

### 2.2.3 Connect Sensors to Raspberry Pi

After choosing the proper sensors, we need to connect them to Raspberry Pi directly. At first, we used Raspberry Pi 3 to design. Raspberry Pi 3 GPIO pinout is shown as below:

Raspberry Pi 3 Model B (J8 Header)					
GPIO#	NAME			NAME	GPIO#
	3.3 VDC Power	1			2
				5.0 VDC Power	
<b>8</b>	GPIO 8 SDA1 (I2C)	3			4
				5.0 VDC Power	
<b>9</b>	GPIO 9 SCL1 (I2C)	5			6
				Ground	
<b>7</b>	GPIO 7 GPCLK0	7			8
				GPIO 15 TxD (UART)	<b>15</b>
	Ground	9			10
				GPIO 16 RxD (UART)	<b>16</b>
<b>0</b>	GPIO 0	11			12
				GPIO 1 PCM_CLK/PWM0	<b>1</b>
<b>2</b>	GPIO 2	13			14
				Ground	
<b>3</b>	GPIO 3	15			16
				GPIO 4	<b>4</b>
	3.3 VDC Power	17			18
				GPIO 5	<b>5</b>
<b>12</b>	GPIO 12 MOSI (SPI)	19			20
				Ground	
<b>13</b>	GPIO 13 MISO (SPI)	21			22
				GPIO 6	<b>6</b>
<b>14</b>	GPIO 14 SCLK (SPI)	23			24
				GPIO 10 CE0 (SPI)	<b>10</b>
	Ground	25			26
				GPIO 11 CE1 (SPI)	<b>11</b>
<b>30</b>	SDA0 (I2C ID EEPROM)	27			28
				SCL0 (I2C ID EEPROM)	<b>31</b>
<b>21</b>	GPIO 21 GPCLK1	29			30
				Ground	
<b>22</b>	GPIO 22 GPCLK2	31			32
				GPIO 26 PWM0	<b>26</b>
<b>23</b>	GPIO 23 PWM1	33			34
				Ground	
<b>24</b>	GPIO 24 PCM_FS/PWM1	35			36
				GPIO 27	<b>27</b>
<b>25</b>	GPIO 25	37			38
				GPIO 28 PCM_DIN	<b>28</b>
	Ground	39			40
				GPIO 29 PCM_DOUT	<b>29</b>

**Attention!** The GPIO pin numbering used in this diagram is intended for use with WiringPi / Pi4J. This pin numbering is not the raw Broadcom GPIO pin numbers.

<http://www.pi4j.com>

FIG 5 Raspberry Pi 3 GPIO pinout

We connect PIR motion detector to GPIO19 in Raspberry Pi. The PIR will always output low (0v) unless movement is detected, in which case it will output high (3.3V). Therefore, we can set our PIR-OUT GPIO pin as an input, and use Python to detect any voltage change.

DHT11 temperature and humidity sensor only requires three connections to the Pi, that is 3.3v, ground and one GPIO pin. In this project, we connect humidity and temperature sensor to GPIO13. We use bread board to design and test, which is shown as below:

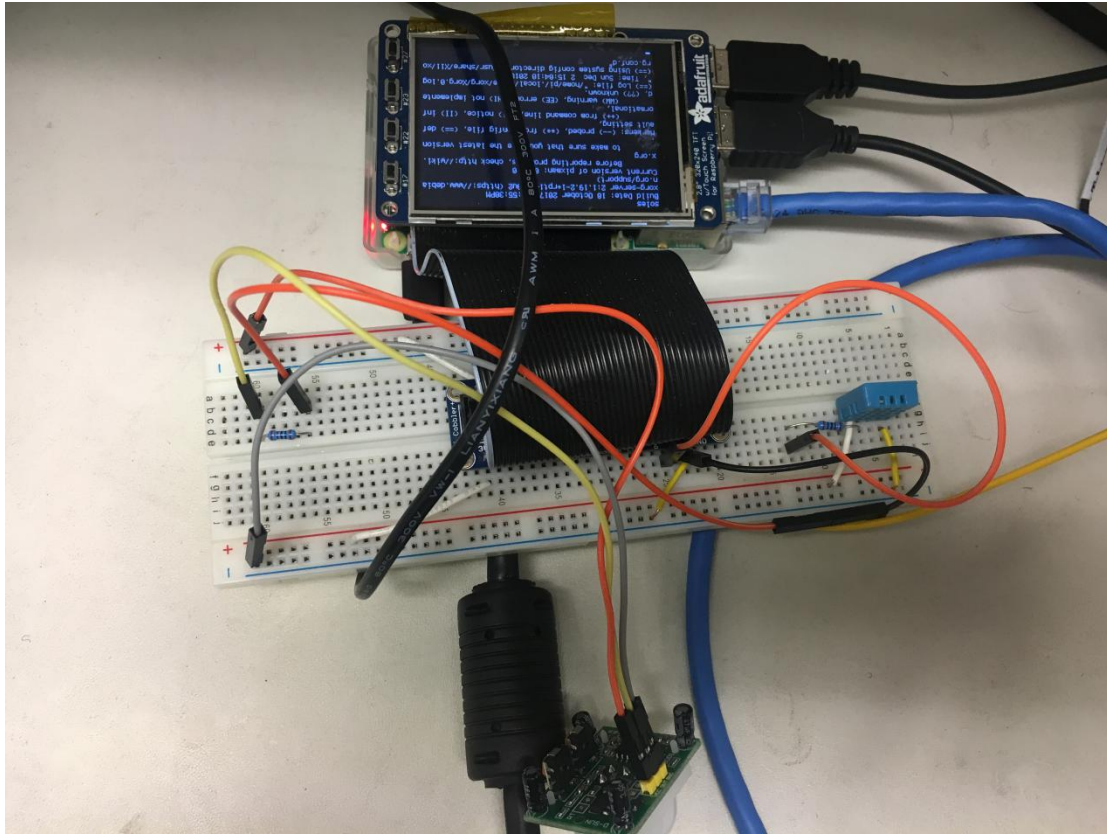


FIG 6 Test with bread board in Raspberry Pi 3

After we finished design part, we translated from Raspberry Pi 3 to Raspberry Pi 0. As a result, we should also connect two sensors to Raspberry Pi 0 directly. Raspberry Pi 0 GPIO pinout is shown as below:



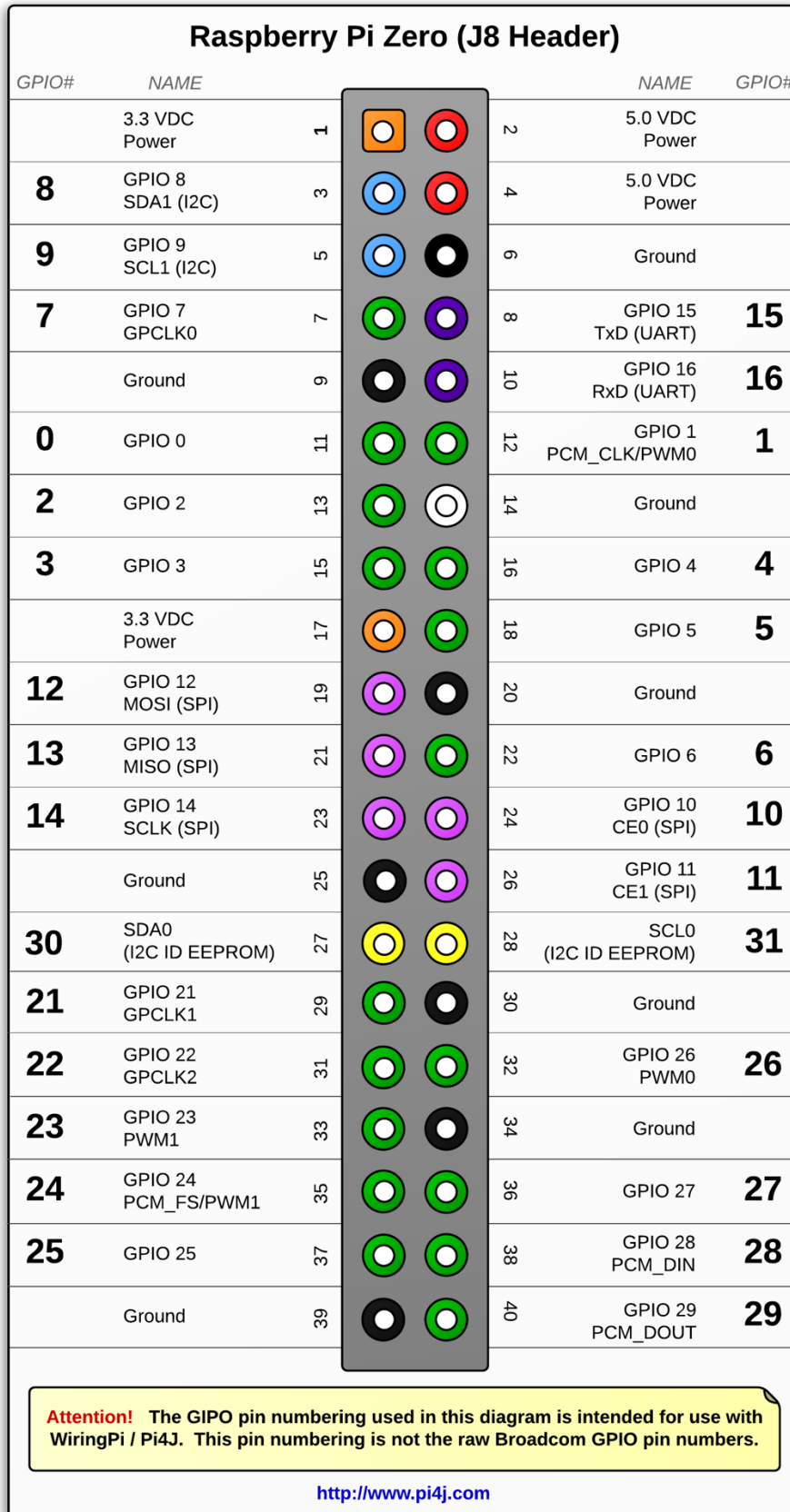


FIG 7 Raspberry Pi 3 GPIO pinout

What's more, in order to package them, we need to change from bread board to a much more compact board. In this project, we choose Perma-Proto Quarter-sized Breadboard. We directly used the jumper wires instead of soldering them on the board. The designed board with two sensors is shown as below:

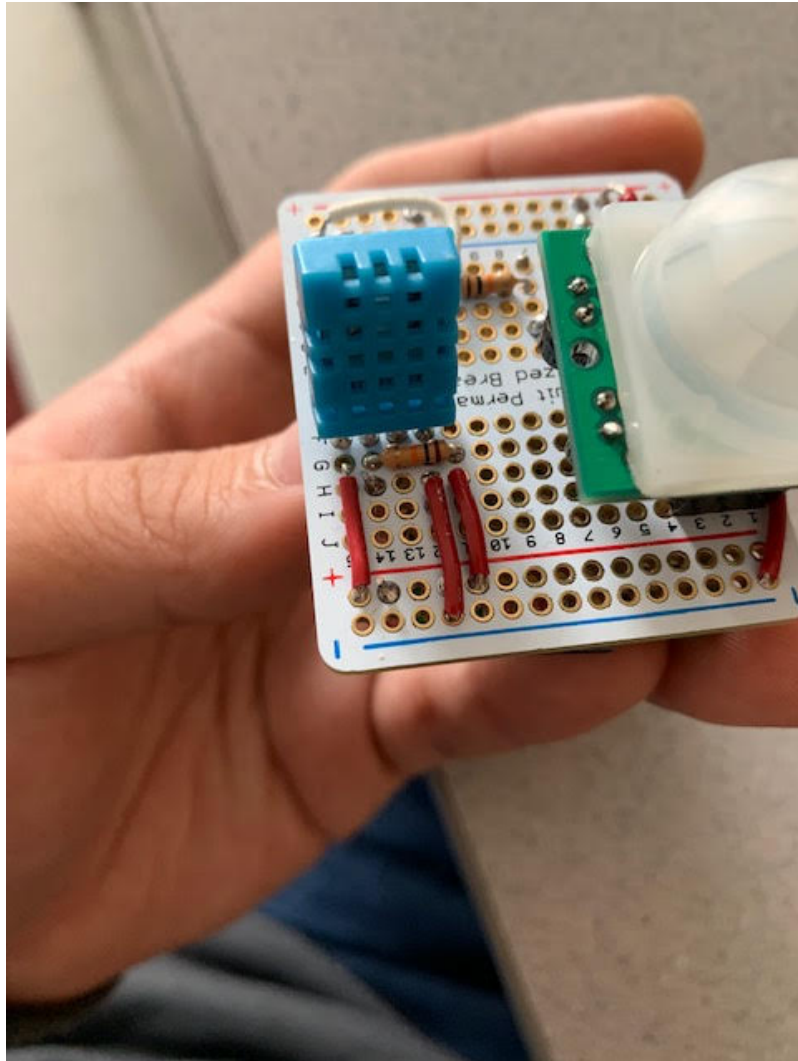


FIG 8 Raspberry Pi 0 W H (From Second Prototype)

## 2.3 Mechanical Design

The mechanical part is as important as any other parts. The software and hardware parts can be straightforward, but mechanical part requires strategically planning. The possible problems can be: which parts to use; what is the best size for a home security system; what

is the limit of both the components and out group members; does the packaging process require group members to change the design constantly (for example, the size of the box requires us to change the pin, or how we solder the sensors); how to place the components; where to put the sensors, etc. Therefore, we have solid reason to believe that mechanical part is strategical and requires the coordination and cooperation of group members.

Here is how we built it: After we chose the wanted sensors, we should decide the board and box we would use. For the board part, we first tried to start from the bread board, since it was easy to do and moderate, and below was our first prototype. In our first prototype, we used Raspberry Pi 3. After that, we switched to Raspberry Pi 0 W for our second prototype. The reason why we chose Raspberry Pi 0 was that Raspberry Pi 3 is over qualified for what we were trying to do: it has higher price and higher cost for more USB ports, better performance and Ethernet port, which are things that we do not necessarily need. Besides, it is larger than Raspberry Pi 0, which is in fact a disadvantage to our project. Therefore, we switched to Raspberry Pi 0. For Raspberry Pi 0, we did not need that size of bread board any more as well. We again, started with a bread board. This time, we chose a much compact board, which was Perma-Proto Quarter-sized Breadboard. For the second prototype, we directly used the jumper wires instead of soldering them on the board, so we could easily change the wiring. Accordingly, we used the Raspberry Pi 0 H version, which by factory has a header installed on it, so we can directly plug the jumper wires on it. As shown below in Figure 9.

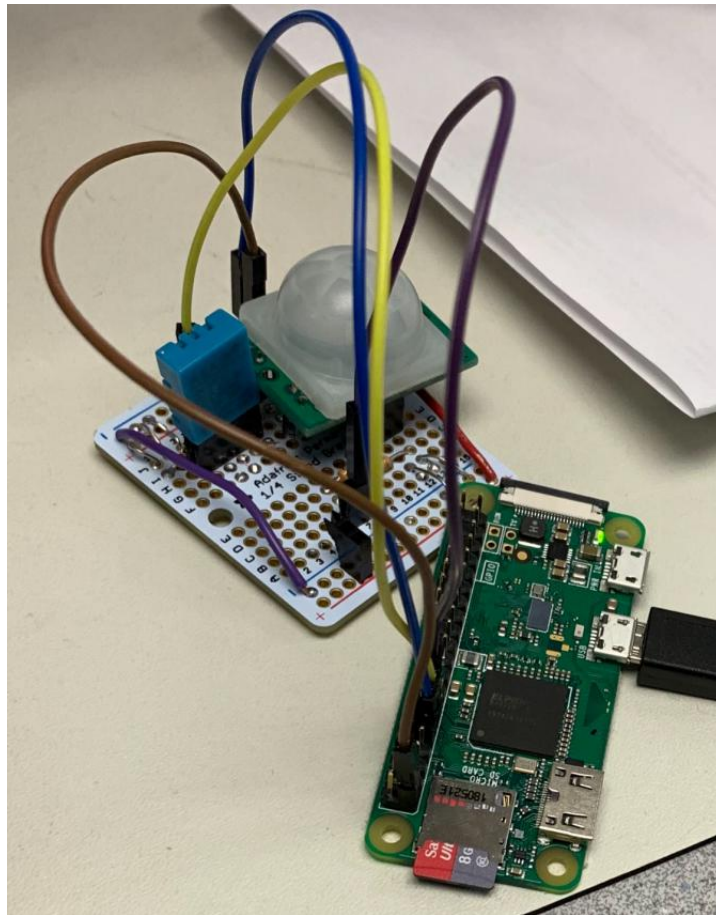


FIG 9 Raspberry Pi 0 W H (From Second Prototype)

Then, after it worked with the jumper wires, we moved on to the core breadboard wires. It turned out that breadboard with core wires were definitely good enough for this device to work, and much easier to be put into mass production in future as desired. Therefore, we decided to abandon the PCB design idea.

In order to package them, we also need a power supply for the components and a box for the whole device. For the box, material could be ABS plastic, since it is light, easy to deal with (drill holes, etc.), relatively cheap, and a large amount of selections. The selected box should be at appropriate size, which is too big for a home security device, but not too small to put all these components in. For the power supply, it needs to be sending out the power

under 5V 2A, which was the required kernel for the Pi 0. Below is the component we chose and the mechanical properties of them:

PIR Motion sensor (SR501): 32mm \* 24mm \* 25mm. As shown below in Figure 3.3.2, the PIR sensor is consisted of two parts: lower part has some pin foots which cannot be removed, which will take some space when we are placing them; upper part is a hemisphere, and it has to be placed outside through a round hole.

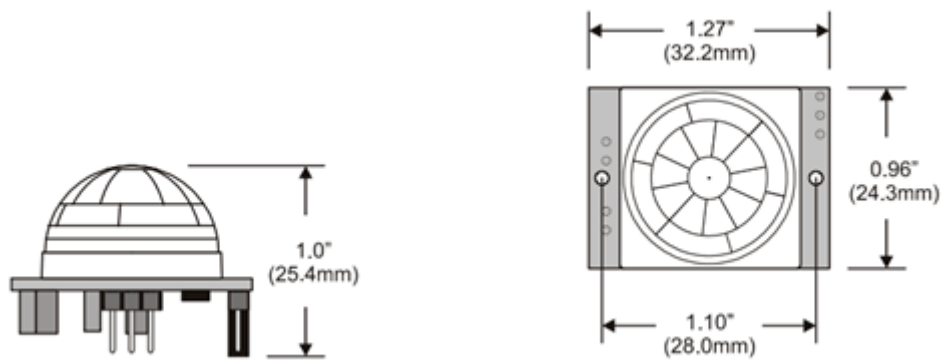


FIG 10 Dimension of SR501 PIR Sensor

Humidity & Temperature Sensor DHT11: 15.1mm \* 25mm \* 7.7mm. As show below in Figure 12, the blue part of DHT11 sensor needs to be placed outside, and its four pins has to be stuck inside of the box without being directly exposed.

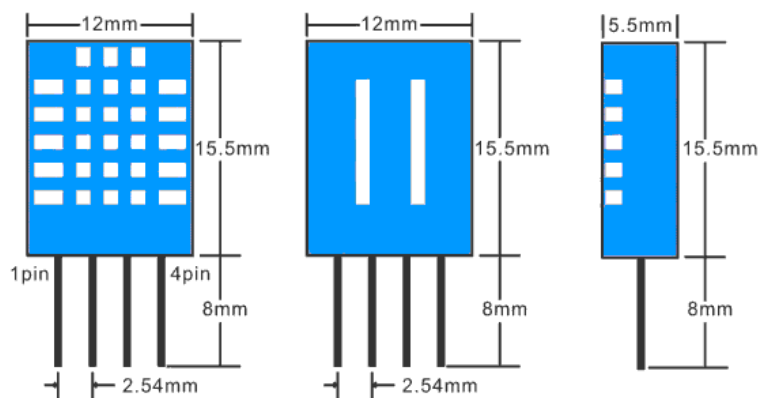


FIG 11 Dimension of DHT11 Sensor

Raspberry Pi 0: 65mm \* 30mm, as shown below in Figure 12

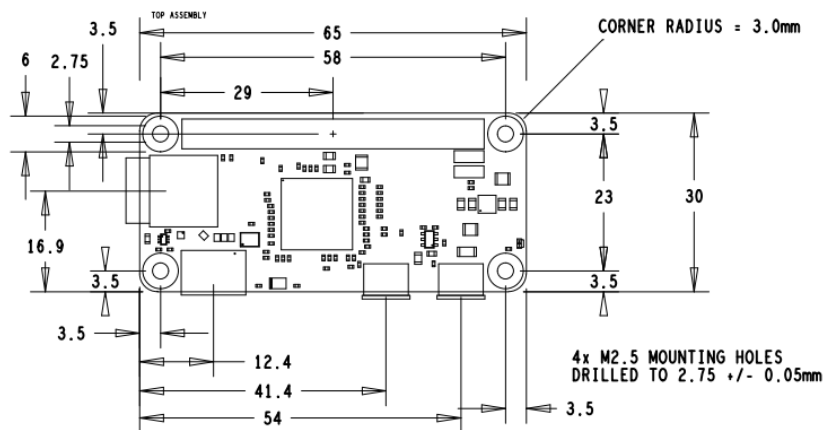


FIG 12 Dimension of Raspberry Pi 0

Quarter-sized Breadboard: 44mm \* 55mm

Box: 111.25mm \* 82.55mm \* 35.56mm.

After choosing the components, we started to package them in the box. To mention that, we need to know that these sensors should be placed outside of the box, since the PIR motion sensor needs to directly sense the motions using its lenses, and temperature sensor needs to avoid the heat caused by the components in the box. Therefore, we still need to use long wires to make them separate from the board, instead of attaching on the board.

For these two sensors, they need to be not close to each other, so that the wires would not be interacted with each other. Also, the humidity and temperature sensor would be placed to the top or the bottom, so it would not be “standing” out of the device and easily be snapped. For the power supply, due to the same reason, the plugs have to be at the back of the box, so the whole box could be steadily attached and plug in to the wall. Since the charger would take almost half of the box size, it also needs to be placed on the other half of the box in opposite of the Raspberry Pi, board and sensors. Therefore, Figure 13 is our original design

of placing:

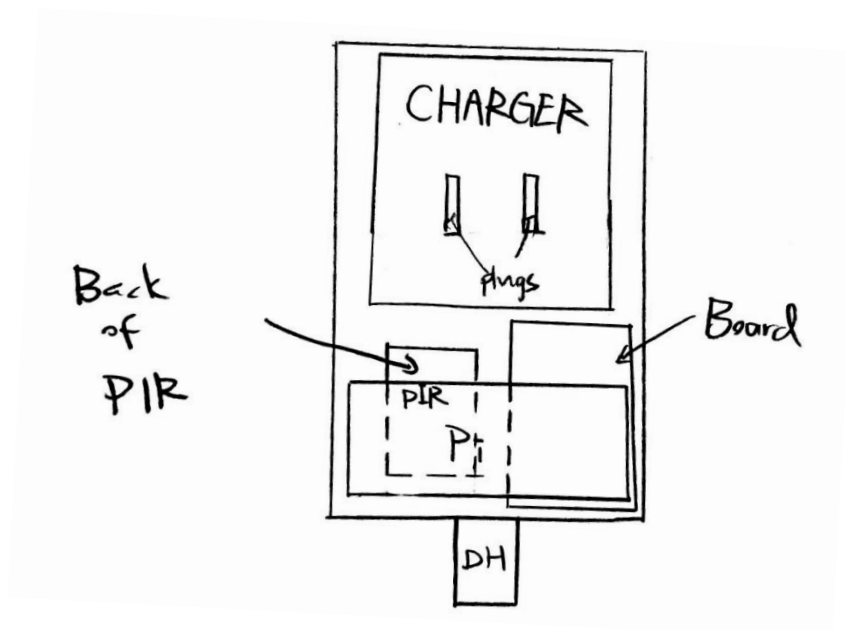


FIG 13 Design of Placing of Components

After that, we decided to make holes for sensors and charger plugs. For the PIR motion sensor, I need to drill a relatively large hole for its lens. The best way of doing that is to use a mechanical punch. Mechanical punch could allow us to drill a hole with diameter of 1 inch, which is approximately the diameter of the lens of the PIR sensor. Below in the Figure 14 is the hole we slugged out:

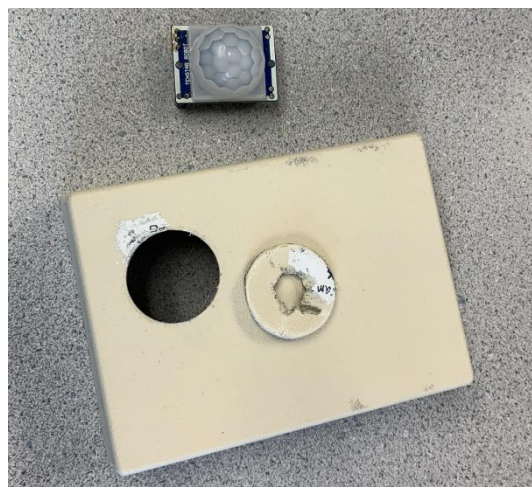


FIG 14 Hole Drilled by Mechanical Punch

For the humidity and temperature Sensor, we would place it at the bottom of the box, therefore the sensor would be placed outside of the box and the pin foot would be stuck into the box and wired to the Proto board. In order to drill four tiny holes for the Humidity & Temperature Sensor, we used 1mm drill bits. Avoid of drifting away the drill bits during the drilling, we would use knife, rasp or screw driver to cut four pre-drill holes before drilling it. Finally, the holes would eventually look like it in the Figure 15:



FIG 15 Shape of Hole Drilled by 1mm Drill Bit

After that, we would wire them up to the Proto Board. For these pins, we did not want to directly expose them out, so we would need to use core wire to wrap them up. Here was the trick we used: we firstly cut a section of a hot shrink tube to the length of slightly shorter than the pin foot, then put it on the foot. After, we soldered the core wire to the pin foot, then moved that section of hot shrink to the place we connected the pin foot and the wire to cover them up, then used a heat gun to heat it, so the shrink tube would be shrunk to wrap them up. We did this to all the other pin foots.



Then, we would move on to the power supply plugs. The power supply needs to be attached to the back of the box from inside, so the plugs can be standing out enough to be plugged into the power. Here, we measured the position and size of the power supply plugs, marked them on the back of the box, then used rasp to cut holes for the plugs, as shown below. Similarly, we used hot glue here to attach the power supply to the back of the box, and it turned out the height of the plugs were good enough to be plugged in. Another thing to mention was, the plugs could not be loosely standing, or it would easily affect the reliability of the device. Again, we used hot glue to fix the plugs so it won't loosely lie down. Also, as shown below in the Figures 16, there is a window that can cover the back up. The window can always be removed to do more customizations. Figure 18 shows the hole we cut for the plugs.

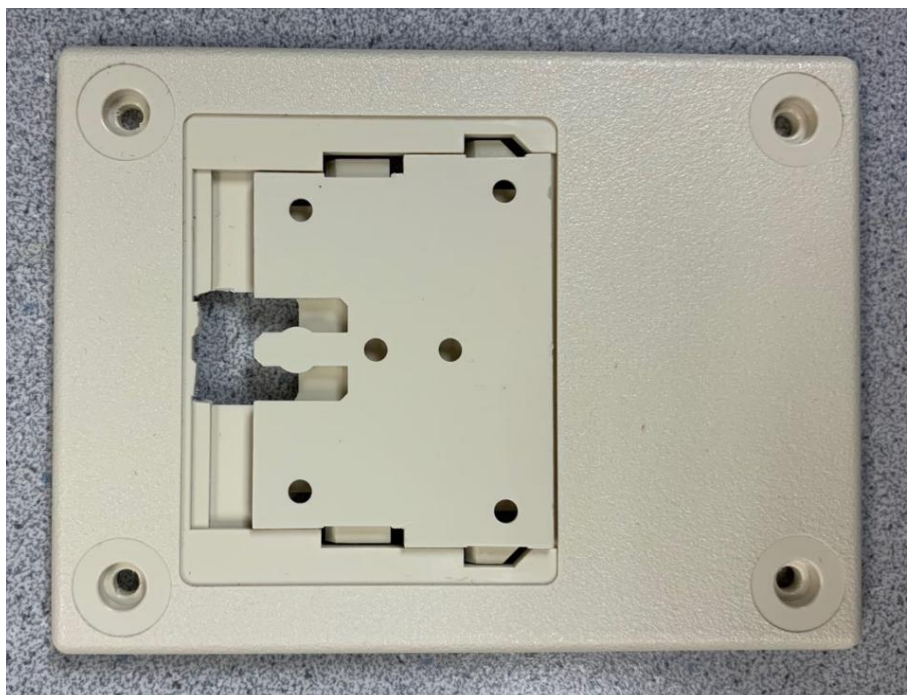


FIG 16 Back of the Box (With Window)

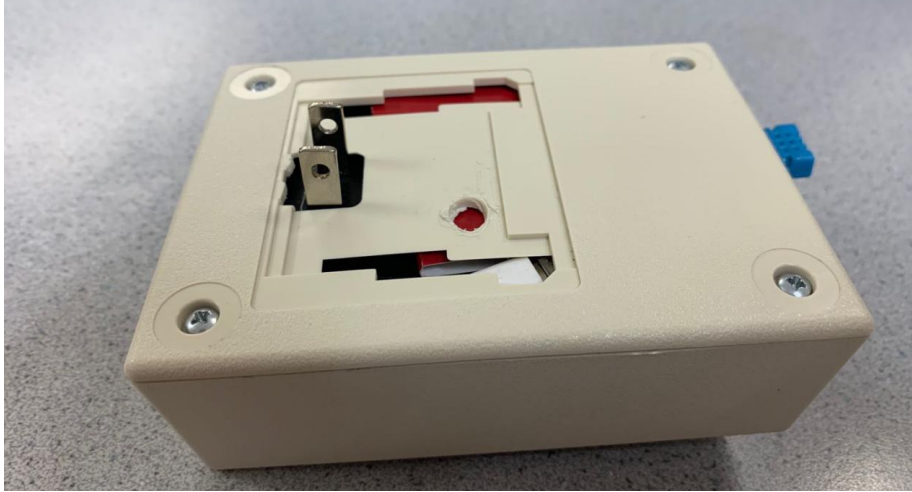


FIG 17 Back of the Box (Without Window)

After setting up the sensors, we need to place the Pi 0, Proto board and the power supply. As stated before, the power supply as well as its cable took almost half of the space of the box and the cable of the power supply too strong that its position could not be easily changed.

For our first try, we put the board and the Pi 0 on the two sides of the port, as shown below in the Figure 18, as our first version of packaged prototype. However, that suddenly caused a problem that these two had to be far away from each other, and the wires for both two sensors could not get to the board, and even the wires between the board and the Pi 0 were easily be snapped. Therefore, we had to abandon this plan.

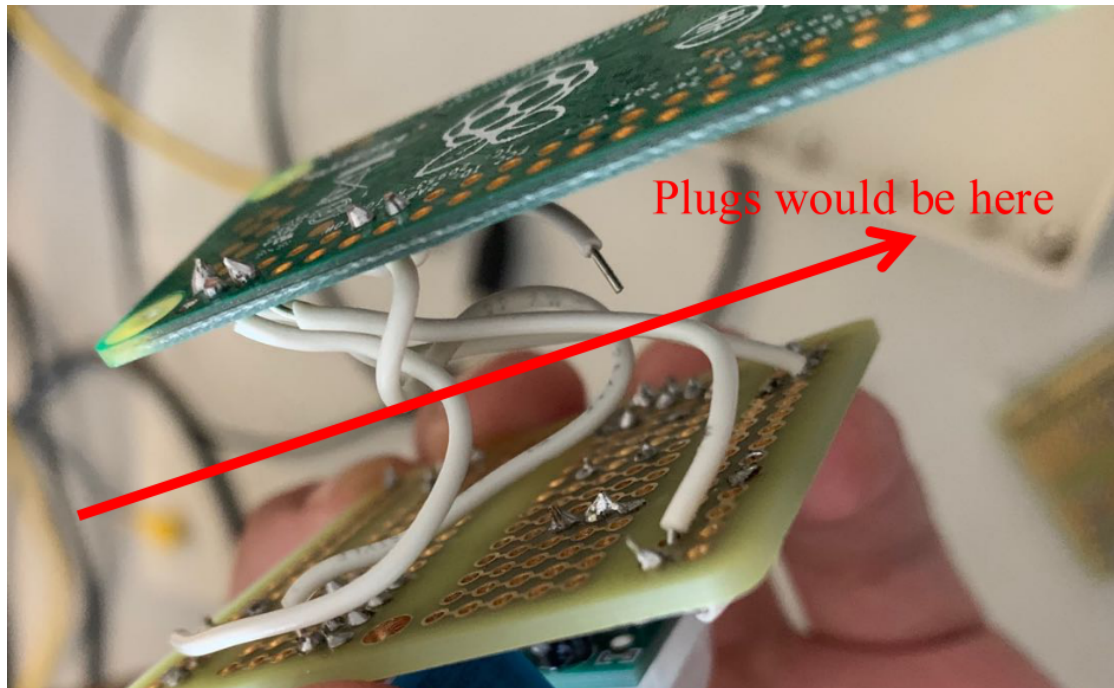


FIG 18 One Way of Placing the Board and Pi 0

For the second try as shown in the Figure 19, we move both the board and the Pi 0 to the same side (near the plugs side), so it would give enough space for the wiring of the two sensors, and it also won't bring much heat to the temperature sensor. Also, we used a paper as an insulator to separate them, so the wires and the components won't be short-cut.

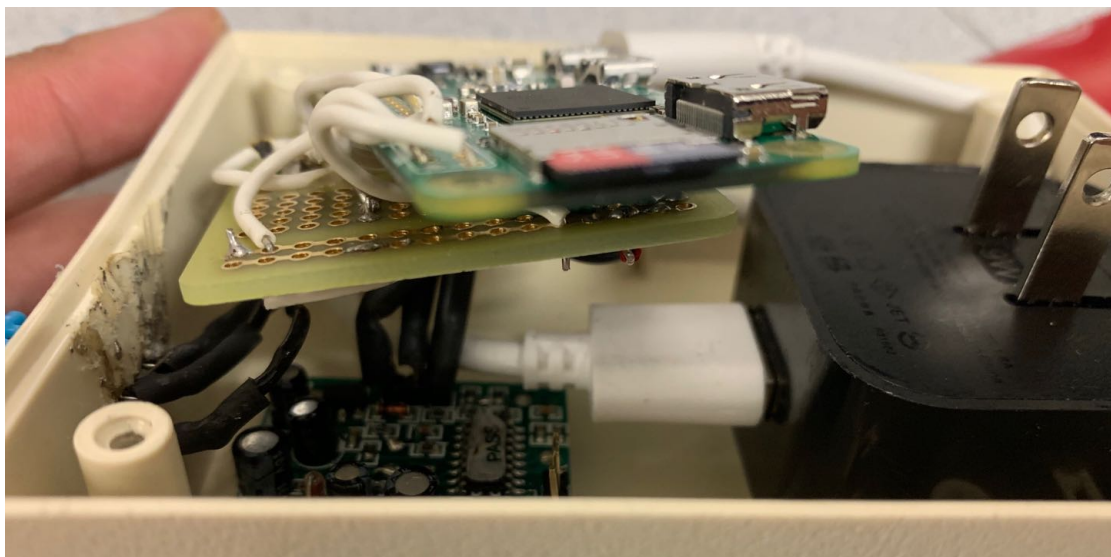


FIG 19 Second Way of Placing the Board and Pi 0

Finally, they could be all put into the box, and worked as an entire packaged device. As mentioned before, this part required all team members to actively communicate with each other and constantly changed the design and the work we have done: specifically, for example, when we switched to the plan that the board and the Pi 0 would be at the same side of the charger instead of on the two sides, all these wires need to be re soldered at the hardware part, also the pin number would be probably changed at the software part. All these wires and pins needed to be redesigned and redone at every step. Below is the final product we have made.

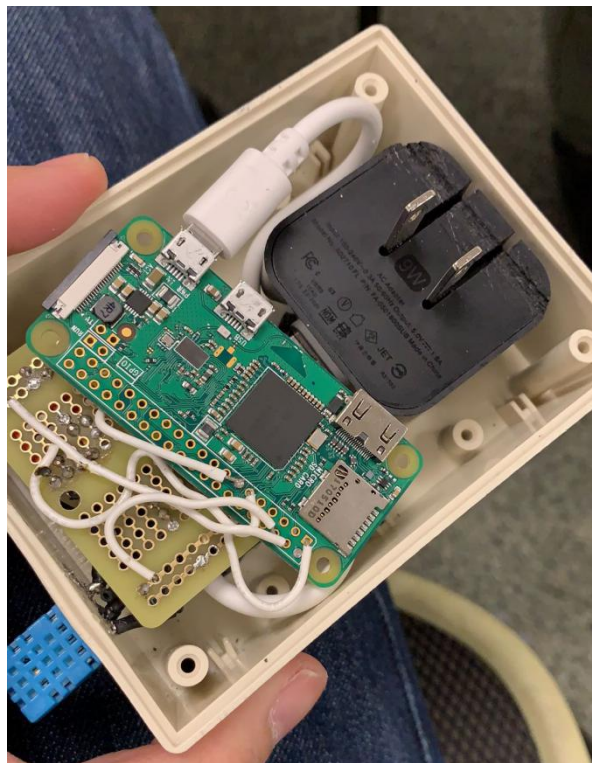


FIG 20 Interior of the Device



FIG 21 Appearance of the Device

### 3. Issues

#### 3.1 Software Part Issues

##### **Issue: temperature and humidity sensor read zero value sometimes**

The DHT11 detecting program sometimes will return zero values. For the reason why the detecting program of standard library read zero value sometime, we check its source code which list below:

```
if t > -100.0 and t <150.0 and hum >= 0.0 and hum<=100.0:
```

```
    return [t, hum]
```

```
else:
```

```
    return [float('nan'),float('nan')]
```

From the source code, we see that it may return nan when the value sensor read is out of range. For the reason why DHT11 may read an out of range value, we search it on the Internet and got the theory that the probability of encountering a bad reading increases as

time passes and that's why people need to design a filter system. For DHT11, its communication protocol is 1-wire protocol which has a very high requirement of the timing sequence and Python is not very good to deal with such situation. In ECE5725, we do the experiment and find that when the required output frequency is about 10 kHz, there would be 5% to 10% frequency error sometimes if we write in python code and from the 1-wire protocol, 1 lasts about 120 $\mu$ s and 0 lasts 76 $\mu$ s, whose frequency is approximate to the 10kHz. Then we got the reason why it may read some erratic values, first, it is due to the python program do not have as good performance as other static language like C. Secondly, it may due to the environment noise.

To solve this problem, we have two solutions, one is to rewrite the whole sensor part with C code and use python to integrate them with other parts. The second solution is that we can change the DHT11 detecting program. For solution 1, it is a lot of work to rewrite almost the whole project and we may also need to design a filtering system for the C program. Considering that situation does not happen frequently, we choose solution 2 and we only need to change one line of code to make the detecting function recursive. If the collecting data is out of range, we can call the detecting function recursively instead of returning zero values.

## **3.2 Hardware Part Issues**

### **Issue: Whether to use PCB board**

Initially, we planned to use PCB design in hardware part. However, after finished quarter-sized breadboard design, it turned out that breadboard with core wires were definitely good

enough to work, and much easier to be put into mass production in future as desired. Therefore, we decided to abandon the PCB design idea.

### **Issue: How to connect sensors directly to Raspberry Pi**

The purpose of our project is to connect sensors directly to Raspberry Pi, so that when Raspberry Pi detects something wrong, it can send a warning email to user directly instead of going out to the cloud for information sharing and control. We connected sensors to Raspberry Pi through GPIO. We need to check whether the desired pin is free or not, the specific function for several GPIO pin, and so on.

### **Issue: How to place two sensors properly in one board**

In addition to wire two sensors and let them work, we should also consider how to place them properly in one board in order to make it convenient for mechanical design. We designed several times and used several boards until we got the final product.

## **3.3 Mechanical Part Issues**

### **Issue: Header on Raspberry Pi 0**

We started with the Raspberry Pi 0 W H version, where H means the header is installed by factory. Though we would not use it as the final prototype, this Header version was way easier to build and change the design if needed. Therefore, we chose to build the device on the Header version, then switch to the original version.

**Issue: Wiring placing**

The wiring between the board, the sensors and the Pi 0 is critical. We had to design strategically about where to put the wires and how long should the wires be. We started that by placing wires on different quadrant of the breadboard before designing the place these components should be by their components. Then, we worked on the hardware part.

**Issue: Electrical insulation**

We used a name card to separate components inside, since they were good enough as an insulator, and the heat caused by device would not achieve the burning point of it.

**Issue: Cable of Power Supply**

In the United States, almost all the cables is at least 25 cm, which is way longer than what we needed. In order to do that, we brought a mini 10 cm cable from China before this semester, which was almost the shortest cable available all over the world.

**Issue: Hot Glue**

Instead of using the tape, we used hot glue to attach all these components together, since hot glue was strong enough to hold things together, but really easy to redo the design (it melted after heated again).

**Issue: Power supply alternatives**

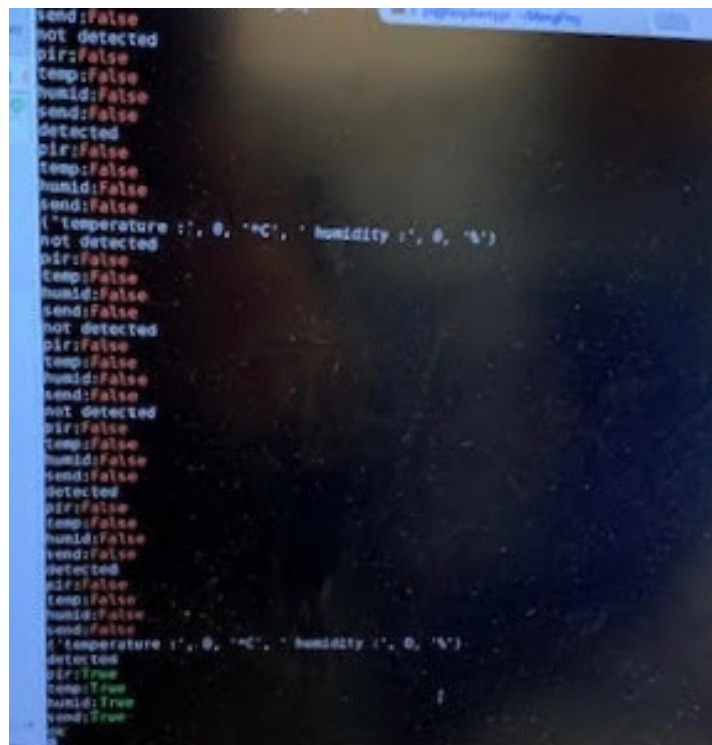
Though we considered other power supply, even a self-made or modified power supply. However, that does not increase the complexity of this project, but more importantly,



dramatically reduce the possibility of customization and mass production of it. For example, every time this power supply failed, we need to redo it again. Also, we need to make as many as them if we want to build multiple of them, instead of buying them online.

## 4. Results

After connecting all sensors directly to Raspberry Pi, we need to test whether they can work or not. Here is our test result:



```
send:False
not detected
pir:False
temp:False
humid:False
send:False
detected
pir:False
temp:False
humid:False
send:False
{"temperature": 0, "humidity": 0}
not detected
pir:False
temp:False
humid:False
send:False
not detected
pir:False
temp:False
humid:False
send:False
not detected
pir:False
temp:False
humid:False
send:False
detected
pir:False
temp:False
humid:False
send:False
detected
pir:False
temp:False
humid:False
send:False
{"temperature": 0, "humidity": 0}
detected
pir:True
temp:True
humid:True
send:True
```

FIG 22 Test result for sensors

After that, we need to integrate software part and hardware part and package it. Then, we plug it and make it work for one week.



FIG 23 Make this system work for a week

We connect it from PC via SSH and the product is worked at the sensing mode. When the product detect something unusual, it would send users an alarm email, which is shown as below:

# Warning!! Unusual Situation Reported!!

Inbox



Land Guardian Mar 26

to me ▾



Dear Customer,

According to data from sensors: the current room temperature is 24 Celsius compared to our preset reference 10-34 Celsius. The room humidity is 5% compared to our preset reference 0-60%. We have detected unusual motion in your house.

We highly suggest you check any potential hazards in your house such as: water outlets, gas, air conditions, windows, doors, etc. If necessary, please contact emergency services as soon as possible.

We are trying our best to make sure your house is safe, and we will keep monitoring your house!

Yours,  
Land Guardian

FIG 24 Alarm email for home security system

We originally set that if the temperature is above 50 degree Celsius or below 15 degree Celsius, then the alarm email will be sent to residents. If the humidity is above 60% or below 25%, then the alarm email will be sent to residents. If the PIR motion sensor detects people's motion, then the alarm email will also send to residence. During our one week's test, we find the test result meets our planning.

Besides, we originally set the interval of two alarm emails to be sent is at least 5 minutes. Since our equipment is used in the case of when the owner is not in the house, our test environment is placed in the doorway of our teammate Haodong Ping's room. For PIR

motion sensor, in our one-week's test, the time alarm email was sent matches the time of Haodong Ping's school time and home time. The alarm email was also sent when Haodong Ping's roommate visited. For the temperature and humidity sensor, since the temperature and humidity in the normal bedroom are basically in a comfortable range, no alarm email is sent during our one-week test.

In order to test the temperature and humidity alarm capability, we set the maximum allowable temperature in the room to 8 degrees Celsius, and the lowest temperature is 1 degree Celsius. Then we tested for 2 hours, and we found that it sent a total of 7 emails. It indicates that when the user receives the alarm email, it takes about 15 minutes to process the emergent situations at home. In our opinion, this time interval is good and enough for residents to process the emergent situations at home. If the time interval is too short, the user may receive too many duplicate emails. On the contrary, if the time interval is too long, it may not be able to effectively detect whether there is some stealing events happen at the same time.

In addition to all above, we also check the situation when the WiFi is off. We powered off for some time and test when someone passed by the equipment, whether the alarm email would sent or not. The result is that the alarm email would not sent.

Generally speaking, our equipment's one-week test result meets our expectation. However, we also find something to improve and we will perfect our equipment in the future.

## **5. Future Work**

In the future, we plan to replace the charger with power bank, so that you can use this product in anywhere you like. Now, since the only way to power this system is to plug the charger, it can only use in a fixed place. What's more, we want to add more sensors. At this time, we only connect PIR motion sensor and humidity and temperature sensor, which can only obtain motion, humidity and temperature information. We plan to use more sensors to obtain more information, like air quality, smoke, and so on. Besides, we cost nearly \$20 for this product. However, in order to use it widely, we still need to reduce cost.

## **6. Conclusion**

The traditional home security system is based on the cloud server, all data collected from user's home by sensor would be uploaded to cloud and processed in it. What's more, the users cannot change anything about processing the data. To deal with these problems, in this project, the Raspberry Pi is used as the server to replace the cloud server. In the designed home security system, all the sensors, including PIR sensor, temperature sensor and humidity sensor will be connected to Pi directly and Pi will be worked as server to process all the data collected by sensors. When Pi detect something wrong, it can send a warning email to user directly.

Team members finished the prototype of this equipment in Raspberry Pi 3. Then, team members translated the code from Raspberry Pi 3 to Raspberry Pi 0 in order to reduce cost. Then the software and hardware part were integrated. Finally, according to the circuit size,

team members designed the container. After plugging in and working for one week, we proved that this equipment is successful for using.

# Appendix

## 1. Reference

- 1) <https://www.mbtechworks.com/projects/pir-motion-sensor-with-raspberry-pi.html>
- 2) <http://www.circuitbasics.com/how-to-set-up-the-dht11-humidity-sensor-on-the-raspberry-pi/>
- 3) <https://www.digikey.com/product-detail/en/serpac/032WAL/SR032-WA-ND/304170>
- 4) <https://www.adafruit.com/product/589>
- 5) <https://components101.com/hc-sr501-pir-sensor>
- 6) <https://www.raspberrypi-spy.co.uk/2017/09/dht11-temperature-and-humidity-sensor-raspberry-pi/>
- 7) [http://courses.ece.cornell.edu/ece5990/ECE5725\\_Fall2018\\_Projects/lab\\_hp394\\_mf734/index.html](http://courses.ece.cornell.edu/ece5990/ECE5725_Fall2018_Projects/lab_hp394_mf734/index.html)
- 8) <http://robocraft.ru/files/datasheet/DHT11.pdf>

## 2. Code

### pir.py:

```
import RPi.GPIO as GPIO

import time

class pir():

    def __init__(self,pir):

        self.pir = pir

        GPIO.setmode(GPIO.BCM)

        GPIO.setup(pir, GPIO.IN)
```

```
time.sleep(2)
```

```
def detect(self):
```

```
    if GPIO.input(self.pir):
```

```
        time.sleep(1.5)
```

```
        if GPIO.input(self.pir):
```

```
            return True;
```

```
    return False;
```

### **temp\_detect.py:**

```
import RPi.GPIO as GPIO
```

```
import time
```

```
class temp_hum_sensor():
```

```
    def __init__(self,channel):
```

```
        self.channel = channel
```

```
    def collect(self):
```

```
        THdata = []
```

```
        channel = 13
```

```
        data = []
```

```
        GPIO.setmode(GPIO.BCM)
```

```
        time.sleep(2)
```

```
        GPIO.setup(self.channel, GPIO.OUT)
```

```
        GPIO.output(self.channel, GPIO.LOW)
```



```

time.sleep(0.02)

GPIO.output(self.channel, GPIO.HIGH)

GPIO.setup(self.channel, GPIO.IN)

while GPIO.input(self.channel) == GPIO.LOW:

    continue

while GPIO.input(self.channel) == GPIO.HIGH:

    continue

j = 0

while j < 40:

    k = 0

    while GPIO.input(self.channel) == GPIO.LOW:

        continue

    while GPIO.input(self.channel) == GPIO.HIGH:

        k += 1

        if k > 100:

            break

    if k < 8:

        data.append(0)

    else:

        data.append(1)

    j += 1

# print("sensor is working.")

```

```

# print(data)

humidity_bit = data[0:8]

humidity_point_bit = data[8:16]

temperature_bit = data[16:24]

temperature_point_bit = data[24:32]

check_bit = data[32:40]

humidity = 0

humidity_point = 0

temperature = 0

temperature_point = 0

check = 0

for i in range(8):

    humidity += humidity_bit[i] * 2 ** (7 - i)

    humidity_point += humidity_point_bit[i] * 2 ** (7 - i)

    temperature += temperature_bit[i] * 2 ** (7 - i)

    temperature_point += temperature_point_bit[i] * 2 ** (7 - i)

    check += check_bit[i] * 2 ** (7 - i)

tmp = humidity + humidity_point + temperature + temperature_point

if check == tmp:

    print("temperature :", temperature, "*C", " humidity :", humidity, "%")

    THdata.append(temperature)

    THdata.append(humidity)

return THdata

```

```
else:
    # print("wrong")
    time.sleep(1)
    return self.collect()
```

### **sendemail.py:**

```
#coding:utf-8
import smtplib #load smtplib
from email.mime.text import MIMEText
from email.utils import formataddr

class sendEmail():
    def __init__(self,user):
        self.user = user

    def mail(self,temp,temp_ref,hum,hum_ref,pir):
        ret=True
        try:
            det = 'not '
            if pir == True:
                det = "

                content = 'Dear Customer,\n According to data from sensors: the current room
temperature is '+str(temp)+' Celsius compared to our preset reference '+str(temp_ref)+'
Celsius. The room humidity is '+str(hum) +'%' compared to our preset reference
```

```
'+str(hum_ref)+'%. We have '+det+'detected unusual motion in your house.\n We highly suggest you check any potential hazards in your house such as: water outlets, gas, air conditions, windows, doors, etc. If necessary, please contact emergency services as soon as possible.\n We are trying our best to make sure your house is safe, and we will keep monitoring your house!\n\nYours,\nLand Guardian'
```

```
#print content
```

```
msg=MIMEText(content,'plain','utf-8')
```

```
msg['From']=formataddr(["Land Guardian",'pgxapply1@sina.com'])
```

```
msg['To']=formataddr(["AWater",self.user])
```

```
msg['Subject']="Warning!! Unusual Situation Reported!! " #title of the mail
```

```
server=smtplib.SMTP("smtp.sina.com",25)
```

```
server.login('pgxapply1@sina.com',"941218")
```

```
server.sendmail('pgxapply1@sina.com',[self.user,],msg.as_string())
```

```
server.quit()
```

```
except Exception:
```

```
ret=False
```

```
return ret
```

**main.py:**

```
import temp_detect
import pir
import sendemail
import time
import threading
import RPi.GPIO as GPIO

def init_sys():
    #declare the sensor, mail system and reference value

    global ths

    global p

    global se

    global sendflag

    global temp_max_ref

    global temp_min_ref

    global humid_max_ref

    global humid_min_ref

    #initialize the detect value

    global pir_detect

    pir_detect = False

    global temp

    temp = -300

    global humid

    humid = -1
```

```
#initialize the whole system, including the sensor, mail system and reference value
```

```
ths = temp_detect.temp_hum_sensor(13)
```

```
p = pir.pir(19)
```

```
se = sendemail.sendEmail('mf734@cornell.edu')
```

```
temp_max_ref = 50
```

```
temp_min_ref = 15
```

```
humid_max_ref = 60
```

```
humid_min_ref = 25
```

```
sendflag = False
```

```
def pir_work():
```

```
    global pir_detect
```

```
    global pir_count
```

```
    global sendflag
```

```
    pir_count = 0
```

```
    while True:
```

```
        # print(str(count))
```

```
        time.sleep(1.5)
```

```
        pir_detect = p.detect()
```

```
        if pir_detect and pir_count < 3 :
```

```
            #print("detected")
```

```
            pir_count += 1
```

```
    elif pir_detect and pir_count >= 3 and not sendflag:
```

```
#print("detected")
```

```
sendflag = True
```

```
else:
```

```
#print("not detected")
```

```
pir_count = 0
```

```
def temp_hum_work():
```

```
    global ths
```

```
    global temp
```

```
    global humid
```

```
    while True:
```

```
        # print(str(count))
```

```
        THdata = ths.collect()
```

```
        temp = THdata[0]
```

```
        humid = THdata[1]
```

```
        print("temp:"+str(temp))
```

```
        print("humid:"+str(humid))
```

```
        time.sleep(3)
```

```
init_sys()
```





```
global humid_min_ref
```

```
pir_flag = False
```

```
temp_flag = False
```

```
humid_flag = False
```

```
try:
```

```
while True:
```

```
    if ( count%2 == 0 ):
```

```
        THData = ths.collect()
```

```
        count = 0
```

```
        temp = THData[0]
```

```
        humid = THData[1]
```

```
        if ( temp >= temp_max_ref or temp <= temp_min_ref ) and temp_count < 2:
```

```
            temp_count += 1
```

```
            elif ( temp >= temp_max_ref or temp <= temp_min_ref ) and temp_count >= 2
```

```
and not temp_flag:
```

```
    temp_flag = True
```

```
else:
```

```
    temp_count = 0
```

```
        if ( humid >= humid_max_ref or humid <= humid_min_ref ) and humid_count
```

< 2:

```
    humid_count += 1
```

```
    elif ( humid >= humid_max_ref or humid <= humid_min_ref ) and
```

humid\_count >= 2 and not humid\_flag:

```
        humid_flag = True
```

```
    else:
```

```
        humid_count = 0
```

```
pir_detect = p.detect()
```

```
if pir_detect and pir_count < 2 :
```

```
    print("detected")
```

```
    pir_count += 1
```

```
        elif pir_detect and pir_count >= 2 and not pir_flag:
```

```
            print("detected")
```

```
            pir_flag = True
```

```
        else:
```

```
            print("not detected")
```

```
            pir_count = 0
```

```
time.sleep(1.5)
```

```
count += 1
```

```
'''
```

```
if( temp_max_count >= 3 and temp > temp_max_ref):
```

```
    print '1'
```

```

elif( temp_max_count < 3 and temp > temp_max_ref ):
    temp_max_count += 1

elif( temp_min_count >= 3 and temp < temp_min_ref ):
    print '1'

elif( temp_min_count < 3 and temp < temp_min_ref ):
    temp_min_count += 1

elif( temp < temp_max_ref and temp > temp_min_ref ):
    temp_min_count = 0
    temp_max_count = 0
    ""
    sendflag = pir_flag or temp_flag or humid_flag
    print("pir:"+str(pir_flag))
    print("temp:"+str(temp_flag))
    print("humid:"+str(humid_flag))
    print("send:"+str(sendflag))

if(sendflag):
    #print("hello,huamiaomiao")
    ret = se.mail(temp,50,humid,60,pir_detect)
    if(ret):
        print "ok"
    else:
        print "fail"

```

```
        time.sleep(7200)

        sendflag = False

'''

while True:

    data = ths.collect();

    if p.detect():

        print("detected")

    else :

        print("not detected")

    time.sleep(3)

'''

except Exception:

    GPIO.cleanup()

#se.mail(21,4,31,60,True);
```