UVA Intelligent Seat Detecting System

Hans Zhang, Tim Meng, Mike Wang, Yingxiang Sun,

12/12/2017

Capstone Design ECE 4440 / ECE4991

Signatures

Hans Zhang

[illegible]

[illegible]

Timothy Meng

Statement of work:

Hans Zhang:

I was in charge of developing the whole web application, including building server, configuring URLs, constructing Model-View-Controller (MVC)[13] architecture, designing visual user interface, enhancing website performance and deploying the site to the public.

For the web application MVC constructions, I used Django framework[14]. Furthermore, for the front-end development, I employed AJAX[15] for refreshing table figures and bootstrap 3.0[16] for styling. Lastly, for the application deployment, I used Heroku[17] as the platform and the whole deployment processes were done through command line instructions.

Timothy Meng:

The major part of the project that I was responsible for is building a consistent connection between the MQTT public broker and the client side webpage. To be specific, after the seat occupancy data derived from the CC3200 and the thermal is transmitted to the public broker, the scripts that I write brings the data to the webpage in order to be displayed.

In addition, I was also involved in integrating the data receiving scripts with the Django web frame and the deployment of the website to be viewed on any computer. Since the webpage is always running, my scripts will run every time the web page is refreshed. Lastly, I was also involved in the incremental testing of the individual components and the holistic testing of the final product.

Mike Wang:

I was responsible for all the hardware selection, design, and testing. Hardware selection includes selections for the CPU, thermal sensor, batteries, logic switch, various connectors and etc. I designed the PCB board as a interconnect, 3D printed case and BJT power saving switch (ended up not including in the final product). Testing includes both hardware (testing of the sensor, logic switch, sensor-CPU connection, PCB board connection and 3D printed case compatibility) and software (I2C communication code, human detection algorithm, Wi-Fi connectivity, and website display). Along with the hardware, I was responsible for I2C communication coding, ordering parts, reimbursement process and organizing team folder on the collab. I also helped Yingxiang with some of the human detection codings, especially with coming up with ideas to compensate for changing room temperature.

Yingxiang Sun:

I was responsible for transferring raw data from thermal sensor to public broker after analysis. The first task was to collect data from the thermal sensor to CC3200 launch board. After that, I developed a human detection algorithm by manipulating temperature data collected from the thermal sensor. Then a Boolean signal which represents if people were detected or not and the device serial number would be sent to the public broker through MQTT protocol. CC3200 Launchpad sends MQTT datagram every second to guarantee accuracy and timeliness of data.

Lastly, I made up test plan for this section of the datapath, also participated test for system integration.

Table of Contents

Contents

Table of Figures

Abstract

The Vacant Seat Detector System (VSDS) is an urgent call of needs. It should allow any student, walking into Rice Hall, to know where the vacant seats are on a webpage or a mobile App. As the idea of the goodness of Internet of Things (IoT), "a system creating opportunities for more direct integration of the physical world into computer-based systems" [1], has been brought into public focus, our VSDS helps to convert this new concept into a tangible tool that eases students' lives. The system acts as a distributed sensing system: when the distributed device is fed with an infrared signal (a person sit in a designated area); the web APP would display that area as "occupied" in real-time. The end product of this project will be composed of distributed units, each containing an IR sensor, a CC3200 Wi-Fi enabled processor, a central web server running on AWS EC2 and a web user interfaces written in Django framework.

Background

Anyone who has studied at Rice Hall previously knows that it is a huge pain going through all the floors to find an available study spot. To resolve this pain point, my team has decided to implement a seat detection system [11] for Rice Hall, so any student who comes to find a study spot can know where the available places are without wandering through all the floor space of the building. For example, occupancy sensors are installed and widely used in modern parking lots for shopping malls (US. Department of Energy) [8]. To cope with the accompanying high traffic volume, the vacant spot detection system informs the customers how many and where the vacant parking spaces exist. Another example is the activation of airbags on passenger seats in cars(Becker) [9]. When the seats do not smartly detect any passenger, the airbag is automatically deactivated in order to reduce the meaningless cost of an accident.

One unique area of our project is that our detection system will be the first that incorporate occupancy sensors and internet server connection to enable users easily access the vacancy status of seats for their convenience. Another unique area of our occupancy sensor is that it involves the Mixed-Signal Microcontroller 430 and its Wi-Fi booster, which allows us to connect the device to internet via MQTT and send data to be processed onto the server(Sinha) [10].

Our team aims to bring a similar people-detecting system to Rice Hall by uniquely utilizing knowledge gained from previous courses, such as Web Technologies from CS3240 for displaying results of IR sensor data, low-level C programming in Code Composer Studio learned in ECE 3430 Introduction to Embedded Computer Systems, and the critical circuit building and designing experience gained from

Fundamental I, II and III. Aside from those tools introduced in past classes, we will also do researches on CAD for 3-D printing cases and parallel programming for running scripts on web pages.

Constraints (Design, Economic and Cost Constraints)

One major constraint area is privacy for the users, and it can be grouped into two sets issues. One is that by constantly displaying online the status of occupancy for seats in Rice Hall renders limited privacy protection for the users. In addition, this also proposes safety concerns, for it exposes occupancy information. In addition, the data collected from the sensors can be further processed and extracted to analyze usage patterns to minimize energy consumption. The protocols that we set for our device must be on par with the IEEE standards in order to connect to Wi-Fi. Another constraint is that if one is sitting still in the seat over a long period of time, our sensor may not pick up on that. Therefore, this is an issue that our software needs to check for and resolve by performing different analysis on the returned data.

The installation of our end product should be simple for users who have related basic product knowledge because the connections between the necessary components are already done in the backend. The installation is a one-time process, indicating that after the initial installment, users do not have to worry about another installment during the entire lifespan of the system. After thorough research, we have confirmed the availability of the manufacture components that we would be using. The end product usage is suitable for occupancy detection under various settings. Also, the final size of the end product will not minimize to the full extent due to the lack of advanced manufacture ability. To massively reproduce the prototype and launch to the market, the end product would require partnership with industry level manufacture company to properly design, test, and the market for the finished product.

Due to economic constraints, the purchased manufactured parts are coming from different companies with different functionalities, rather than a fully assembled device packaged under industrial standards. The accompanying result is that the performance may be compromised. In addition, because we lack the ability to produce a size-minimized device, the size of the inserted battery is also limited. The environmental impact coming from the IR sensors and the CC3200 CPU is minimal, mostly coming from the disposal of the manufactured components afterward. For example, if the batteries and other plastic-composite material are disposed of incorrectly, potential environmental damage may be inflicted. The sustainability of our project is directly dependent on the lifespan and battery usage of the manufactured components. The less energy that our device uses and the longer its lifespan is, the more sustainable our end product will. Therefore, when we design the device, using less energy is definitely a major goal. The software produced can always be reused and recycled, with minimal maintenance required. As long as the entire system does not have to be replaced or if the online server is not removed, the software that we produced will always be in effect. The vacant seat detector produced should not have any major health concerns because its basic form is just an IR sensor emitting infrared signals. The output signal should be relatively weaker compared to other emitting device signals. The mass reproduction of the system is definitely foreseeable; the related variables will be the market demands and the technology supply trends. There is already a great demand for occupancy detectors in the market, and our product offers a fully integrated solution to users and designers of the occupancy detectors saving them time and bring convenience. Ethically, the end product should bring much societal benefits and impact. Specifically, the device will both add to the technology advancement and improve quality of life by bringing convenience and liberty.

External Standards

Hardware

In order to interface the sensor with the CC3200 board, I2C communication protocol [22] has to be followed correctly in the coding implementation in order to establish a connection. In the process of designing the PCB board, trace thickness and other tolerances (min 0.006 line, no overlapping drill hits and a minimum of 0.015-hole size) [23] have to be followed in order to ensure proper functionality.

Software

CC3200 must follow IEEE 802.11[24] and MQTT IoT protocol [25] in order to ensure that Wi-Fi connectivity is established and the correct datagram is passed to the cloud. On the other hand, the web server has to conform to HTTP 1.1 protocol [26] in order to receive an error-free datagram from the public broker (part of the MQTT protocol). In addition, the data transmission from the CC3200 along with the data receiving by the web server, Heroku, strictly follow the IEEE 802.11 wireless LANs.

Tools Employed

Rhinoceros was used to develop the STL file for the 3D printed case. The software is typically used for 3D modeling in both engineering and architecture. Similar to Autodesk Inventor, it has the ability to create a 3D object by creating 2D shapes, extruding and Boolean differencing object in order to design for proper shapes. The STL file was then imported by using the software and the STL file was further converted to a cubepro file by the Cubepro software in order to print the 3D case using the NI Lab 3D printer.

Multisim and Ultiboard were used for simulating the sensor-processor system and designing the interconnect PCB board. Custom parts including the CC3200 board pin layout and logic level shifter were designed in Multisim and exported to Ultiboard. The components were then placed properly on the board and traces were connected following the design rules.

RedoGerBer File was used to create the correct Gerber file for FreeDFM to test the PCB board for production. This file was provided by Professor Harry Powell. The file helped to identify and compress different parts (smb, sst and etc) into a single zip file.

When developing C code on CC3200 launch board, we used Energia as our platform. Energia is an open-source electronics prototyping platform with the goal to bring the Wiring and Arduino framework to the Texas Instruments CC3200 based Launchpad. The Energia IDE is cross-platform and supported on Mac OS, Windows, and Linux. Energia uses the mspgcc compiler by Peter Bigot and is based on the Wiring and Arduino framework. Energia includes an integrated development environment (IDE) that is based on Processing. Energia is also a portable framework/abstraction layer that can be used in other popular IDEs. Utilize a web browser based environment with CCS Cloud at dev.ti.com. Community maintained Energia plug-ins and integrations are available for Xcode, Visual Studio, and Code Composer Studio.

When developing the web application, since we are writing our application in Python, a good Python IDE is desired. As a result, PyCharm became our first choice since it has a great compatibility to different Python interpreters and it has the best support for package installations. When deploying our web application, we used Heroku. Heroku is a cloud platform as a service (PaaS) supporting several programming languages that are used as a web application deployment model which will maintain a log known as the append-only ledger of releases the developer makes. [17] In addition, our application is sent to Heroku using git and GitHub.

When building the connection between the server and the endpoint page web, some of the tools and technologies employed include python, Sublime, MQTT, Paho, and Github. The underlying software code is written in python and text editor that was used was Sublime. In order to set up a stable connection with and receive secure data from the MQTT server, our software incorporates the Eclipse Paho Library. This way, our design is enabled to receive the data that is passed to the free public Hqtt server from the CC3200 board and the thermal sensor.

Ethical, Social, and Economic Concerns

Environmental Impact

The only concern on the environment is that improper recycling or disposing of the battery inside the device. Rechargeable batteries are usually nickel cadmium, nickel metal hydride or lithium ion. Secondary batteries are commonly found in cordless phones, cordless drills, mobile phones, laptops and computers, shavers, digital cameras, video cameras and house alarms. Although rechargeable, secondary batteries may need to be recycled when they no longer hold a charge. Why are Batteries Harmful to the Environment? Environmental problems Batteries are identified as a problem material in the waste stream. Batteries are made from a variety of chemicals to power their reactions. Some of these chemicals, such as nickel and cadmium, are extremely toxic and can cause damage to humans and the environment.

Sustainability

After calculation, our battery power is projected to sustain our sensor and CC3200 boards for about 3 months. In an industry perspective, only four annual system inspection while switching the battery required is considered practical and beyond par.

Health and Safety

Since we are using the thermal sensor as our only sensing device, which is harmless to people when the device in working state. Also, it has a self-sustained and rechargeable battery to support its own energy consumption, so there’s no obvious health and safety problem.

Manufacturability

Our prototype device can definitely be modified and put into mass production and the cost will go down significantly. The thermal sensor costs 52 dollars for one piece but if we’ve purchased it in one hundred quantities, the price will go down to 38 dollars. Also, for the prototype, we used not only the Wi-Fi enabled CC3200 chip but also the Launchpad that comes with it. In fact, we don’t need most of the Launchpad functionalities. For manufacturing purpose, we would like to reduce the size of our device and just use the CC3200 chip, which is 10 dollars compared to the 30 dollar Launchpad. 3D printed case and PCB board also have good manufacturability and the prices of them will go down a lot if they are mass produced. Our current battery solution is to use a commercial battery bank with 5V USB output and in order to improve manufacturability and reduce device size, a custom rechargeable battery that can be used to directly supply the chip and sensor should be chosen. All of the above modifications can be standardized and the device will be suitable for mass production.

Ethical Issues

A primary ethical issue that we are concerned with is the proper usage of the temperature/seat status data. If the device is mass produced and implemented at Rice Hall, every person with internet access will be able to know the seat status of every public seat at Rice Hall at any time. It is obviously beneficial to students and staff but can also cause potential problems if, for example, terrorists are trying to monitor the status of Rice Hall. In this case, terrorists will have free access to the seat status information and have the ability to infer the density of crowd on each floor. Thus it is best to have the information encrypted under UVa sis system and only allow student and faculty access.

Another issue is about privacy. Even though our seat detection system only interprets seat status and users will not be able to tell how many people are around the seats and who they are, the system shares part of the private data with the public users and it could cause concerns for a small group of people. It is best for us to set up a privacy agreement on using the system with the users before users can get access to the system.

Intellectual Property Issues

Closely related to our project are three different patents, all published in the last 15 years. The patents are named, "Image processing occupancy sensor" [19], "Occupancy sensor and method for home automation system" [20], and "Occupancy pattern detection, estimation, and prediction" [21].

The first patent is associated because its purpose is to detect human presence in a particular room utilizing vision. Although the overall goal is the same, this patent involves technologies that not used in our capstone project, but rather discussed in depth. The reasoning that we moved away from using vision to detect occupancy is not only extensive unnecessary exposure of privacy details but also intensive power consumption. The second patent utilizes angle motion sensors to detect occupancy of the household spaces. As mentioned in the claims section, the patent acknowledges the existence of data inaccuracy in a situation that corresponds to what is referred as false-positive. This patent is similar to our project in terms of technology and purpose. However, its main aim is to solve occupancy issues in households in order to trigger automated devices instead of bringing public benefits in larger and more densely-populated buildings. The third patent is different from the previous, where it includes claims and insights on prediction and estimation algorithms involved with occupancy systems. The goal is to allow the machine or the algorithm to calculate and predict more accurate and fitful value thresholds in order to produce better results. In other words, there is an aspect of machine learning to these developing devices. Likewise, the algorithm used within our thermal sensor data analysis process uses similar concepts and produce an environment that allows the algorithm or the machine to learn how to better predict new constants based on its historical data.

Detailed Technical Description of Project

Overview

The Vacant Seat Detector System was developed by four teammates working in a parallel fashion. On the processor and sensor side, Mike (hardware and I2C) devised a way to connect CC3200 processor with the thermal sensor to ensure correct data path while Yingxiang worked on human detection algorithm, Wi-Fi connection and MQTT protocol establishment on the CC3200 processor assuming that the data path is achieved properly. On the software side, Timothy worked on the back-end programming (server and data collection) while Hans realized the front-end of the website (display). Because of the fact that MQTT protocol has the feature that makes the hardware side almost independent of the website side by inserting a broker in the middle of the hardware-website data path, the teammates worked almost entirely in parallel but still ensured testability of the product.

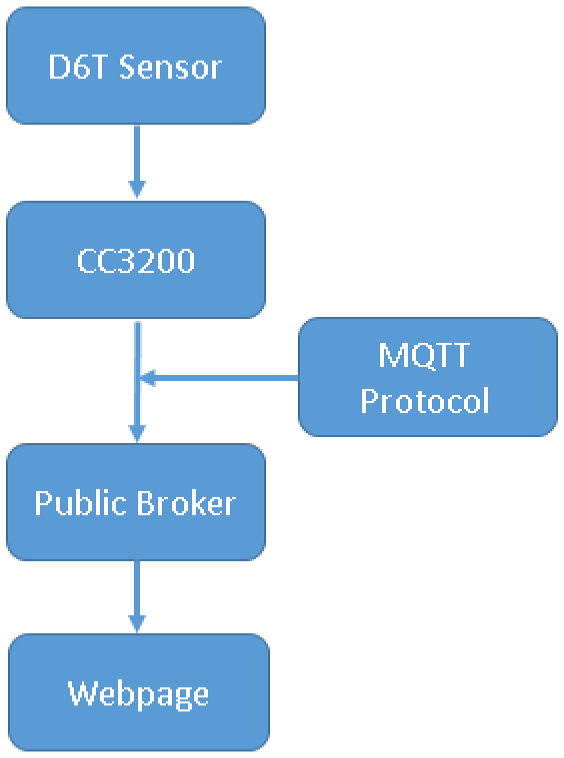


Figure 1: Vacant Seat Detector System Flowchart

Hardware Design

Hardware Choice

In order to detect the presence of human in a reasonable range, instead of using passive infrared sensor, an active thermal sensor was picked to fulfill the requirement since active thermal sensor will be able to consistently output temperature data while a passive infrared sensor that only detects motion fails to detect the presence of an idle human. Specifically, the D6T thermal sensor from OMRON was chosen because it outputs 16 temperature data points in a grid, has excellent sensing range and supports I2C communication. Additionally, in order to connect the D6T sensor to the processor, a jst gh 4 pin connector with wiring harness was chosen to actualize connection.

Our project aims to have a wireless communicating sensing device. Therefore, a Wi-Fi enabled processor was needed to fulfill the requirement. CC3200 Wi-Fi enabled CPU Launchpad was chosen because it supports wireless communication without an extra antenna for signal boost and various public libraries are available online for development.

Voltage Tolerance and Battery Requirements

The input and output of the D6T are both 5V. Thus the CC3200 has to be able to tolerate 5V on its digital read pins and output 5V to supply the sensor. The fact is that CC3200 has a steady 5V output pin on board (only when USB power is used) but input pins all have only 3.3V tolerance. In order to address this voltage tolerance problem, a device that boosts and regulates voltage in two directions needs to be used (I2C communication uses one channel bidirectionally). A simple solution is a logic level shifter and we chose a bidirectional level converter (5-3.3V) from Satisfyelectronics. This specific converter has four channels that can shift logic level between 5V and 3.3V and for our project, two channels were used for the I2C data line and clock line.

Our vision of the sensing device is a stand-alone device powered by the battery with reasonable battery life. D6T sensor consumes consistently 5 mA of current and CC3200's current consumption, compared to the sensor, can be neglected (CC3200's CPU activates only in transmitting mode and goes into hibernation mode during downtime). Thus A 12000 mAh commercial battery bank from Jackery Gaint+ was chosen to serve as a power solution because it outputs 5V and enough current to support the 5V USB input of the CC3200 and its battery life can support the sensing device for at least three months, from a conservative estimation. (12000 mAh / 5mA = 2400 hrs = 100 days)

Hardware connection, Multisim, and Ultiboard

The internal library of Multisim doesn't support simulation of the CC3200 board and neither does it have the specification of the D6T sensor and the logic converter. Thus custom footprints were created to represent the components while the 20+20 pin configuration for the CC3200 is replaced by the 10+10 pin configuration for MSP430 in Multisim since only the left 20 pins will be used to establish the connection. The figure below shows the Multisim schematic of the system.

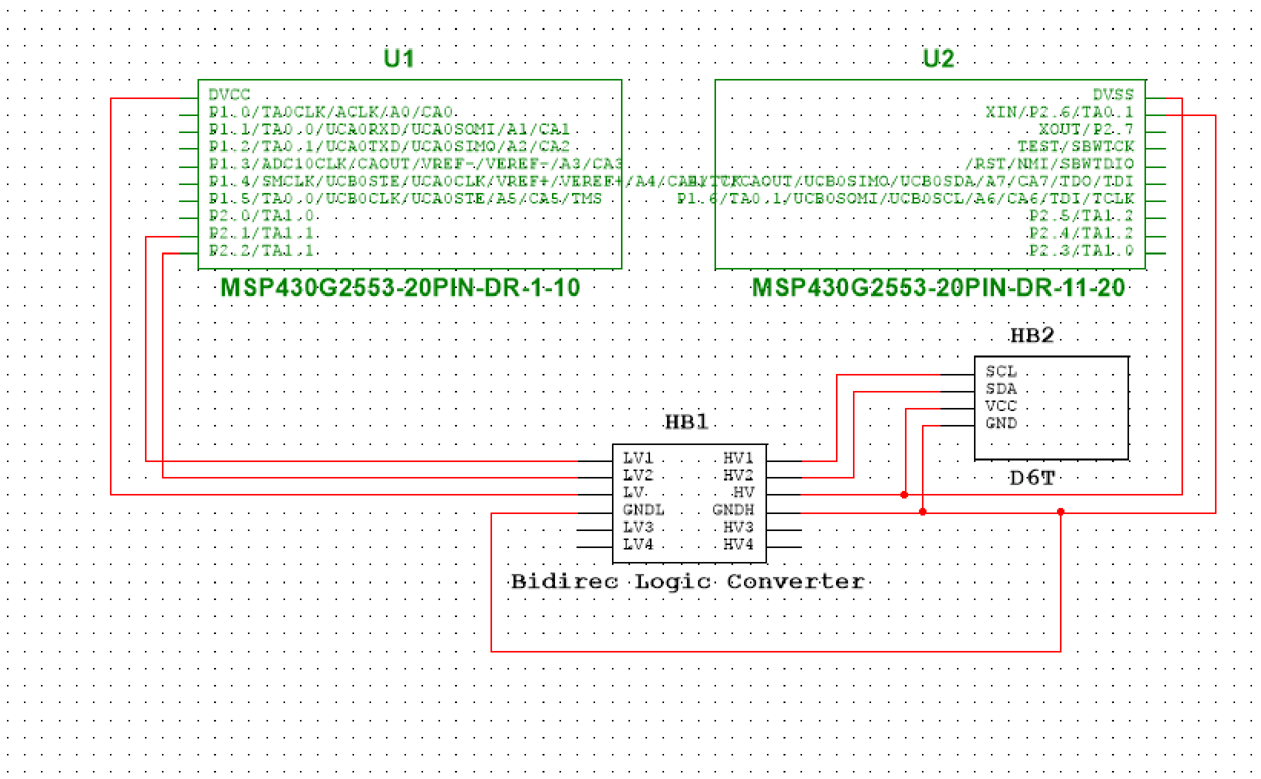

U1
DVCC
P1.0/TA0CLK/ACLK/A0/CA0
P1.1/TA0.0/UCA0RXD/UCA0SOMI/A1/CA1
P1.2/TA0.1/UCA0TXD/UCA0SIMO/A2/CA2
P1.3/ADC10CLK/CAOUT/VREF-/VEREF-/A3/CA3
P1.4/SMCLK/UCB0STE/UCA0CLK/VREF+/VEREF+/A4/CA4/TCK
P1.5/TA0.0/UCB0CLK/UCA0STE/A5/CA5/TMS
P2.0/TA1.0
P2.1/TA1.1
P2.2/TA1.1
MSP430G2553-20PIN-DR-1-10
U2
DVSS
XIN/P2.6/TA0.1
XOUT/P2.7
TEST/SBWTCK
/RST/NMI/SBWTDIO
P1.7/CAOUT/UCB0SIMO/UCB0SDA/A7/CA7/TDO/TDI
P1.6/TA0.1/UCB0SOMI/UCB0SCL/A6/CA6/TDI/TCLK
P2.5/TA1.2
P2.4/TA1.2
P2.3/TA1.0
MSP430G2553-20PIN-DR-11-20
HB2
SCL
SDA
VCC
GND
D6T
HB1
LV1
LV2
LV
GNDL
LV3
LV4
HV1
HV2
HV
GNDH
HV3
HV4
Bidirec Logic Converter

Figure 2: CC3200, D6T, Logic Converter Multisim Schematic

HB1 indicates the logic converter and the top two channels are utilized. The third and fourth row (marked with “LV HV” and “GNDL GNDH”) on the logic converter represent the supply voltages and ground references. HB2 is a schematic of the D6T sensor with SCL and SDA being the I2C lines and VCC and GND being power supply and ground. Additionally, U1 and U2 combine to form the left 20 pin configuration on the CC3200 Launchpad. DVCC and DVSS represent the 3.3V and 5V power pins, P2.1 and P2.2 correspond to clock line and the data line and finally P2.6 is the ground on board. The physical connection is established just as the schematic indicates above.

After the schematic was completed, the Ultiboard layout was designed according to it.

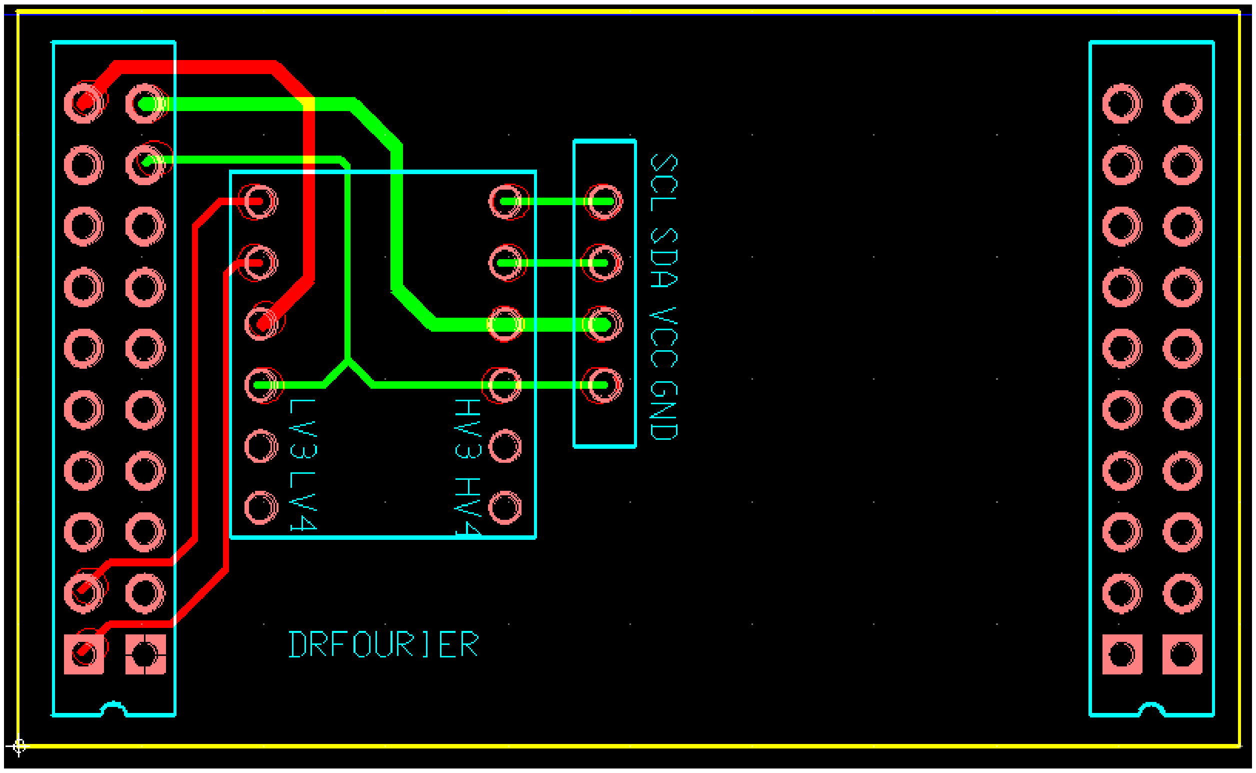


Figure 3:PCB interconnect

As shown in the figure above, the interconnect PCB layout only utilizes the left 20 pins on the CC3200 Launchpad. Logic Converter is placed near the 20 pins and the D6T sensor is close to the converter for the convenience of tracing. Since the layout was designed without importing from Multisim, there are red circles on the layout, indicating DRC errors, which are just misinterpretations of the software.

Case Design

In order to ensure that the sensing device has an aesthetic looking, a 3D printed case was designed. This case serves to enclose the battery, CC3200, PCB interconnects and sensor with open sockets for power indicator, power switch, USB charging, and sensor. The case is consisting of two parts that can be brought together to form a complete case and the dimension is 16cm * 9cm * 6cm.

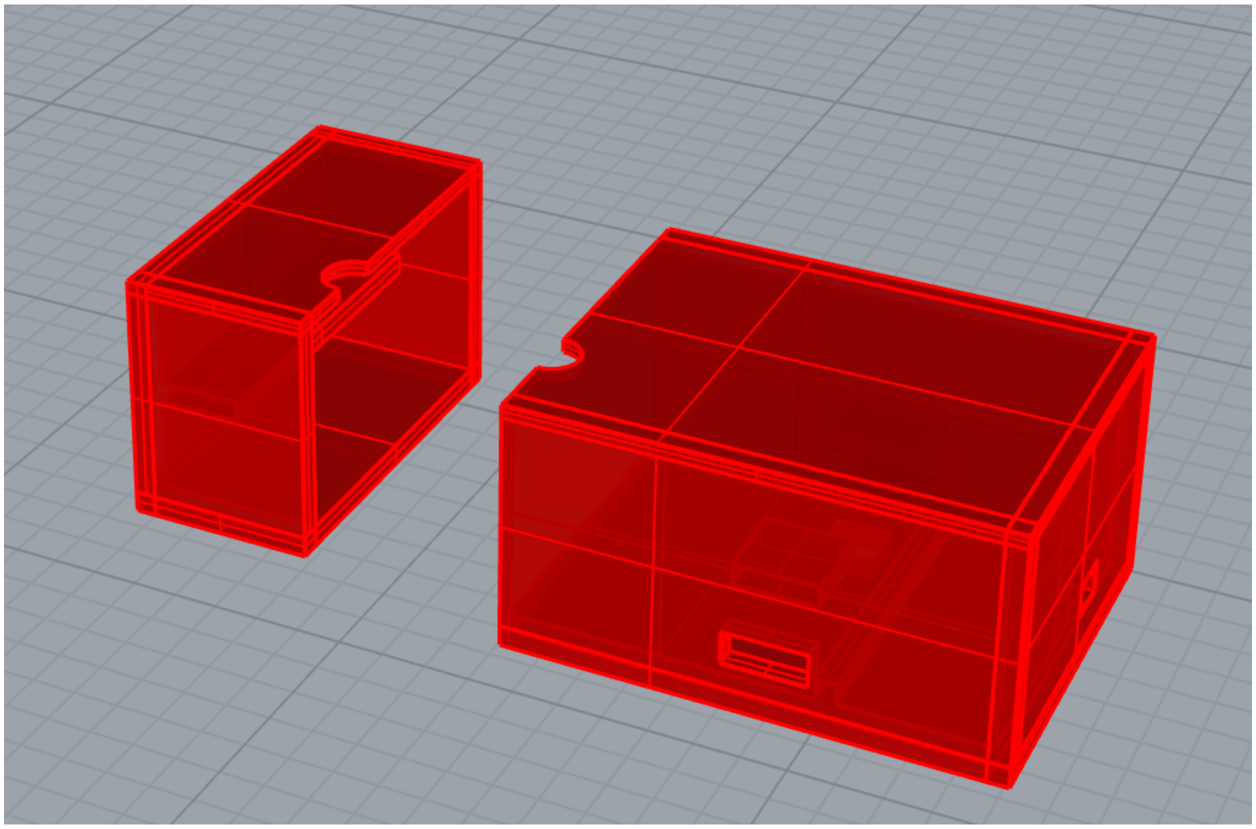


Figure 4: 3D Printed Case-1

The hole on top is an opening for the sensor, a rectangular socket at the front is the power switch for the battery bank and another socket on the right is for USB charging.

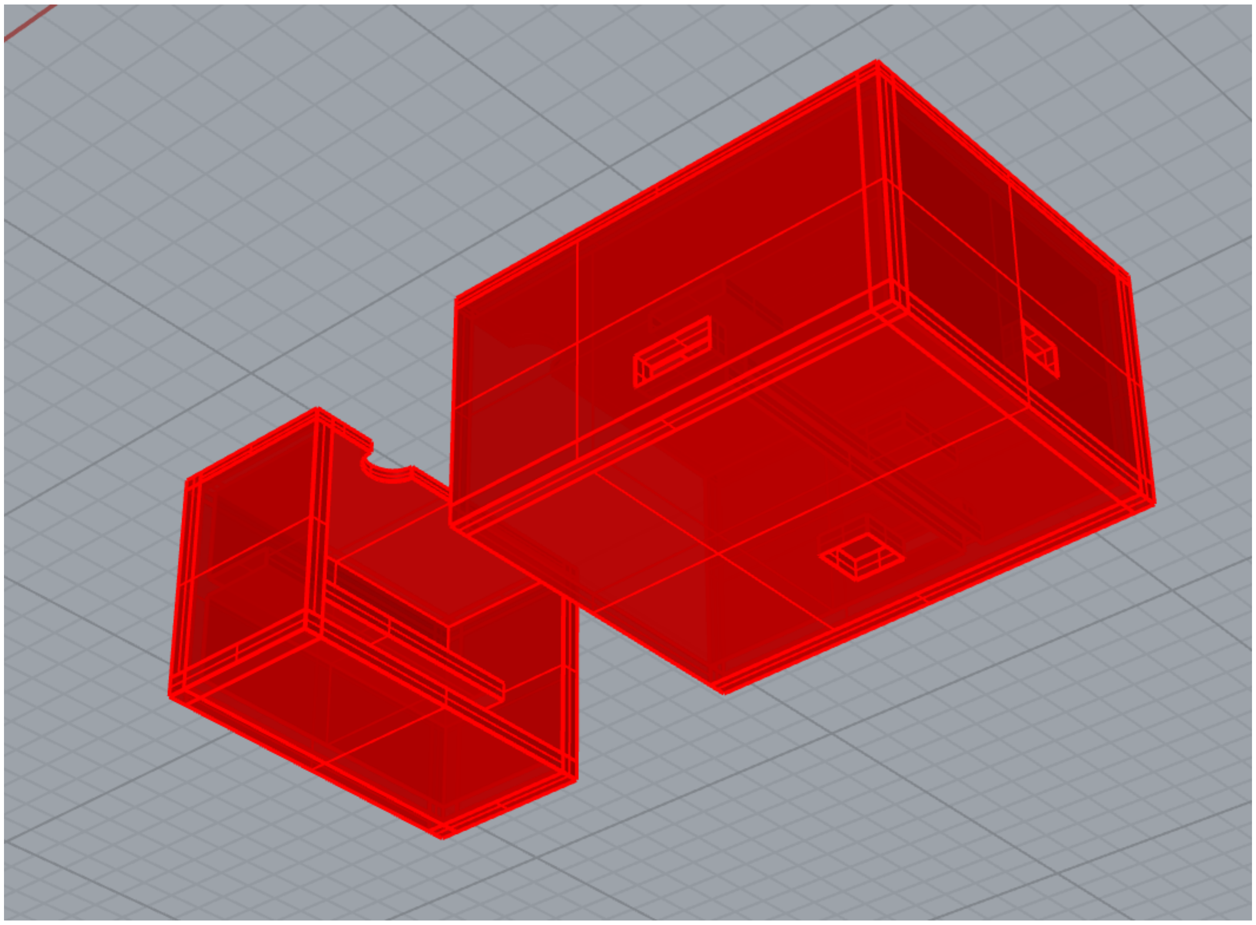


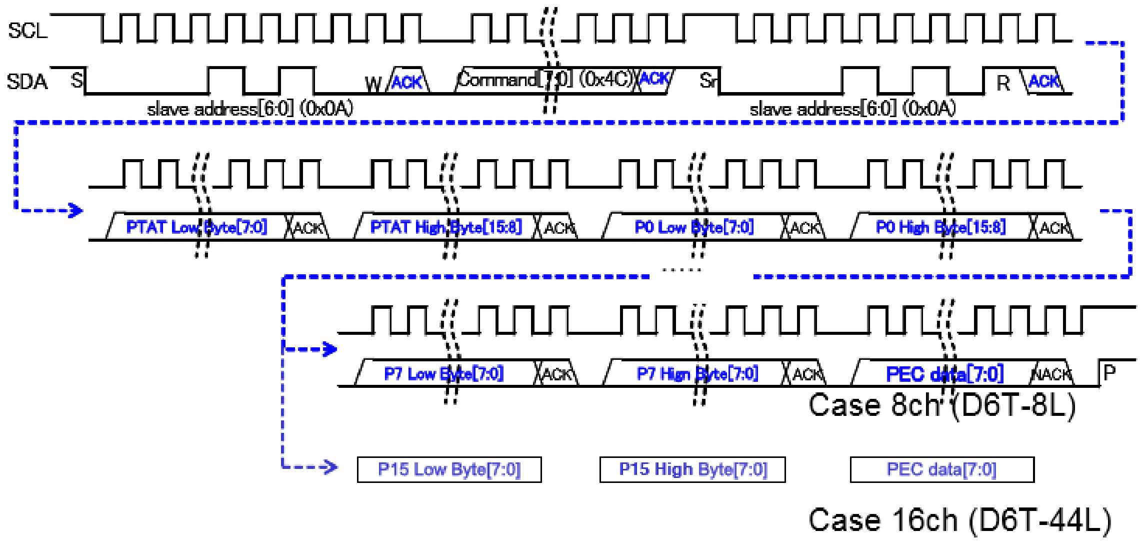
Figure 5: 3D Printed Case-2

The socket on the bottom is the battery life indicator for the battery bank.

Software

D6T Sensor to CC3200

The D6T sensor supports I2C communication and CC3200 is also I2C compliant. For writing the I2C communication code on CC3200, Energia IDE was used instead of CCS because Energia has a well maintained embedded library for CC3200 on Wi-Fi and I2C. According to the I2C timing diagram below, the I2C code was developed in Energia using C language.



“S”	: Start Condition
“Sr”	: Repeat Start Condition
“P”	: Stop Condition
“W/R”	: Write (Lo) / Read (Hi)
“ACK”	: Acknowledge reply
“NACK”	: No-acknowledge reply

Figure 6: I2C Timing

```
void loop()
{
  Wire.beginTransmission(D6T_addr); // transmit to device 0001010b
  Wire.write(D6T_cmd);  //14h = { 0Ah(Addr7) : Write(0b) }
  Wire.endTransmission(false); //repeated start


  // 0001010b is the slave address (10 in decimal)
  // 35 bytes output from sensor (17*2 + 1 = 35)
  Wire.requestFrom(D6T_addr, 35);    // request 2 bytes from slave device

  while(Wire.available() < 35);    // wait for availability
  for(int x = 0; x < 35; x++){
    readbuff[x] = Wire.read();
  }
```

Figure 7: I2C Code Segment

The code segment above shows the initialization and reading instruction for the D6T sensor. D6T_addr is previously defined as the slave address of D6T and D6T_cmd is defined as the reading temperature data command. Up to the end of the code, the I2C communication data path is completed and all temperature data (temperature data of a 4x4 grid) will be stored in the array readbuff[] and it will be further organized and processed by the CC3200.

CC3200 to MQTT Broker

We used CC3200 from Texas Instrument, which is a launch board embedded with Wi-Fi module with TCP/IP connection, to get access to the network. Firstly, we collect data using the code shown in appendix Figure 22. We stored all useful data in an int array with a length of 17 which includes 16 temperatures in a 4 by 4 matrix and a reference temperature inside sensor itself. For the human detection algorithm, when the device starts working, it requires 20 seconds to calibrate room temperatures. Basically, it stores 17 temperature data each second into a two-dimensional array with size of 20 * 17 (sample [20] [17]). After this, we calculate the average temperature in each column into a new array (average [17]). When first 20 seconds end configuration flag(conf) will be set from 1 to 0. When a new set of data arrives after calibration state, a line of old data in the sample [20] [17] will be replaced according to cursor's position. Cursor repeats from 0 to 19 to indicate which row of data inside sample [20] [17] needs to be replaced. We compare each new set of data collected from the sensor to newly calculated average value one by one. If any temperature from the sensor is higher 1℃ than averaged value, we consider human is detected and change flag human from 0 to 1. If the human flag is 1, we will stop updating data in the sample [20] [17]. When data collected from the sensor is longer higher than 1℃ than averaged value, we set human flag from 1 back to 0 and data updating in the sample [20] [17] continues.

With this algorithm, we can avoid temperature variance in different location of sensing area but only capture sudden temperature rises. Thus, hot area or gradually increased temperature due to heat sources such as sunlight or AC will not be considered as human.

Each second program reads human flag and sends out with device serial number to the public broker using MQTT protocol. The detailed codes are shown in Code figure 24. The rest of the essential codes is attached to the appendix section for reference.

MQTT Broker to the Web

We have selected the MQTT server because not only it is free, but also it is very compatible with a low data transmission required system. The receiving end of the socket is set up using the public Paho python libraries. Because this code is incorporated into the webpage, the receiving socket is actively listening whenever the web page is refreshed. The socket will be listening for data sent through a previously agreed port channel, "UVAFourer". The expected data formats will be id number of the thermal sensor and an integer representing whether the seating area for that particular sensor is occupied

(1 is occupied and 0 is unoccupied). Consequently, the system will first parse the data transmitted by the MQTT server and then transmit the results to the webpage handlers.

Integrating the MQTT receiver scripts with the Django platform required further revision of the independent code. Since our web application follows the MVC structure, we use controllers for HTTP requests. In our index page controller, MQTT receiving scripts runs as a separate thread from the main webpage rendering processes. In this case, we not only save time on page rendering but also improved efficiency on the web application as a whole. For the receiver socket of the connection endpoint, our script will receive one data unit from the MQTT topic we subscribe. And the script can be executed in two ways: on-demand (when the page is refreshed) or regularly (the script will execute itself in a certain time period)

Web app construction

For the HTML portion of the web page, the index page as a whole is divided into two parts, head, and body. In the head, we imported JavaScript scripts customized CSS scripts and online supporting scripts. In addition, we added metadata about the webpage configurations and webpage title in the head section as well. The body section includes a navigation bar, a header, a map for NI lab, a list of token explanations and an advanced footer.

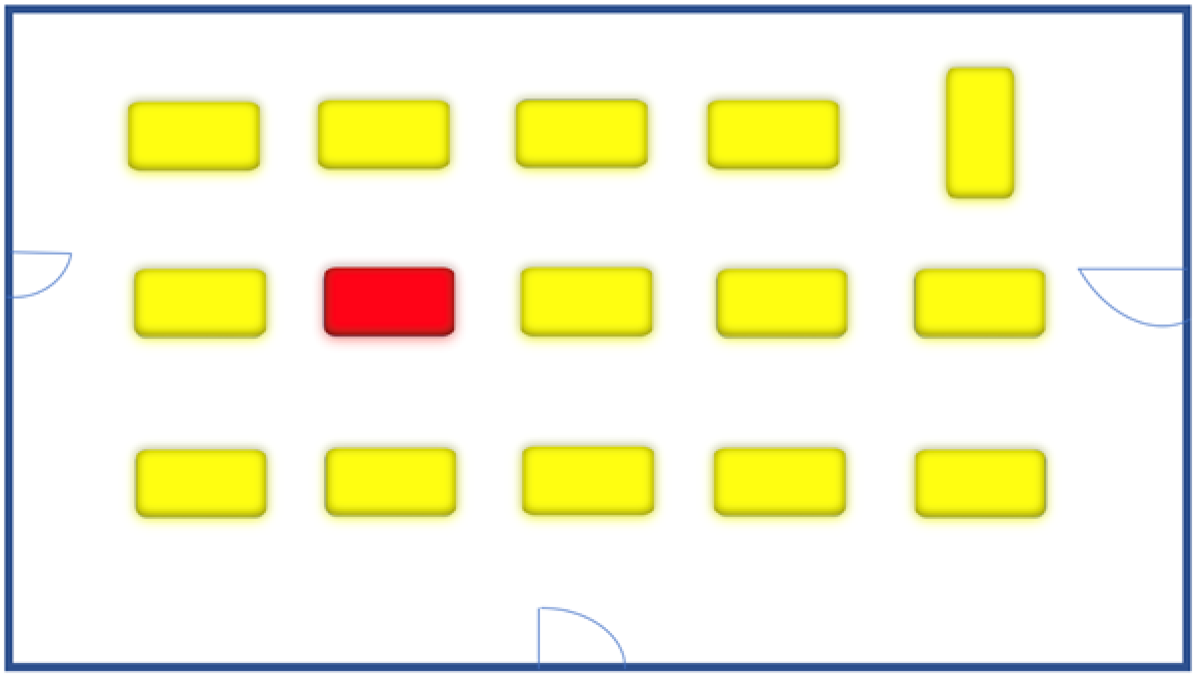
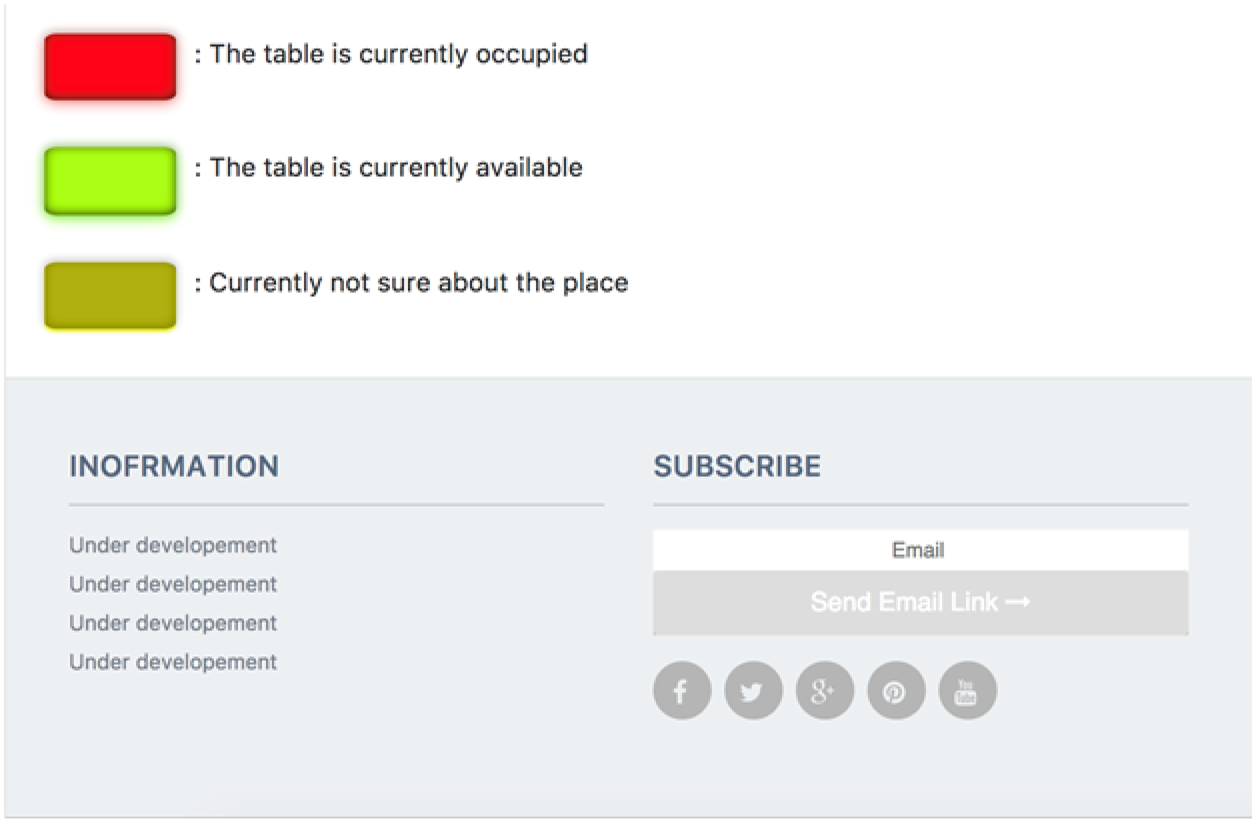


Figure 8:Website Showcase 1


🟥 : The table is currently occupied

🟩 : The table is currently available

🟨 : Currently not sure about the place

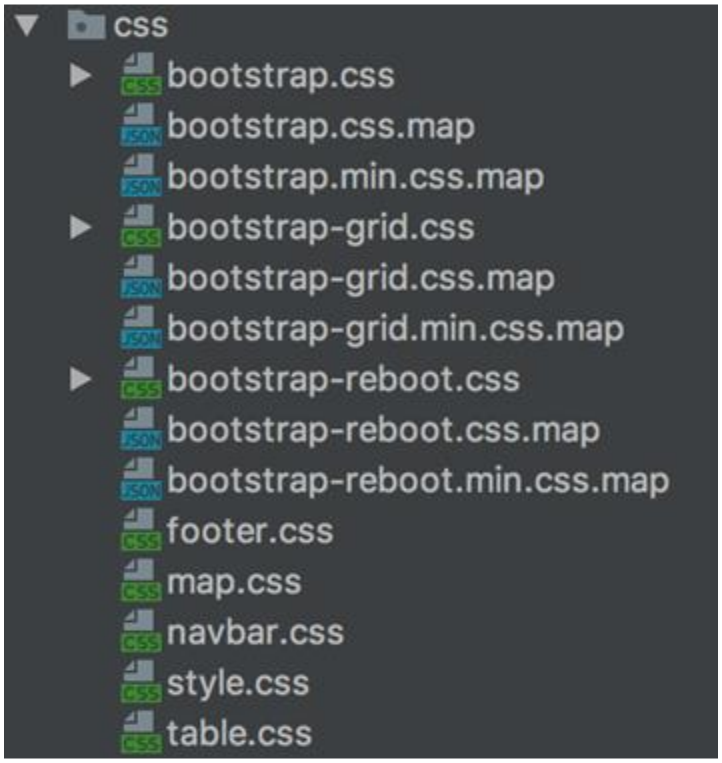
INOFRMATION

Under developement
Under developement
Under developement
Under developement

SUBSCRIBE

Figure 9:Website Showcase 2

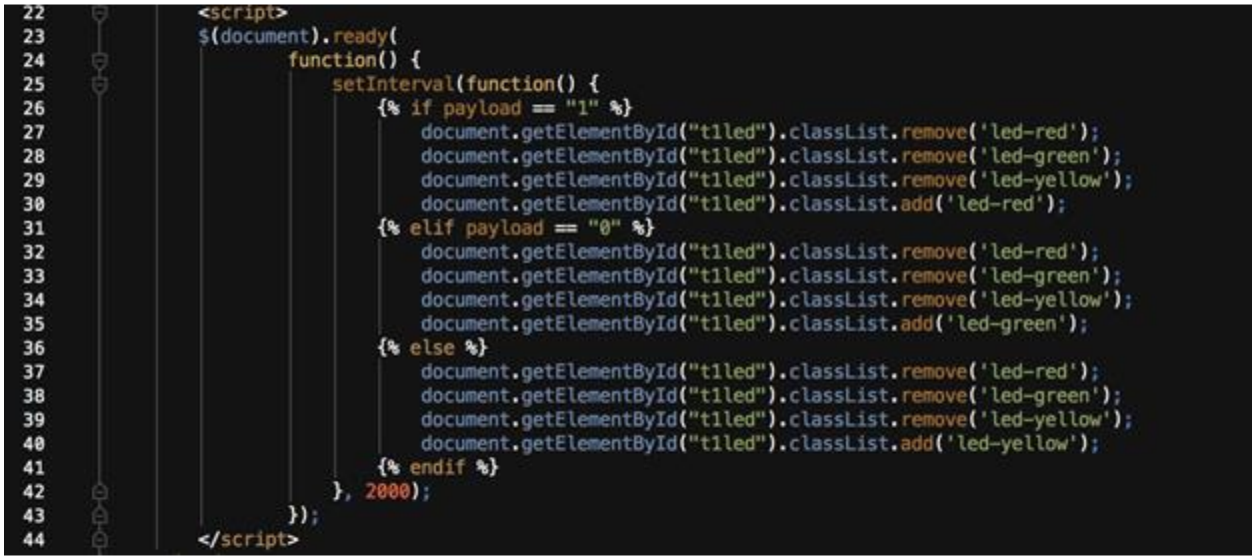
For the styling CSS portion of the web page, we used bootstrap 3.0 in our styling development. More specifically, we used container, row, col classes for designing layout. Furthermore, we used modulation for CSS: we divided the styling scripts into different files and each module contains the rule for a specific logic block, i.e. header, navigation bar, body, and footer.



```
▼ css
  ▶ bootstrap.css
    bootstrap.css.map
    bootstrap.min.css.map
  ▶ bootstrap-grid.css
    bootstrap-grid.css.map
    bootstrap-grid.min.css.map
  ▶ bootstrap-reboot.css
    bootstrap-reboot.css.map
    bootstrap-reboot.min.css.map
    footer.css
    map.css
    navbar.css
    style.css
    table.css
```

Figure 10:CCS Styling

Since we are updating the status for each table, we do not want to refresh the whole page any time there is one small change. Therefore, AJAX is employed for refreshing the individual LED indicators. The logic is anytime when we receive a signal for a certain table, we remove all its CSS classes for displaying effects, and then based on the value of the input data, we assign a corresponding CSS class to the certain table.

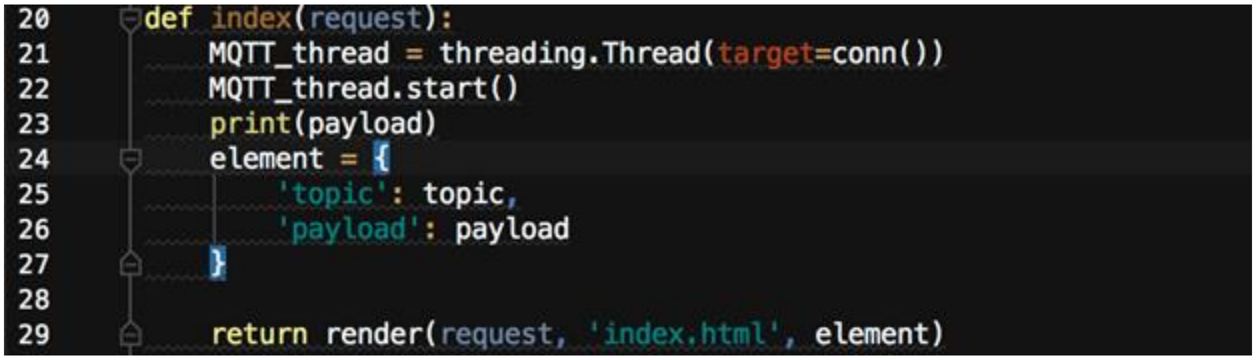


```
<script>
$(document).ready(
        function() {
            setInterval(function() {
                {% if payload == "1" %}
                    document.getElementById("t1led").classList.remove('led-red');
                    document.getElementById("t1led").classList.remove('led-green');
                    document.getElementById("t1led").classList.remove('led-yellow');
                    document.getElementById("t1led").classList.add('led-red');
                {% elif payload == "0" %}
                    document.getElementById("t1led").classList.remove('led-red');
                    document.getElementById("t1led").classList.remove('led-green');
                    document.getElementById("t1led").classList.remove('led-yellow');
                    document.getElementById("t1led").classList.add('led-green');
                {% else %}
                    document.getElementById("t1led").classList.remove('led-red');
                    document.getElementById("t1led").classList.remove('led-green');
                    document.getElementById("t1led").classList.remove('led-yellow');
                    document.getElementById("t1led").classList.add('led-yellow');
                {% endif %}
            }, 2000);
        });
</script>
```

Figure 11: LED Indicator

Running MQTT Receiver Scripts in Controller

Since our web application follows the MVC structure, we use controllers for HTTP requests. In our index page controller, MQTT receiving scripts runs as a separate thread from the main webpage rendering processes. In this case, we not only saved time on page rendering but also improved efficiency on the web application as a whole.

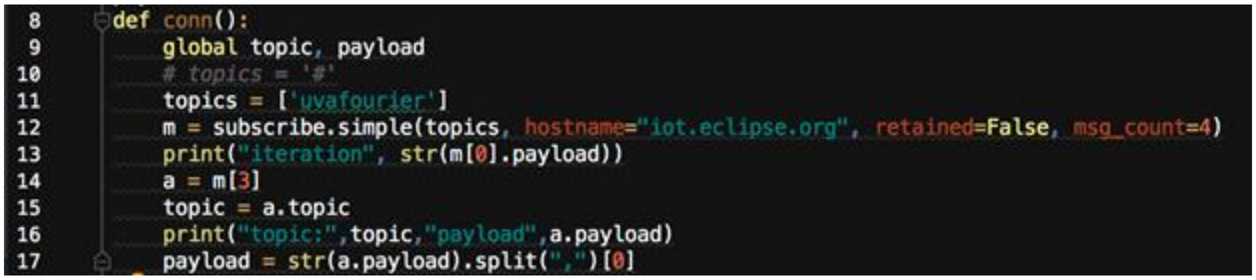


```
def index(request):
    MQTT_thread = threading.Thread(target=conn())
    MQTT_thread.start()
    print(payload)
    element = {
        'topic': topic,
        'payload': payload
    }

    return render(request, 'index.html', element)
```

Figure 12:MQTT Receiver 1

Receiving MQTT Feed on a Topic

Our script will receive one data unit from the MQTT topic we subscribe. And the script can be executed in two ways: on-demand (when the page is refreshed) or regularly (the script will execute itself in a certain time period)



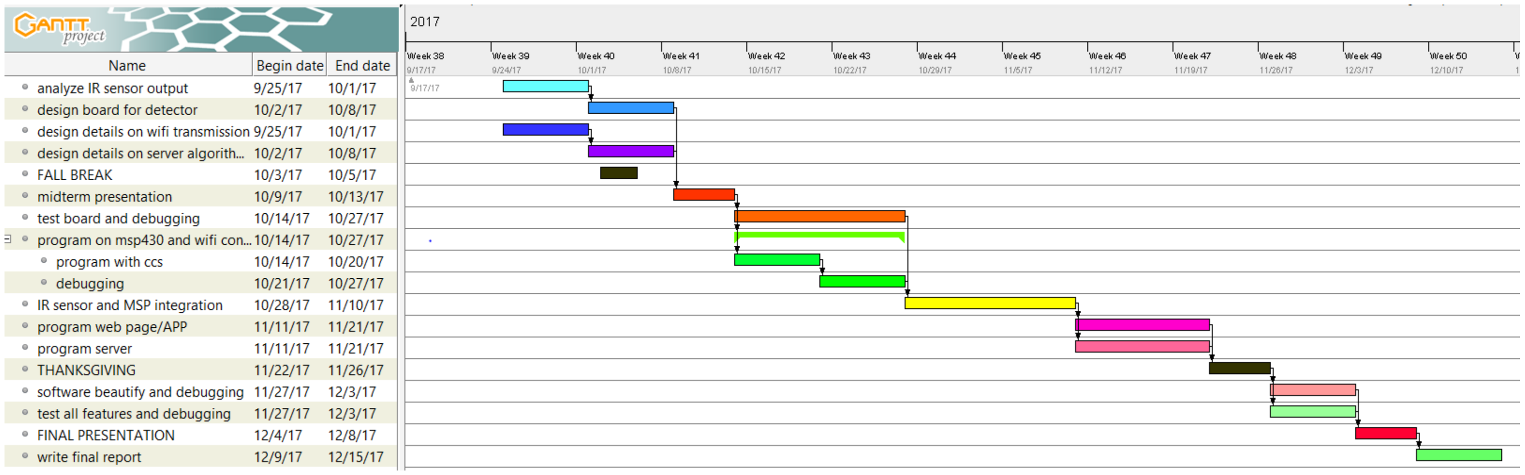
```
def conn():
    global topic, payload
    # topics = '#'
    topics = ['uvafourier']
    m = subscribe.simple(topics, hostname="iot.eclipse.org", retained=False, msg_count=4)
    print("iteration", str(m[0].payload))
    a = m[3]
    topic = a.topic
    print("topic:",topic,"payload",a.payload)
    payload = str(a.payload).split(",")[0]
```

Figure 13:MQTT Receiver 2

Since our application will not only work locally, but also we want it to be able to adapt to various working environments, we used Django Whitenoise to achieve this goal. First of all, we configured the static file path to be somewhere inside the project folder. Then we enabled Whitenoise as a middleware in the settings file. Furthermore, we added compression and caching support. And finally, it is ready to go to any content delivery networks (CDN).

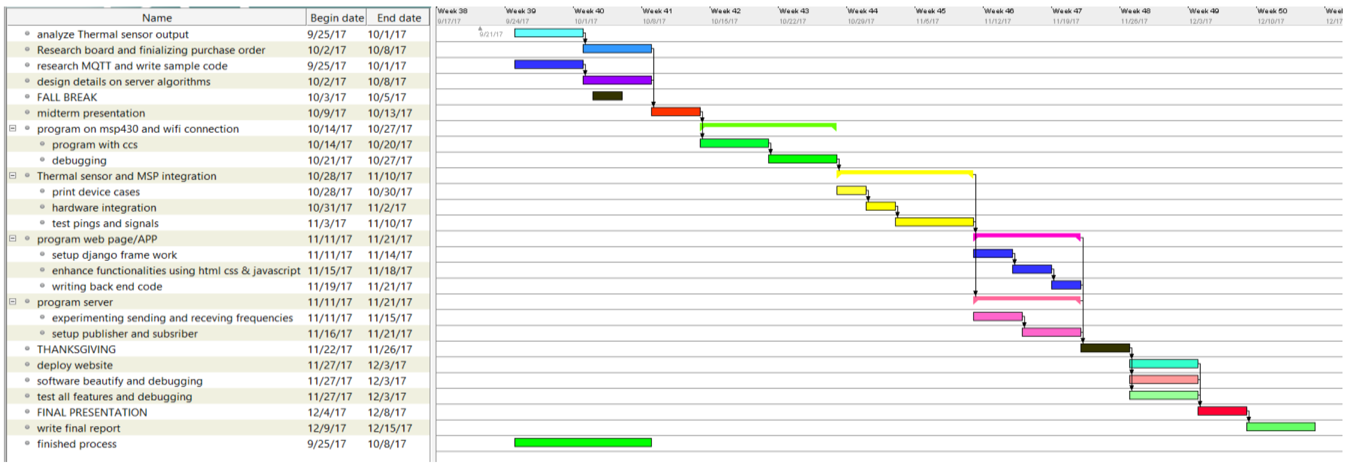
Project Timeline

The timeline for the project underwent some changes primarily due to the fact that an essential wiring harness took too long to arrive and without the wiring harness, D6T sensor was unable to be tested and further code implementation on CC3200 was delayed. Mike and Yingxiang were responsible for the hardware and low-level embedded coding implementation. Mike's primary responsibility was to complete all the hardware design and testing and secondary job was to help with embedded coding. On the other hand, Yingxiang primarily worked on the coding aspect and helped on the hardware selection from time to time. On the software side, Kefan and Timothy were primarily website front-end and secondarily developed website back end.



Name	Begin date	End date
analyze IR sensor output	9/25/17	10/1/17
design board for detector	10/2/17	10/8/17
design details on wifi transmission	9/25/17	10/1/17
design details on server algorith...	10/2/17	10/8/17
FALL BREAK	10/3/17	10/5/17
midterm presentation	10/9/17	10/13/17
test board and debugging	10/14/17	10/27/17
program on msp430 and wifi con...	10/14/17	10/27/17
program with ccs	10/14/17	10/20/17
debugging	10/21/17	10/27/17
IR sensor and MSP integration	10/28/17	11/10/17
program web page/APP	11/11/17	11/21/17
program server	11/11/17	11/21/17
THANKSGIVING	11/22/17	11/26/17
software beautify and debugging	11/27/17	12/3/17
test all features and debugging	11/27/17	12/3/17
FINAL PRESENTATION	12/4/17	12/8/17
write final report	12/9/17	12/15/17

Figure 14:Old Gantt Chart



Name	Begin date	End date
analyze Thermal sensor output	9/25/17	10/1/17
Research board and finializing purchase order	10/2/17	10/8/17
research MQTT and write sample code	9/25/17	10/1/17
design details on server algorithms	10/2/17	10/8/17
FALL BREAK	10/3/17	10/5/17
midterm presentation	10/9/17	10/13/17
program on msp430 and wifi connection	10/14/17	10/27/17
program with ccs	10/14/17	10/20/17
debugging	10/21/17	10/27/17
Thermal sensor and MSP integration	10/28/17	11/10/17
print device cases	10/28/17	10/30/17
hardware integration	10/31/17	11/2/17
test pings and signals	11/3/17	11/10/17
program web page/APP	11/11/17	11/21/17
setup django frame work	11/11/17	11/14/17
enhance functionalities using html css & javascript	11/15/17	11/18/17
writing back end code	11/19/17	11/21/17
program server	11/11/17	11/21/17
experimenting sending and receving frequencies	11/11/17	11/15/17
setup publisher and subsriber	11/16/17	11/21/17
THANKSGIVING	11/22/17	11/26/17
deploy website	11/27/17	12/3/17
software beautify and debugging	11/27/17	12/3/17
test all features and debugging	11/27/17	12/3/17
FINAL PRESENTATION	12/4/17	12/8/17
write final report	12/9/17	12/15/17
finished process	9/25/17	10/8/17

Figure 15:updated Gantt Chart

As can be seen from a comparison of the two Gantt charts, A major change is that because of the delay of the arrival of wiring harness, our team had to add more parallel work to ensure the timeliness of the delivery of the product. Also, some of the tasks were delayed due to the same reason. Due to the effectiveness of conducting the schedule, we ended up finishing the project earlier than expected and had sufficient time to refine and test.

Test Plan

Hardware Testing

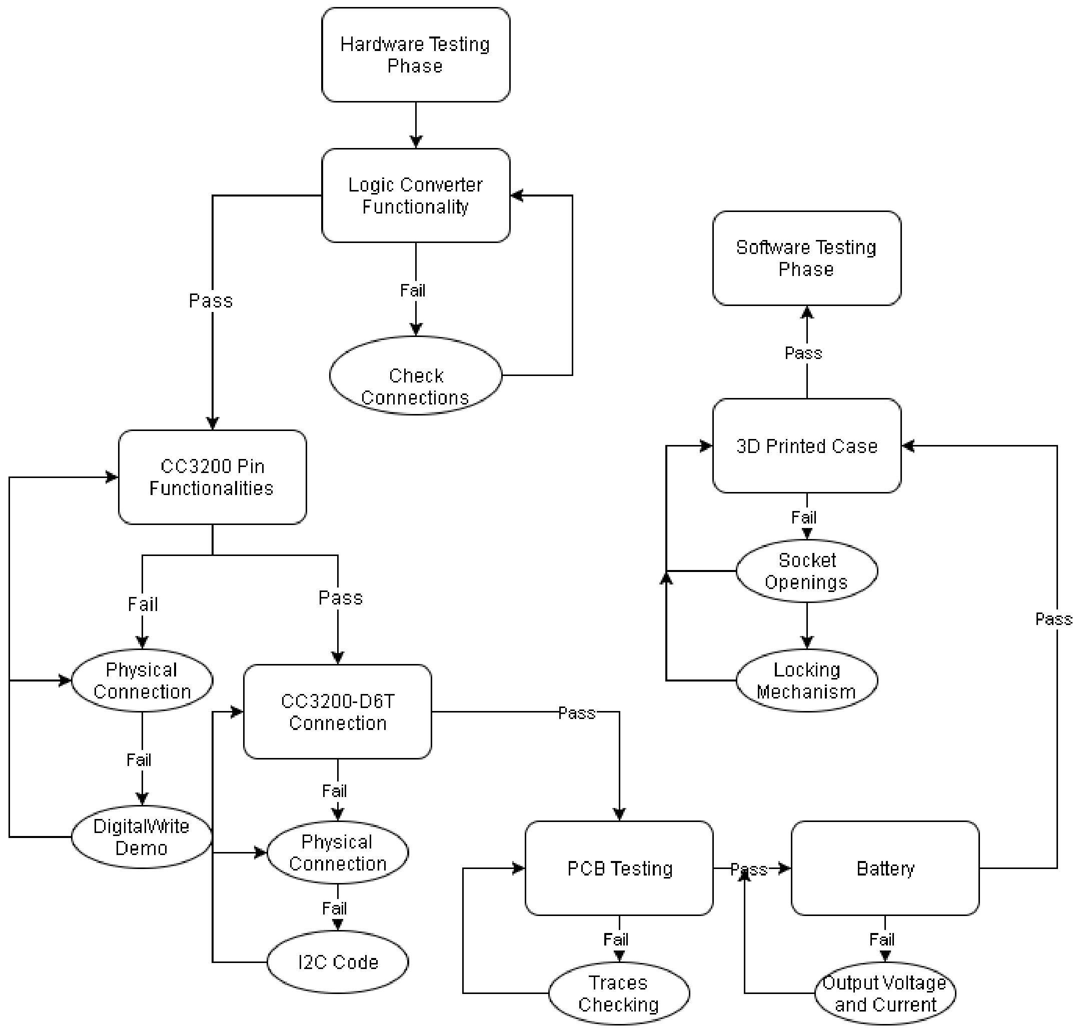


Figure 16:Hardware Testing

Testing on logic converter:

D6T sensor and CC3200 cannot be connected unless the logic converter is tested to be functioning correctly because the high voltage output (5V) of D6T will destroy digital pins on the CC3200. Thus the first step of hardware testing is to make sure that the logic converter behaves as desired.

The testing was done on the breadboard by establishing correct input voltage connections (3.3V and 5V input from the VirtualBench) and sequentially testing the first and second channel of the converter with the input of a square wave of various amplitude from the function generator. An input of 0-5V square wave on the HV channel of the converter will be reduced to 3.3V on the LV channel and an input of 0-3.3V square wave on the LV channel will be boosted to 5V on the HV channel. The actual testing was done successfully and the results confirmed that the device was working properly.

Testing on CC3200:

CC3200 has to supply the correct input voltage, clock line and data line to the D6T sensor in order to ensure proper data transmission between the master and slave (I2C protocol). First, Two I2C pins were identified and were tested using a digital-write demo program. The pins must be able to be pulled up and down to both 3.3V and 0V and the testing results on the VirtualBench using oscilloscope showed that they were functioning properly. Secondly, power pins were identified as VCC and 5V pins on CC3200 and tested to have the correct voltage when CC3200 was powered by using a digital multimeter.

Testing on the connection between sensor and CC3200:

I2C code implementation is part of the testing for sensor-processor connection. The code implementation was completed following from the I2C timing diagram and physical connections were established on the breadboard among CC3200, logic level shifter, and D6T sensor. Data path would be successfully created if the serial monitor successfully displays consecutive temperature readings. No results were displayed for the first try of the testing and the attention was directed to the code implementation after physical connections were checked to be correct. After fixing a few timing issues on the I2C coding, CC3200 was confirmed to be able to receive temperature readings from the D6T sensor.

Testing on PCB Board:

All the traces have to be tested before soldering the PCB to the pin layout on CC3200. Since the PCB board for our project only functions as an interconnect header for the CC3200 to avoid messy wiring, as long as the traces and connections are correct, the PCB board should function as desired. During the actual testing, PCB was tested to perform well on the first try.

Testing on Battery:

Battery bank should support the 5V USB input of CC3200 Launchpad. The testing was simply done by connecting the battery bank USB output to CC3200 USB input and the onboard power LED turning on indicated that the battery is voltage and current was sufficient to support CC3200

Testing on 3D Printed Case:

Battery, CC3200 and D6T sensor should be enclosed tightly in the case and the case's locking mechanism should function correctly. Additionally, case openings for the power button, battery life indicator, and USB charging should have sufficient dimension to support functionalities. During the actual testing of the first version of the 3D printed case, locking mechanism was proved to be not good enough to combine the two pieces of the case tightly. A second version was designed to solve the problem and the current version of the case can be detached freely with ease.

Software Testing

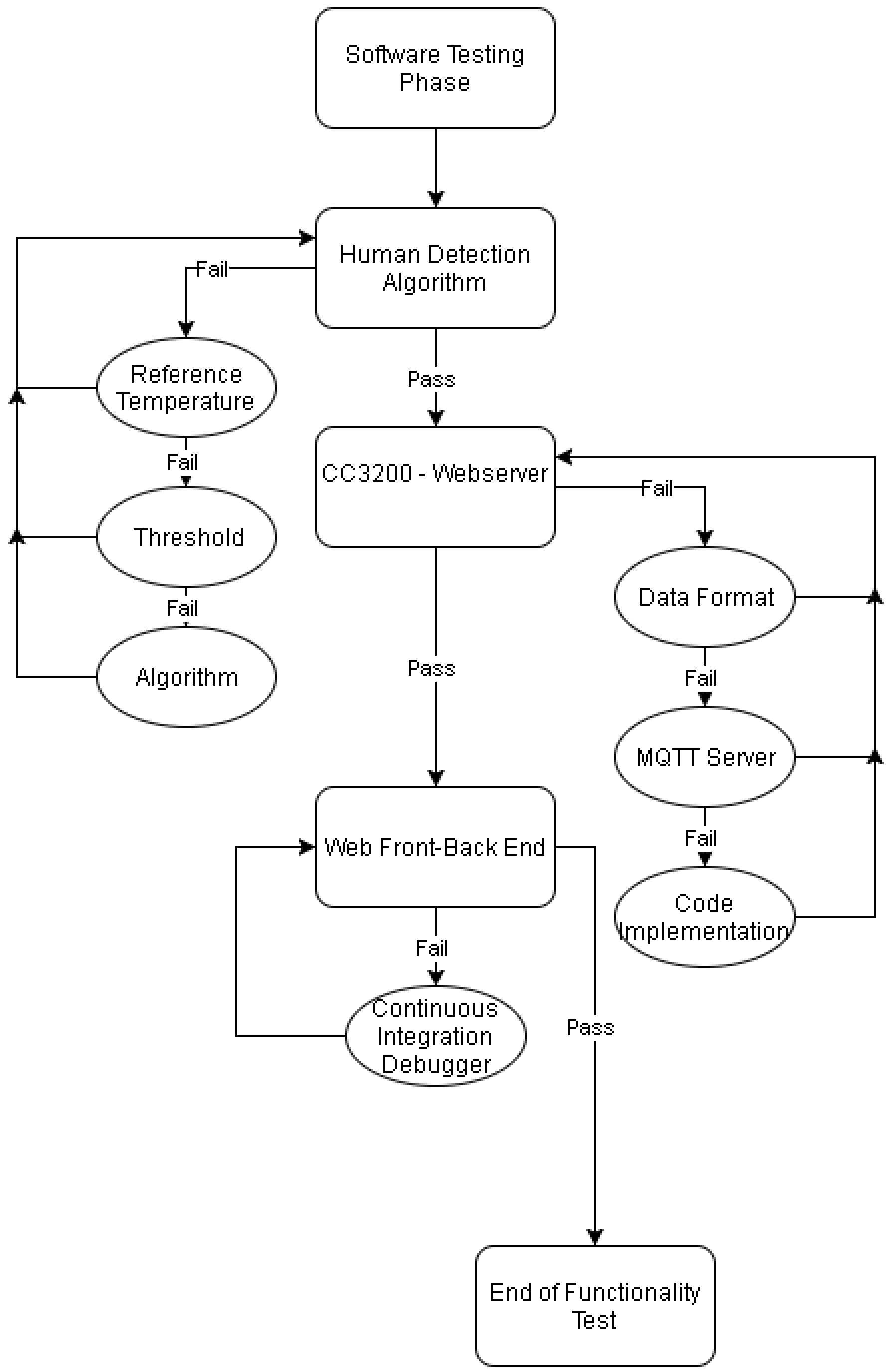


Figure 17:Software Testing

Testing on human detection algorithm:

The algorithm was expected to infer the presence of human from 16 temperature data points in a 4x4 grid with reasonable speed. The algorithm was supposed to interpret human presence as Boolean values 0 and 1 (absence and presence). During the actual testing phase, the algorithm worked properly for the first few tries but as the room temperature changed, there was a problem with identifying the correct reference temperature (room temperature) since our algorithm defines a hard-coded heuristic reference temperature and threshold. In order to compensate for the room temperature change, a calibration phase was added to the algorithm to first calculate the room temperature for the first 20 seconds of starting up phase. The algorithm then continuous tried to update the reference temperature as long as no human presence was detected. The new implementation was tested for hours without issues and became our current version of the human detection algorithm.

Testing on the connection between MQTT sender (CC3200) and MQTT receiver (Python script):

The purpose of this connection stage is to securely send data from the CC3200 to the MQTT server and to receive the data transmitted by the public server using the python script. The difference in the respective computing languages is not a concern because both are independent steps connecting to the black-box MQTT server.

To test the connection between the CC3200 to the MQTT server, we have tried sending various data formats accompanied by different quality of service levels. Consequently, we have observed many combinations that lead to different sending and receiving times. In addition, we would also receive ack responses from the MQTT server to indicate the success of the sending of our data.

To test the connection between the MQTT server and the python script, we have first tested receiving data from a public channel. Since there is constant data sent by anonymous individuals, we should expect to receive random data. The data is received almost instantaneously, as we have expected.

After the initial success, we have tested sending and receiving data through a predefined channel, ‘UVAFOURIER’. This is a challenge to us because we have to not only send and receive the data in a timely manner but also ensure the consistency of the data format to bring usability of the data.

Testing on data visualization and displaying in a web application:

The purpose of our last stage is to deliver our final product’s user interface. The web application is where people can access and use our project, so its testing becomes crucial.

To test the data visualization, I created a logic block that converts received string values to a certain color. More specifically, the empty seat, occupied seat and undetermined seat will be converted to green, red and yellow.

To test the display, I applied a method of unit testing, where I fed the web page with controlled dummy data wrapped inside JSON. Since we have the full control of the input data, we can toggle the display and test all cases out without waiting for the corresponding real data.

Finally, our deployment also made use of continuous integration, which is also a testing on our code. If there happen to be some bugs, they will be detected and developers will be notified before the code goes live.

Final Results

Our final project performed very well compared to our initial expectations in the proposal. The Seat Detecting System is able to accurately assess the presence of humans in the designated seating area and consistently transmit this data to the client web pages. In other words, a student can go on to the specified website and check if there is an unoccupied seat in the building. Based on the result, the student could either happily find a seat to study, or save 20 minutes of his time looking for a nonexistent empty seat by moving on to another building to look for unoccupied study spots.

The only drawback to our system is the refreshing time of our webpage is about 10 seconds instead of instant time. This is due to slow transmission of data through our predefined channel. This problem could possibly be avoided by utilizing a paid server data transmission service. Although the short lag is not as ideal as instant speed, it is definitely acceptable in a client perspective. As a whole, our team has exceeded above and beyond our predefined success criteria shown below.

- The device, embedded with the thermal sensor and Wi-Fi connection, is able to communicate to the server.
- The server receives a signal from every single device and displays correct information to the user on webpages or app.
- A:
    1. All features are correctly implemented
    2. The detector sends correct signal to online server
    3. End-host receives signal and shows seat availability
- B:
    1. One or two features are missing

    2. The detector has some flaws in detecting occupancy of seats.

- C:
    1. The device detects with flaws
    2. Wi-Fi connection is not stable for device or central display

Costs

Component Name	Supplier	Quantity	Unit Price	Total Price
D6T Sensor	Digikey	1	$52	$52
3D printed case	NI Lab	1	$20 (estimate)	$20
PCB	Unknown	1	$20 (estimate)	$20
Battery	Amazon	1	$25.99	$25.99
Wiring harness and connector	mRobotics	1	$5.3	$5.3
CC3200 chip	Digikey	1	$29.99	$29.99
Total				$153.28

Table 1 - Estimated Costs

If the product is going to mass production, the cost of each unit will go down significantly and an updated individual device cost table is shown below (assuming 10000 unit quantities):

Component Name	Supplier	Quantity	Unit Price	Total Price
D6T Sensor	Digikey	1	$38	$38
3D printed case	NI Lab	1	$10 (estimate)	$10
PCB	Unknown	1	$5 (estimate)	$5
Battery	Amazon	1	$15 (estimate)	$15
Wiring harness and connector	mRobotics	1	$0 (negligible)	$0
CC3200 chip	Digikey	1	$6.7	$6.7
Total				$74.7

Table 2 - Updated Estimated Costs

Future Work

As previously mentioned in the final results section, a short lag is present in the final design of our product. For future projects, it is definitely realistic to research into different means to speed up the data transmission rate through a public server. In addition, it is also worth looking to better reliable data transmission route in addition to using a public MQTT server. The alternative usages may lead to surprising results.

For future hardware integration, we might need to redesign the case of our device. The case will have better push buttons and easier USB port access for charging up the device. Also, we will design another type of the device unit, which can be plugged into wall power sources.

For future web application development, there are several tricks to tweak. First of all, we are going to make the MQTT receiving script constantly run itself and the expected result is the loading time will be reduced to around 1 second from the current 5 to 8 seconds. Second, we will modulate the HTML template for better code reuse since we will have more than one page, one for each place where we set up our devices.

Furthermore, in order for the product to go into mass production, Launchpad can be replaced by just the CC3200 chip. In this way, more PCB design will be involved since pin layout will not be available any more but the cost and the size of the device will go down significantly.

References

[1]"Internet of things," *Wikipedia.* 16-Sep-2017.

[2]Adafruit. (2017). PIR Sensor (#555-28027). [online] Available at: https://cdn-learn.adafruit.com/assets/assets/000/010/136/original/PIRSensor-V1.2.pdf [Accessed 11 Sep. 2017].

[3]Anon, (2004). MSP430 Internet Connectivity. [online] Available at: http://www.ti.com/lit/an/slaa137a/slaa137a.pdf [Accessed 11 Sep. 2017].

[4]Anon, (2014). Wireless connectivity for the Internet of Things (IoT) with MSP430™ microcontrollers (MCUs). [online] Available at: http://www.ti.com/lit/wp/slay028/slay028.pdf [Accessed 11 Sep. 2017].

[5]Anon, (2017). CC3100(ACTIVE) SimpleLink™ Wi-Fi® Network Processor, Internet-of-Things Solution for MCU Applications. [online] Available at: http://www.ti.com/product/CC3100/datasheet [Accessed 11 Sep. 2017].

[6]Anon, (2017). MQTT Protocol. [online] Available at: http://mqtt.org/ [Accessed 11 Sep. 2017].

[7]Anon, (2017). Ultra-Low-Power Motion Detection Using the MSP430F2013. [online] Available at: http://www.ti.com/lit/an/slaa283b/slaa283b.pdf [Accessed 11 Sep. 2017]

[8]US Department of Energy, (2017). Use of Occupancy Sensors in LED Parking Lot and Garage Applications: Early Experiences. [onine] Available at http://www.pnnl.gov/main/publications/external/technical_reports/PNNL-21923.pdf[Accessed [Accessed 20 Sept., 2017]

[9] Becker, Fritzsche, and Prestele(2017). Vehicle occupancy monitoring with optical range-sensors. Available at http://ieeexplore.ieee.org/abstract/document/1336361/ [Accessed 20 Sept., 2017]

[10] Sinha, Sharma, Goswami (2017). Design of an energy efficient Iot enabled smart system based on DALI network over MQTT protocol. Available at http://ieeexplore.ieee.org/document/7977309/ [Accessed 20 Sept., 2017]

[11]“Human sensing,” *Wikipedia*. 17-Dec-2016.

[12]“CC3100 SimpleLink^{TM} Wi-Fi® Network Processor, Internet-of-Things Solution for MCU Applications | TI.com.” [Online]. Available: http://www.ti.com/product/CC3100. [Accessed: 22-Sep-2017].

[13]“Model–view–controller,” *Wikipedia*. 04-Dec-2017.

[14]“The Web framework for perfectionists with deadlines | Django.” [Online]. Available: https://www.djangoproject.com/. [Accessed: 11-Dec-2017].

[15]“Getting started · Bootstrap 3.0.3 Documentation - BootstrapDocs.” [Online]. Available: http://bootstrapdocs.com/v3.0.3/docs/getting-started/. [Accessed: 11-Dec-2017].

[16]“AJAX Introduction.” [Online]. Available: https://www.w3schools.com/xml/ajax_intro.asp. [Accessed: 11-Dec-2017].

[17]“Heroku,” *Wikipedia*. 03-Dec-2017.

[18] “IEEE 802.11” [Online]. Available: https://en.wikipedia.org/wiki/IEEE_802.11

[19] L. J. Brackney, “Image processing occupancy sensor,” 27-Jul-2016.

[20] B. Bilger, “Occupancy sensor and method for home automation system,” 21-Jun-2005.

[21] A. M. Fadell, M. L. Rogers, K. A. ROGERS, A. K. ISHIHARA, S. Ben-Menahem, and R. SHARAN, “Occupancy pattern detection, estimation and prediction ,” 12-May-2012.

[22] 4pcb.com. (2017). BareBones, 2 Layer & 4 Layer PCB Pricing | Advanced Circuits.[online] Available at: http://www.4pcb.com/pcb-prototype-2-4-layer-boards-specials.html [Accessed 15 Dec. 2017].

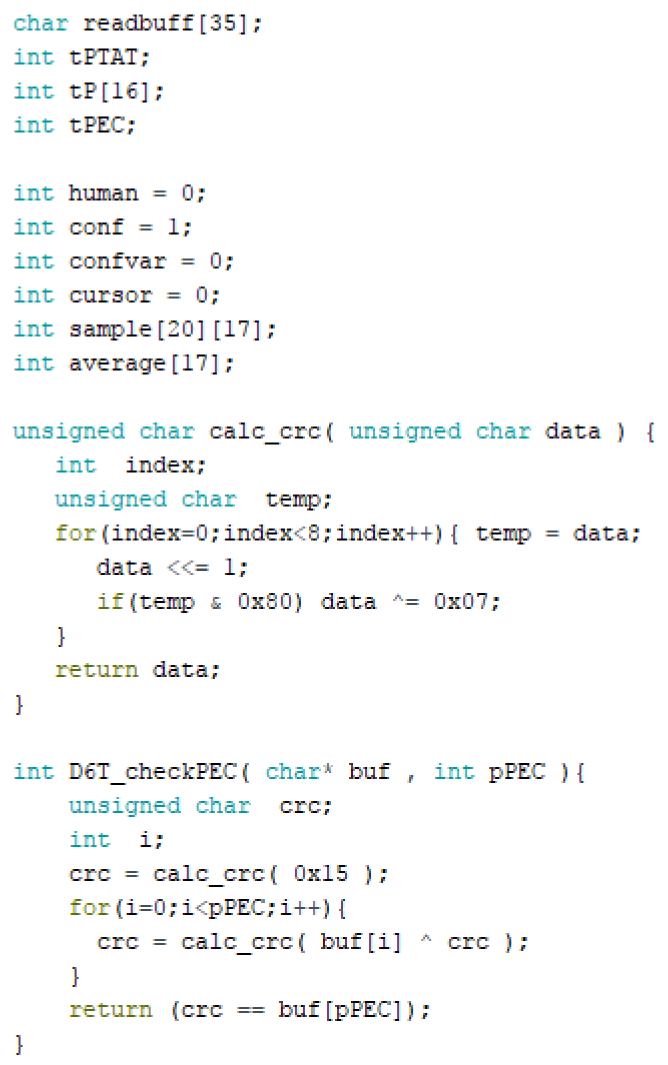
[23] En.wikipedia.org. (2017). I²C. [online] Available at: https://en.wikipedia.org/wiki/I%C2%B2C [Accessed 15 Dec. 2017].

[24] En.wikipedia.org. (2017). IEEE 802.11. [online] Available at: https://en.wikipedia.org/wiki/IEEE_802.11 [Accessed 15 Dec. 2017].

[25] Mqtt.org. (2017). MQTT. [online] Available at: http://mqtt.org/ [Accessed 15 Dec. 2017].

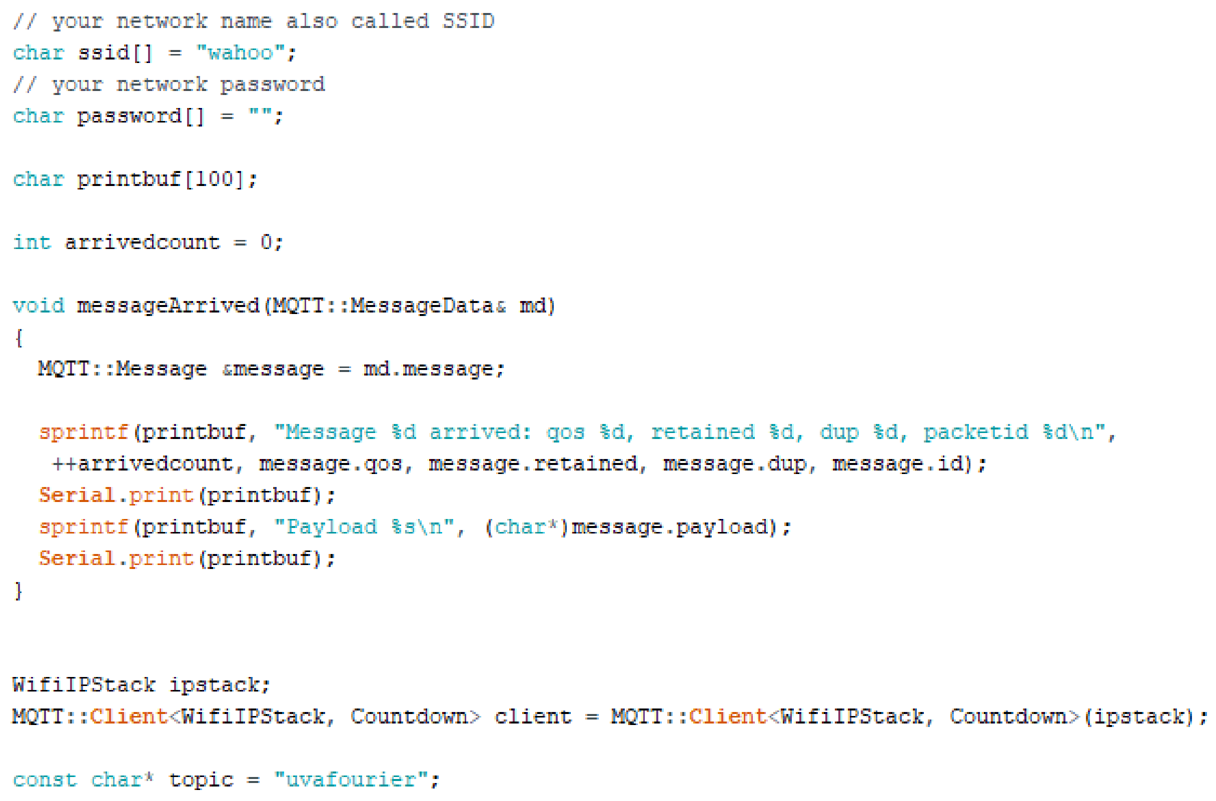
[26] Tools.ietf.org. (2017). RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1. [online] Available at: https://tools.ietf.org/html/rfc2616 [Accessed 15 Dec. 2017].

Appendix



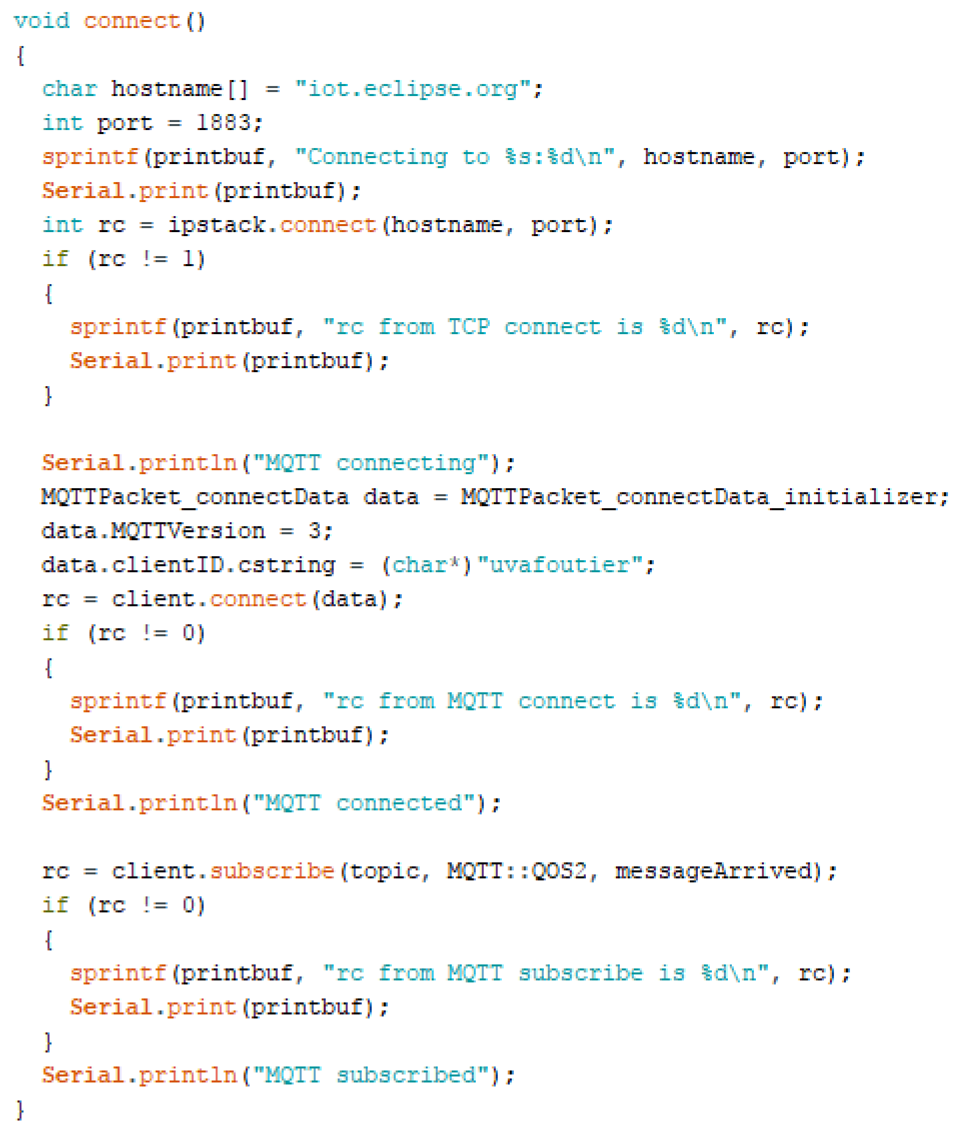
```
char readbuff[35];
int tPTAT;
int tP[16];
int tPEC;

int human = 0;
int conf = 1;
int confvar = 0;
int cursor = 0;
int sample[20][17];
int average[17];

unsigned char calc_crc( unsigned char data ) {
   int  index;
   unsigned char  temp;
   for(index=0;index<8;index++){ temp = data;
      data <<= 1;
      if(temp & 0x80) data ^= 0x07;
   }
   return data;
}

int D6T_checkPEC( char* buf , int pPEC ){
    unsigned char  crc;
    int  i;
    crc = calc_crc( 0x15 );
    for(i=0;i<pPEC;i++){
      crc = calc_crc( buf[i] ^ crc );
    }
    return (crc == buf[pPEC]);
}
```

Figure 18:program setup and variables initiation



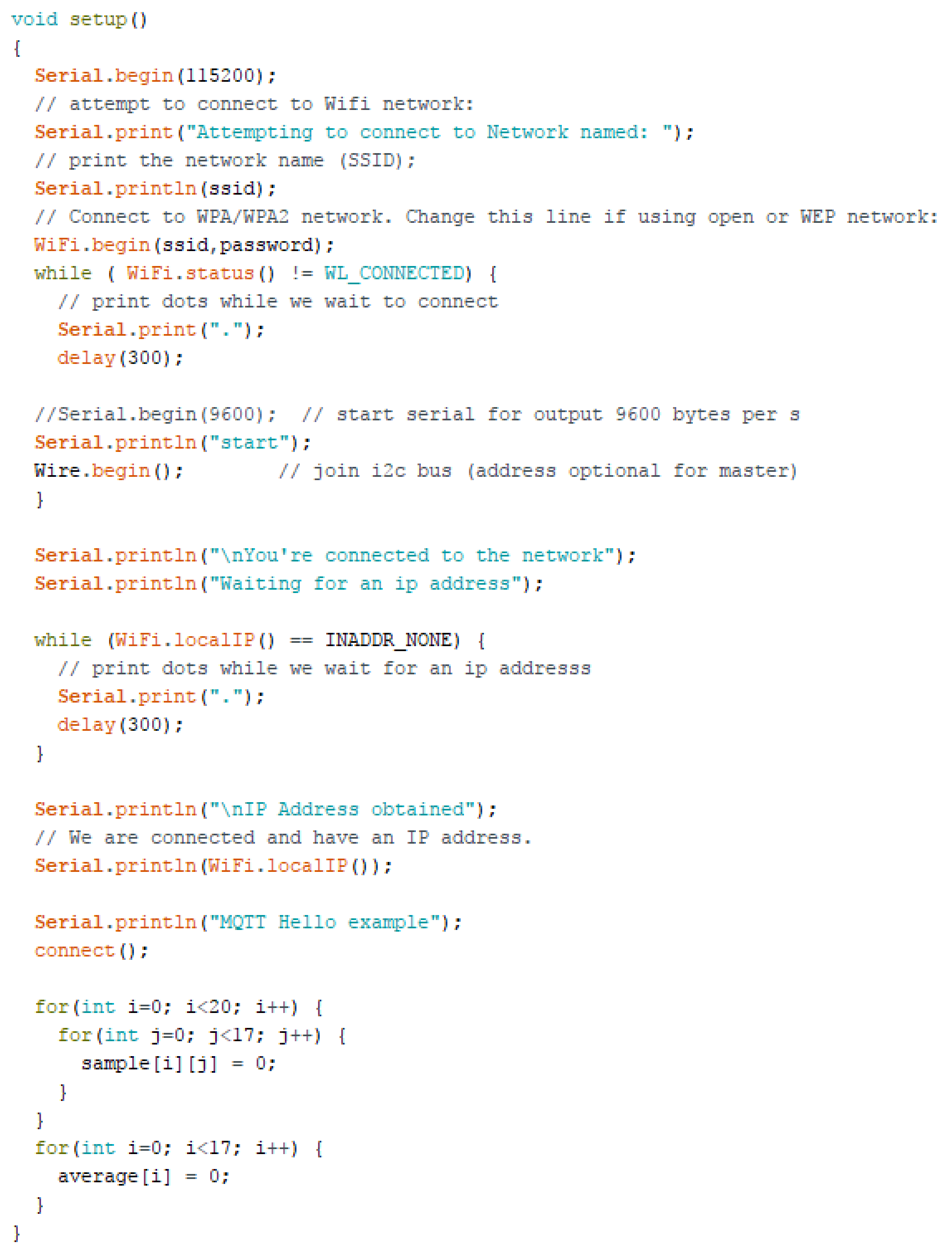
```
// your network name also called SSID
char ssid[] = "wahoo";
// your network password
char password[] = "";

char printbuf[100];

int arrivedcount = 0;

void messageArrived(MQTT::MessageData& md)
{
  MQTT::Message &message = md.message;

  sprintf(printbuf, "Message %d arrived: qos %d, retained %d, dup %d, packetid %d\n",
   ++arrivedcount, message.qos, message.retained, message.dup, message.id);
  Serial.print(printbuf);
  sprintf(printbuf, "Payload %s\n", (char*)message.payload);
  Serial.print(printbuf);
}


WifiIPStack ipstack;
MQTT::Client<WifiIPStack, Countdown> client = MQTT::Client<WifiIPStack, Countdown>(ipstack);

const char* topic = "uvafourier";
```

Figure 19:Wifi connection setup



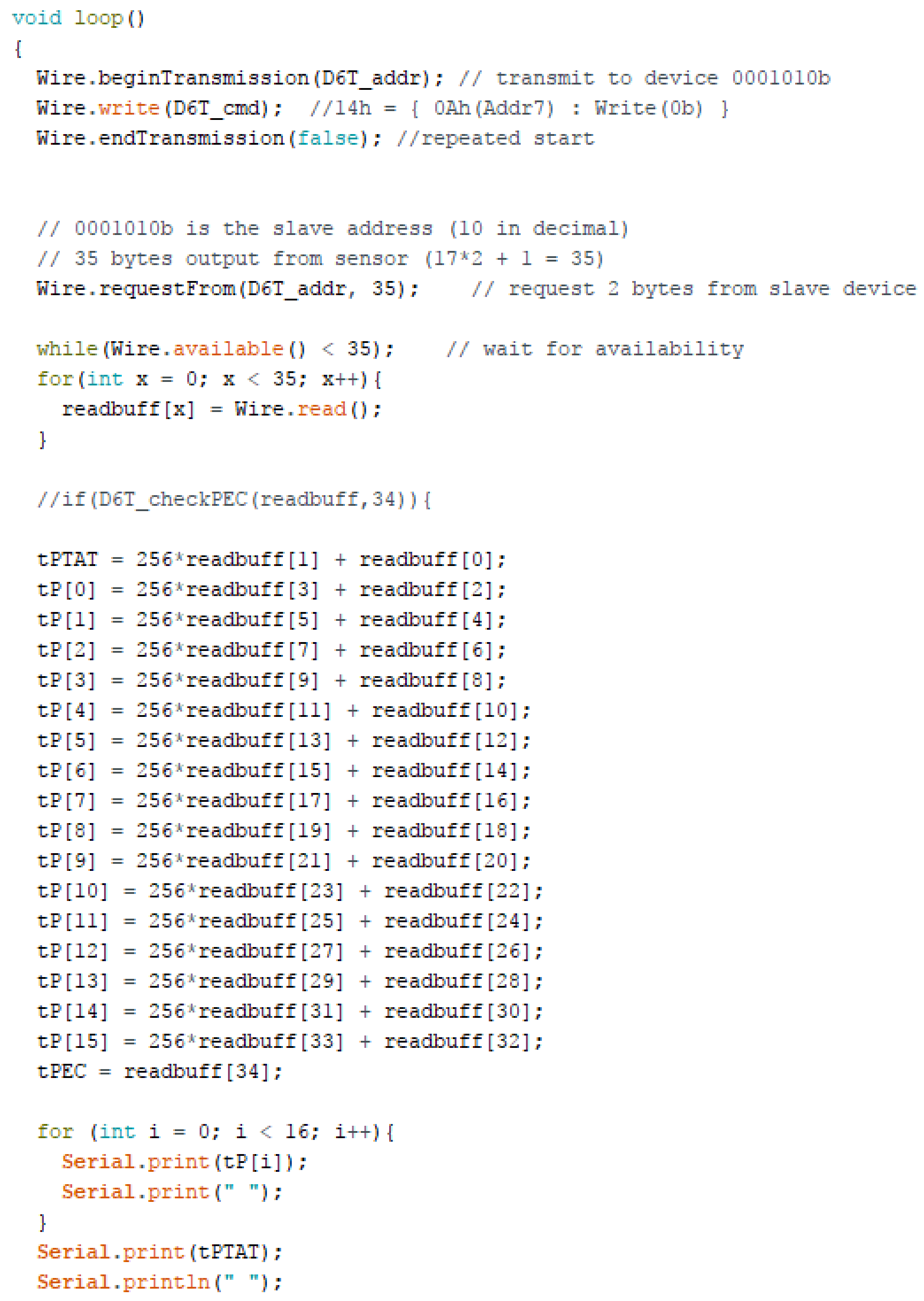
```
void connect()
{
  char hostname[] = "iot.eclipse.org";
  int port = 1883;
  sprintf(printbuf, "Connecting to %s:%d\n", hostname, port);
  Serial.print(printbuf);
  int rc = ipstack.connect(hostname, port);
  if (rc != 1)
  {
    sprintf(printbuf, "rc from TCP connect is %d\n", rc);
    Serial.print(printbuf);
  }

  Serial.println("MQTT connecting");
  MQTTPacket_connectData data = MQTTPacket_connectData_initializer;
  data.MQTTVersion = 3;
  data.clientID.cstring = (char*)"uvafoutier";
  rc = client.connect(data);
  if (rc != 0)
  {
    sprintf(printbuf, "rc from MQTT connect is %d\n", rc);
    Serial.print(printbuf);
  }
  Serial.println("MQTT connected");

  rc = client.subscribe(topic, MQTT::QOS2, messageArrived);
  if (rc != 0)
  {
    sprintf(printbuf, "rc from MQTT subscribe is %d\n", rc);
    Serial.print(printbuf);
  }
  Serial.println("MQTT subscribed");
}
```

Figure 20:connection function



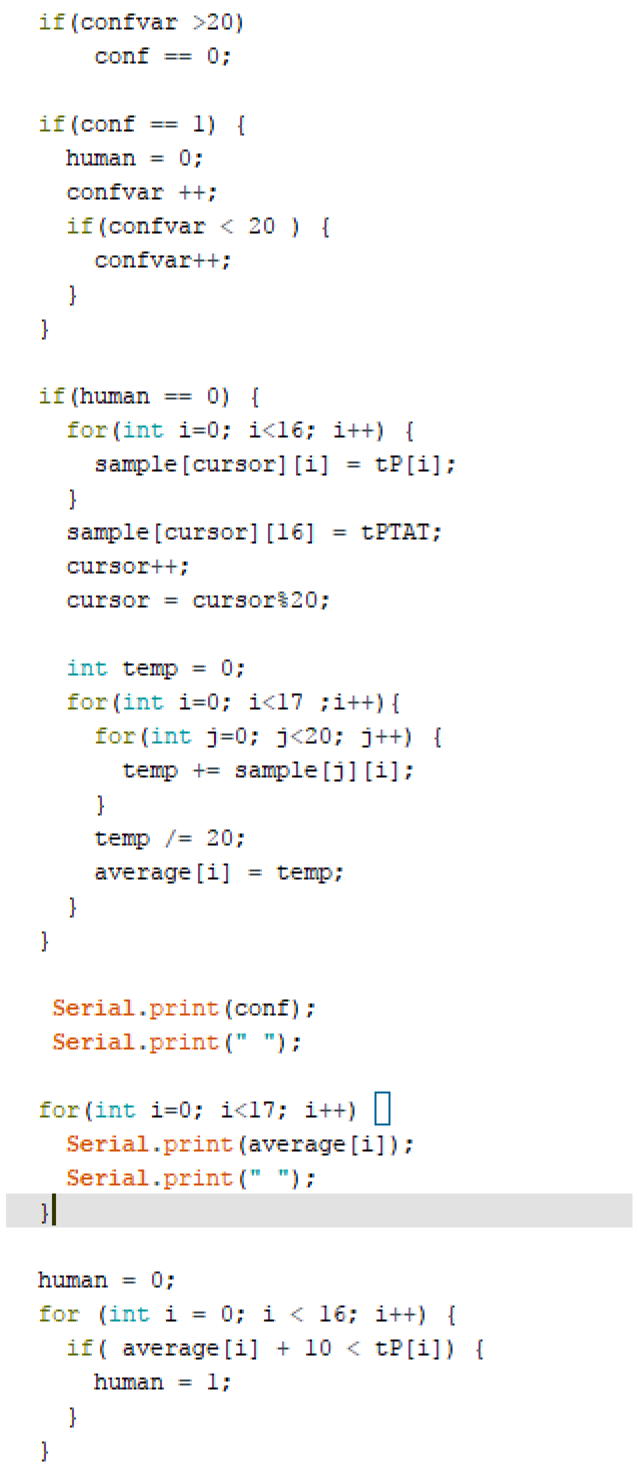
```
void setup()
{
  Serial.begin(115200);
  // attempt to connect to Wifi network:
  Serial.print("Attempting to connect to Network named: ");
  // print the network name (SSID);
  Serial.println(ssid);
  // Connect to WPA/WPA2 network. Change this line if using open or WEP network:
  WiFi.begin(ssid,password);
  while ( WiFi.status() != WL_CONNECTED) {
    // print dots while we wait to connect
    Serial.print(".");
    delay(300);

  //Serial.begin(9600);  // start serial for output 9600 bytes per s
  Serial.println("start");
  Wire.begin();        // join i2c bus (address optional for master)
  }

  Serial.println("\nYou're connected to the network");
  Serial.println("Waiting for an ip address");

  while (WiFi.localIP() == INADDR_NONE) {
    // print dots while we wait for an ip addresss
    Serial.print(".");
    delay(300);
  }

  Serial.println("\nIP Address obtained");
  // We are connected and have an IP address.
  Serial.println(WiFi.localIP());

  Serial.println("MQTT Hello example");
  connect();

  for(int i=0; i<20; i++) {
    for(int j=0; j<17; j++) {
      sample[i][j] = 0;
    }
  }
  for(int i=0; i<17; i++) {
    average[i] = 0;
  }
}
```

Figure 21:setup function, initiate Wi-Fi connection



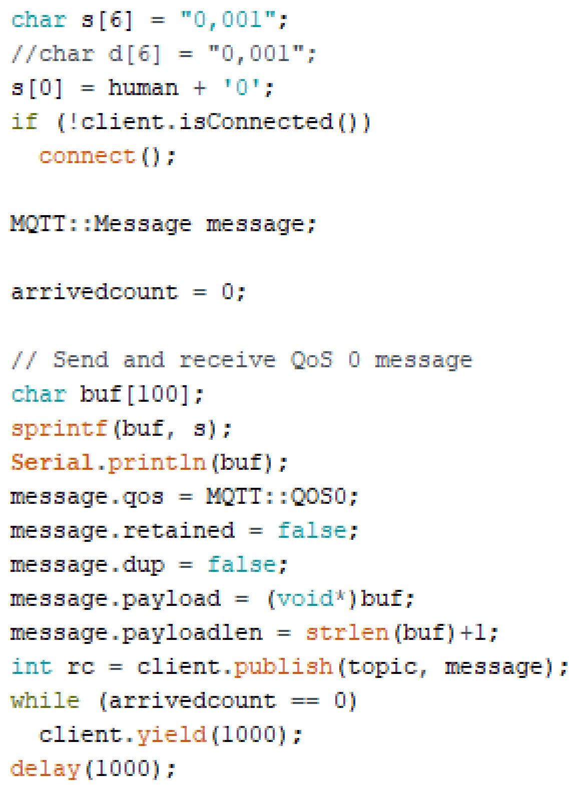
```
void loop()
{
  Wire.beginTransmission(D6T_addr); // transmit to device 0001010b
  Wire.write(D6T_cmd);  //14h = { 0Ah(Addr7) : Write(0b) }
  Wire.endTransmission(false); //repeated start


  // 0001010b is the slave address (10 in decimal)
  // 35 bytes output from sensor (17*2 + 1 = 35)
  Wire.requestFrom(D6T_addr, 35);    // request 2 bytes from slave device

  while(Wire.available() < 35);    // wait for availability
  for(int x = 0; x < 35; x++){
    readbuff[x] = Wire.read();
  }

  //if(D6T_checkPEC(readbuff,34)){

  tPTAT = 256*readbuff[1] + readbuff[0];
  tP[0] = 256*readbuff[3] + readbuff[2];
  tP[1] = 256*readbuff[5] + readbuff[4];
  tP[2] = 256*readbuff[7] + readbuff[6];
  tP[3] = 256*readbuff[9] + readbuff[8];
  tP[4] = 256*readbuff[11] + readbuff[10];
  tP[5] = 256*readbuff[13] + readbuff[12];
  tP[6] = 256*readbuff[15] + readbuff[14];
  tP[7] = 256*readbuff[17] + readbuff[16];
  tP[8] = 256*readbuff[19] + readbuff[18];
  tP[9] = 256*readbuff[21] + readbuff[20];
  tP[10] = 256*readbuff[23] + readbuff[22];
  tP[11] = 256*readbuff[25] + readbuff[24];
  tP[12] = 256*readbuff[27] + readbuff[26];
  tP[13] = 256*readbuff[29] + readbuff[28];
  tP[14] = 256*readbuff[31] + readbuff[30];
  tP[15] = 256*readbuff[33] + readbuff[32];
  tPEC = readbuff[34];

  for (int i = 0; i < 16; i++){
    Serial.print(tP[i]);
    Serial.print(" ");
  }
  Serial.print(tPTAT);
  Serial.println(" ");
```

Figure 22:loop to collect data and send to broker



```
if(confvar >20)
    conf == 0;

if(conf == 1) {
  human = 0;
  confvar ++;
  if(confvar < 20 ) {
    confvar++;
  }
}

if(human == 0) {
  for(int i=0; i<16; i++) {
    sample[cursor][i] = tP[i];
  }
  sample[cursor][16] = tPTAT;
  cursor++;
  cursor = cursor%20;

  int temp = 0;
  for(int i=0; i<17 ;i++){
    for(int j=0; j<20; j++) {
      temp += sample[j][i];
    }
    temp /= 20;
    average[i] = temp;
  }
}

 Serial.print(conf);
 Serial.print(" ");

for(int i=0; i<17; i++) {
  Serial.print(average[i]);
  Serial.print(" ");
}

human = 0;
for (int i = 0; i < 16; i++) {
  if( average[i] + 10 < tP[i]) {
    human = 1;
  }
}
```

Figure 23:human detection algorithm



```
char s[6] = "0,001";
//char d[6] = "0,001";
s[0] = human + '0';
if (!client.isConnected())
  connect();

MQTT::Message message;

arrivedcount = 0;

// Send and receive QoS 0 message
char buf[100];
sprintf(buf, s);
Serial.println(buf);
message.qos = MQTT::QOS0;
message.retained = false;
message.dup = false;
message.payload = (void*)buf;
message.payloadlen = strlen(buf)+1;
int rc = client.publish(topic, message);
while (arrivedcount == 0)
  client.yield(1000);
delay(1000);
```

Figure 24:MQTT sending routine