Projets Industriels – Elec4

Année scolaire 2017-2018

Rapport de Projet

" Bee Connect, La ruche connectée"

" Version 1"

Etudiants : Dihia Bitam, Qingna Xue, Axel Le Bourhis

Encadrant : Pascal Masson

THANKS

The BeeConnect team would like to address its special thanks to Mr Masson, the project tutor and professor at Polytech, for his advices, his follow-up and his trust.

Introduction

BeeConnect is a project that is part of our teaching in the second semester of our 4th year in electronics studies. In a few words, it consists in setting up a system which will allow users to monitor a beehive.

Before tackling the technical aspect of this project, it is important to know its interest and what could we benefit from it.

It is known that bees are crucial elements of our environment. They are essential for biodiversity and the development of agriculture. In fact, they pollinate about 80% of plant species, which contributes to the survival, the evolution and the production of plant species. The decline of bees population is mainly because of pesticides, disease, parasites, and poor weather due to global warming. The monitoring of a beehive can really help prevent loss and keep track of the bee's health.

Our goal is to build, through a series of sensors and warning indicators, a connected hive. The data collected will be processed and sent to a dedicated website, where users can see the activity of the bees during the day/night depending on the influence of some climate factors such as temperature, humidity, brightess, air pressure… The webpage will also indicate the variation of the hive's weight, the number of bees and show a real time video of the hive.

This report will treat of how went our projet. We'll start with the specifications we had to meet, we'll continue with our achievement and failures with the sensors and how we dealt with data sending over the internet. Then we treat of the physical implementation of the sensors and the system itself. Finally we will end with how we planned our project, what is its actual status and what we would like to do in the future.

Chapter I : Specifications

The goal of the project is to make a smart beehive. The beehive will have to provide information like temperature, humidity, atmospheric pressure, weight and brightness. These data will be sent to a webpage and will be available for everyone.

All the data will be processed by an Arduino or a STM Nucleo card. We will use WiFi to send the data to the website. We also will have to use IP cameras to send pictures of the beehive to the webpage so everyone can see it in real time.

We have to remember that the system is outside, so we'll have to keep it safe from weather. We don't have any constraints about the energy consumption as the project is a first experience and we just need to see if it's feasible.

Here are the main functions as well as the constraints functions which give a first overall view of the project:

FP1: To measure temperature and humidity

FP2: To measure the weight of the hive

FP3: To measure the brightness

FP4: To measure the atmospheric pressure

FP5: Counting the bees going in and out

FP6: Taking pictures of the beehive

FP7: Collecting data coming from the sensors on a website

FP8: Data processing with Arduino or STM Nucleo

FC1: To make sure any meteorological event do not disturb the electronic system

FC2: To take the least place possible

FC3: To minimize the noise creating by the system in order to avoid disturbing the bees

FC4: To be able to create the system with the budget allowed by the school

FC5: To take into consideration the delivering time of the components

Chapter II : Sensors : achievements and failures

II.1 - Weight sensor

II.1.1 How to connect four weight sensors?

For weight sensors, the main problem is to study their connection methods.

The structure of our weight sensors is four-wire and full bridge. We first considered using existing components to connect the four sensors. So we searched for available components and just found that the combiner can be used to connect weights sensors, but it is only applicable to three-wire and half bridge strain gauges. So this solution is not workable.

So, we solved this problem from the basic principle of the sensor.

II.1.1.1 wheatstone bridge

A wheatstone bridge is a configuration of four resistors with a known voltage applied like this:

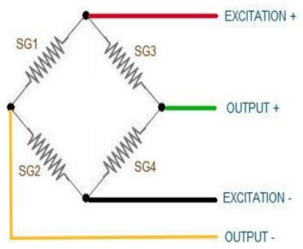


Figure II.1.1.1 wheatstone bridge

Vin is a known constant voltage, and the resulting **Vout** is measured. If R1/R2=R3/R4 then **Vout** is 0, but if there is a change to the value of one of the resistors, Vout will have a resulting change that can be measured and is governed by the following equation using Ohm's law:

$$Vout = [(R3/(R3+R4) - R2/(R1+R2))] * Vin$$

We know that there is a circuit called Wheatstone bridge inside each sensor. When the sensor is stressed, the internal resistance of the circuit will change accordingly, which will cause the output voltage value.

Since the sum of the four sensor forces is the sum of the four voltage values, four sensors can be used in parallel.

II.1.1.2 Using of Analog-To-Digital Converter-HX711

Due to the small resulting voltage, there is a large error if it is directly converted to actual weight. So we use HX711 analog-to-digital converter, it has two advantages to reduce the output error.

- Turn analog voltage into digital voltage to reduce electromagnetic interference
- linear gain in order to amplify the voltage

II.1.1.3 Solution of connection

Therefore, we found two methods to build the connection.

Solution 1:

To save material costs, the number of components used should be reduced as much as possible. Because one HX711 can carry up to two weight sensors mostly, four sensors require two HX711. So, we use 2 HX711 AD module and 4 sensors.

The detailed connection method is that two sensors are connected in parallel to one HX711 by connecting the same color line on the breadboard, for example, the white line is connected with the white line, and the green line to green line. The following figure can shows the display described:

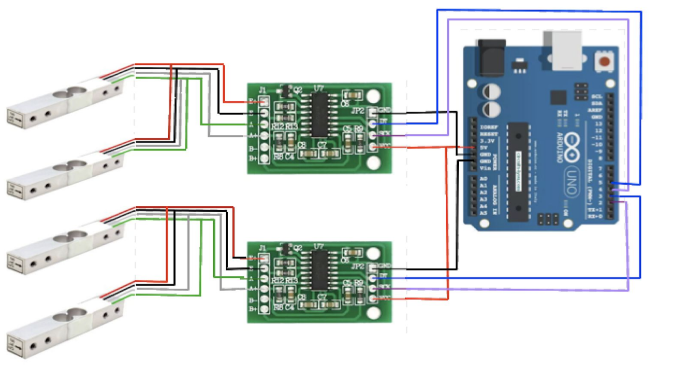


Figure II.1.1.3 Connection with 2 HX711

Solution 2:

If solution 1 is not workable, we can connect 4 sensors independently by using 4 HX711 AD module. Each sensor connected to the Arduino via a HX711. And the connection method is very simple, only need the corresponding pin of sensor and HX711 are connected.

II.1.2 Relationship between the digital voltage and the actual weight

In order to get the actual weight of the object, we should find the relationship between the actual weight and the digital voltage after the HX711 amplification.There are already a large number of HX711 libraries that can be used on the network. Through studying one of the quality libraries, we found a linear relationship:

$$Y=aX+b$$

- Y: Value of current output voltage
- X: Actual weight
- a: Correction coefficient
- b: Offset of HX711

The problem is how to measure the 2 factors a and b. For this, we thought about two solutions.

Solution 1: Obtain the corresponding **Y** and **X** values from two objects of known weight, then solve **a** and **b.**

Solution 2: Starting from the HX711 library, the tare() function can automatically confirm the value of **b**. Simply use another object of known weight to obtain the corresponding Y and X values to find the value of **a**.

II.1.3 Test protocol

II.1.3.1 Sensor Force Study

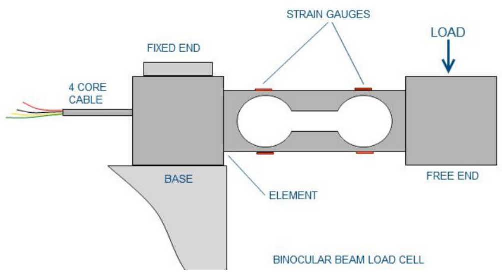


Figure II.1.3.1 Sensor force

The condition for the correct force of the sensor is that one end is fixed and the other end is pressurized. Before installation, we can only roughly estimate the performance of the sensor.

II.1.3.2 Test procedure

Solution 1:
We choose the solution 1 of connection. We know that a HX711 AD analog-to-digital converter corresponds to a HX711 library variable, because there are two HX711, so we need to define two HX711 variables. Based on the existing code found, we can easily write the test code.

In order to ensure balanced force, the four sensors are placed at equal distances horizontally, and we use an electronic scale to measure an object, whose weight is applied to observe **a** and **b** readings. Since we connect two HX711 modules, each HX711 corresponds to a linear relationship (Y1=a1*X1+b1, Y2=a2*X2+b2). Under the condition of force balance of each sensor, we should ensure a1=a2 and b1=b2.

After establishing the connection, by observing the Arduino serial monitor, we found that the actual situation does not match the ideal situation(a1 ≠ a2, b1 ≠ b2).And there are big differences, such as b1_average = 4458000 and b2_average = 1641000 after 5 measurements.

Solution 2:
The second solution's code is not much different from the previous one. Only four HX711 variables need to be defined to obtain four linear correspondences. Test in the same way, this time we can successfully make a1=a2, b1=b2, which provides guarantee for subsequent installation.

II.1.3.3 Exploration of problem

It took us a long time to explore the cause of the problem. Under the condition of ensuring the balance of force again, we found that this kind of connection would cause great interference between sensors.

- Specific connection as shown below :

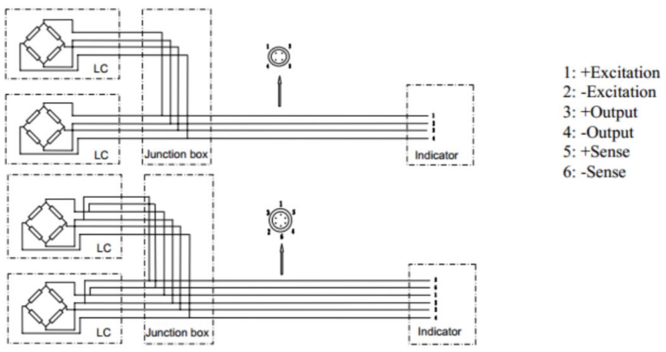


Figure II.1.3.3 Load cell cabling

Load cells should be electrically connected in such a way that the signal (output) lines, excitation (power supply) and sense (when present) lines are in parallel.

The above figures represent the two basic configurations, using four-wire (sense) load cells. With 4-wire extension cables the load cell output should be connected to pairs of diagonally opposite wires. Sometimes it is necessary to trim the output of each individual load cell to avoid corner load differences, which are caused by:

- The parallel connection. Each load cell will be loaded with the resistance of the other load cells. As a result, the individual load cell output tolerances will be increased by the individual output resistance tolerance.
- Unequal load distribution.

Trimming can be done by placing resistors or variable resistors into the excitation (Excitation trim) or output (signal trim) lines.

Reference: http://www.marco-iw.nl/file/1355354080.2319WAfreS/2%20Loadcell%20cabling.pdf

Because this shielding method is complicated, we tried the second solution to ensure independent connection between sensors and avoid its mutual influence.

II.2 - Camera

According to the specification, the system must provide a camera so the user can see the actual beehive on the website, in real time.

II.2.1 Pictures or live video stream ?

The ideal option would be to stream the live video of the beehive recorded by our camera, but it would need a lot of resources we cannot provide with our little website. We decided to use a camera that can send pictures with FTP protocol directly to our web server. Then, the PHP script would get the most recent take and make the html page display it.

II.2.1 Implementation problems

The camera we chose is really easy to use on a common network. It can be accessed on a web browser and then configured. In reality, on Polytech's network, it's much harder to configure. The camera never succeed to connect to Polytech's WiFi, it only worked through ethernet connection, but in real situation, the camera must connect to a WiFi as we can't get an ethernet connection where the beehive will be. One of our member tried to fix this problem during more than 3 weeks, we then decided to give up on this feature for the moment.

II.3 - Temperature acquisition

II.3.1 Internal temperature

II.3.1.a Choice of the sensor

As we have already explained on our previous research work, the temperature inside the hive is not uniform. We notice a higher temperature in the center and immediately above the cluster compared to the low part and in other empty portions of the hive. That is because warm air rises.
We then proposed to place two temperature sensor in the hive to have a more accurate measurement of the internal temperature. But we soon realized that we did not need to know the internal temperature in all parts of the hive. It is the cluster that matters.
That is why, our final decision was to place only one sensor right in the middle of the hive.
We used the DS18B20 sensor. Specifically, a pre-wired and waterproofed version of the sensor which is more handy for measuring something far away, or in wet conditions which is the case inside our hive.

II.3.1.b Pin connection and wiring

Depending on the colours of the wires:

- **Red** (VCC) - VCC 5V of the Arduino/Nucleo board.
- **Yellow** (DATA) - Any digital pin (with a 4.7K resistor pulled high)
- **Black** (GND) - GND of the board.

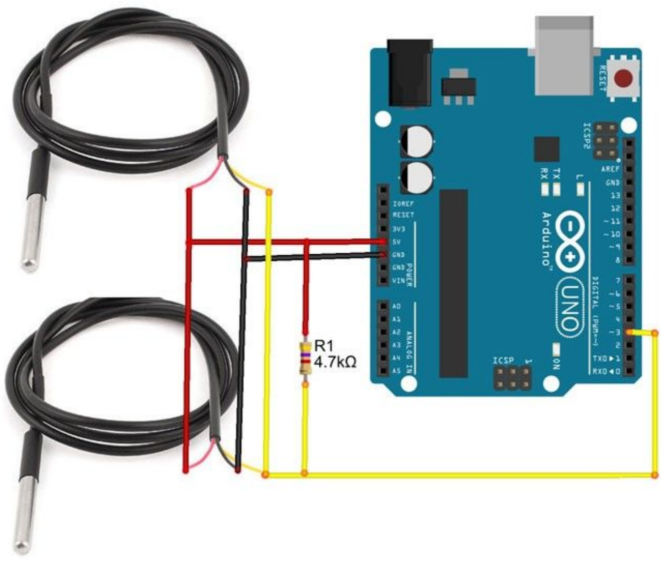


Figure II.3.1 Implementation of the DS18B20 with an Arduino Board

II.3.1 External temperature

Part of the specifications was to measure, in addition to the temperature inside the hive, the external temperature. Which is a good climate indicator influencing the activity of bees, especially their outings.
The sensor we wanted to implement for this acquisition was the BME280 because of its great accuracy in measurements and the additional information it provided on humidity and and atmospheric pressure.
Unfortunately, we did not implement this sensor on the hive since it was never delivered. We had another temperature sensor available (DHT11) but it was imprecise, especially for the humidity.

II.4 Brightness sensor

II.4.a Choice of the sensor

Besides the influence of the temperature on bees, we have explained how the brightness factor influence the outings of bees.

We chose the BH1750 sensor because it s suitable for obtaining the ambient light data and for its great performances regarding the types of measurements we want to make. It is possible to detect wide range of light intensity at high resolution.

We placed this sensor outside, on top of the hive in order to get the ambient light data.
The wiring of this sensor is very simple and can be found on our previous bibliographic report with all the details on its running and performances

II.4.b Pin connection and wiring

GND - connected to the GND of the Arduino/Nucleo board.
VCC - connected to the VCC 3.3V of the board.
ADDR - connected to the GND of the board.
SCL - connected to the Analog input 5 (A5).
SDA - connected to the Analog input 4 (A4).

Below is an example of an implementation of the sensor :

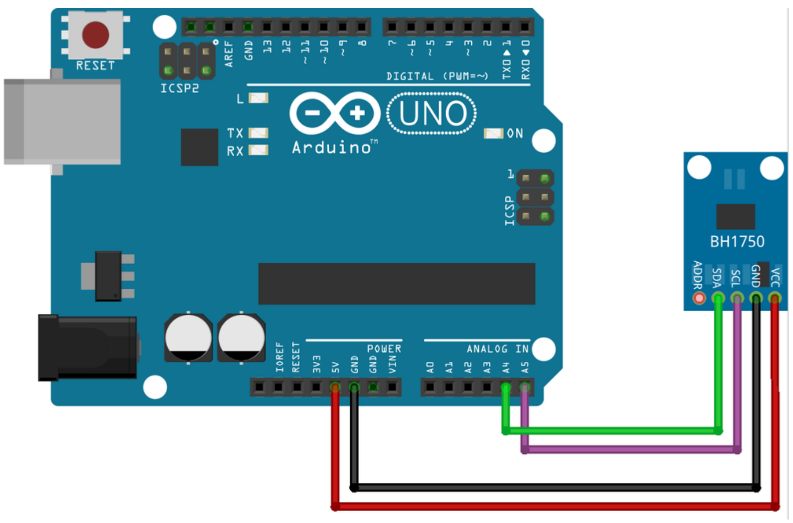


Figure II.4.b Implementation of the BH1750 with an Arduino board

II.5 Test protocol

These sensors being premade and coming from the market. We already have informations on theirs performances and limits. The tests we made were here to confirm the specifications of the products.

The DS18B20 was tested by putting another temperature sensor that we had available (the DHT11) and measuring at the same time, with both sensors, the temperature around the room: on the shaded part of the room and in front of the window where the sun was passing through. We also tested on cold bottle of water and on our skin. We noticed that the temperature acquired by both sensors was about the same with 2°C to 3°C difference due to the impreciseness of the DHT11 sensor.
We also submerged the DS18B20 in a glass of water and ran the tests above one more time to make sure of the waterproofness of the sensor. The results of the second test were the same as the previous one.

Since we did not have another brightness sensor to run a comparison test, the BH1750 was tested by acquiring multiple values of light levels from known environments. We therefore made the table below showing results from our sensor and from internet documentation.

Surfaces illuminated by	Illuminance (lux) - BH1750 sensor	Illuminance (lux) - Wikipedia
Starlight	0 (inside a box, complete dark)	0.0001
Room project lighting	450	320 – 500
Full daylight (not direct sun)	12,000 - 15,000	10,000 – 25,000
Direct sunlight	90,000 - 115,000	32,000 – 100,000

We can see that the sensor's results are accurate and reliable.

II.6 Bee counter

According to our specifications, the bee counter was supposed to be made with proximity capacitive sensors. The goal was to design multiple tubes placed at the entrance of the hive. On the top and bottom part of each tube, a plate of metal would be placed to create a capacitance. When a bee goes through a tube, the capacitance value changes affecting the input signal and therefore indicating that a bee has passed.

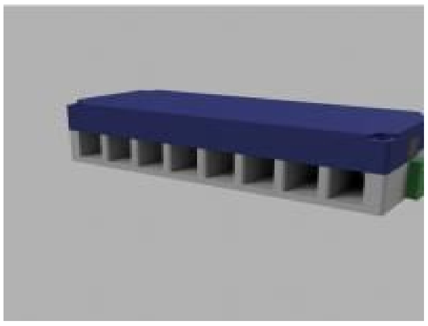


Figure II.6.1 3D view of a bee counter example

We would also have created the hand made capacitance on both edges of the tubes in order to distinguish an entrance or an outing of the bees and therefore prevent a double counting which is incorrect.

The mere fact that the capacitance changes is not particularly useful. To actually perform capacitive touch sensing, we need a circuit that can measure capacitance with enough accuracy to consistently identify the increase in capacitance caused by the presence of the bee.

The main problem we faced was the small value of the capacitance.
We wanted the tubes to have an entrance so that only bees can come in and not hornets which are a threat to honeybees. In order to do that, the entrance of the tubes must be around 16 mm according to a bee size. With that constraint, the capacitance value we calculated with the following equation was around the picofarad.

$$C = \frac{\mathcal{E}.A}{d} = \frac{k.\mathcal{E}0.A}{d}$$

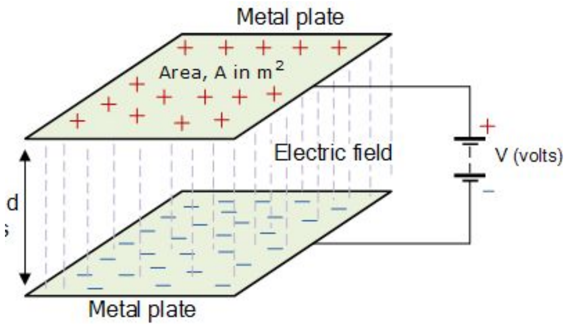


Figure II.6.2 Parallel metal plate capacitor

The capacitance of flat, parallel metallic plates of area **A** and separation between them **d.**
ε permittivity of space and **k** relative permittivity of the dielectric material between the plates (k=1 for free space, approximately =1 for air).

With such a small value, we can't exploit that data to perform any calculation to indicate the change of the input signal sent to the plates. In addition to that, since the entrances of the tubes are so small, and the system being based on the electric field between the plates, each tube could influence the one next to it.

After several week of research and trying to find a solution in order to use the small value of the capacitance, we chose to set aside the capacitive bee counter solution and try another type of counter, based on Infrared detection.
It consist on an IR LED and a phototransistor. The circuit used is simple and is shown below:

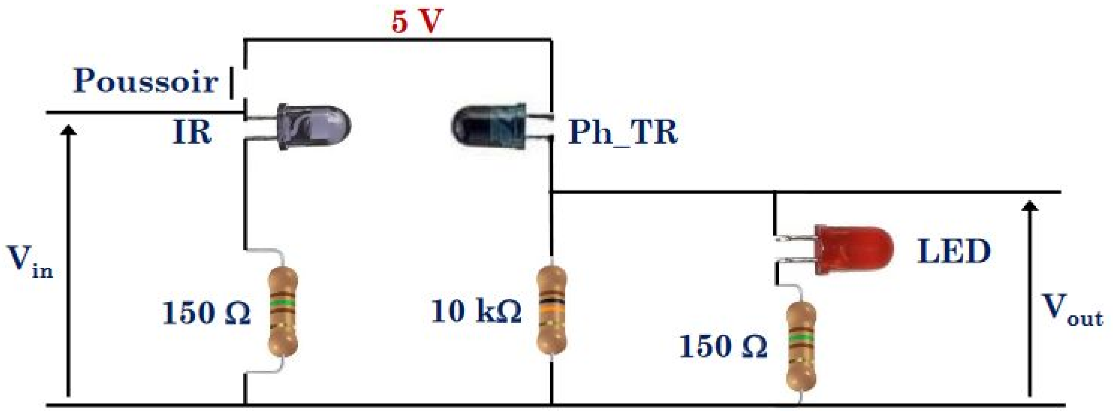


Figure II.6.3 Basic circuit for IR detection (M.Masson Arduino course)

We implemented this circuit on a breadboard in order to write the code that will count each passing of an object. Here we used our finger to emulate the crossing of a bee. We implemented this circuit in two identical copies to emulate, the edges of the tube so that we can differentiate when a bee goes in or out.
Since we had problems using two counters (one for each sensor) with the for loop in our code, we used the function millis() to get the time when a detection is made and then compare which sensor was triggered first.

The difficulty is to cover all the scenarios in our code which are many, for example, the first is to determine whether the bee is going in or out, if a bee goes through the first sensor but not the second, if two or more bees go through the sensors, if one bee goes in whilst another goes out, if its another insect…

Due to a lack of time and a lot of scenarios to cover, we unfortunately did not succeed on having an accurate counting. An additional work on this sensor would have probably shown some positive results regarding the counting, but it would not be precise regarding the scenarios mentioned above

Chapter II : Send data to the website

II.1 - What module and how to send data over internet ?

According to the project specifications, the system must be able to send the data collected by our sensors directly to a web server, so we can process it. There are many ways to do that. In our case, we firstly chose to use an WiFi module called ESP8266. This module can be wired to an Arduino card directly, but our tutor advised us to use a dedicated card that embeds directly a ESP8266, for more stability. But even with this type of card, the more the code was complicated, the less the module was stable, and we faced many crashes. Then, we made some research and decided to use the LoRa WiFi module ESP32 from Heltec, with OLED screen, because of its performances and stability. After some code adaptation, we saw that it was a good choice because it worked perfectly fine.

The code is pretty simple, as it simply makes the module connect to a WiFi host and send GET requests to a PHP script host on our website. The ESP32 receives data from the Arduino card and then send them. This was the first step of the web development of the project, be able to send and get data through the internet. The final goal is to store these data and display charts of each data type so we can see the evolution over time. The code is available in the dedicated section in annexe.

In fact, we had many problems to face. The first problem was to send our GET request through Polytech’s network. Indeed, this network is very secured and each device must log in with an account to be able to use the internet connection. But it was to complicated and unsecured to implement it in our code. We had to get a special access which would allow our device to bypass this authentication, by using its MAC address to authorize it to use the internet. So far, nothing complicated, but even with this bypass, our GET requests were unable to reach our website, even if our device could connect to the host. Then, we decided to not lose too much time on this part, and decided to use the connection from one of our phones to test if our system was working. Moreover, the delay to get responses from the IT service was a bit too long, and it would have stopped us in our progress.

The second problem was the web development itself. None of us was familiar with PHP so we had to learn on the job. We succeed to develop a test page to check if we get the data, but we haven’t be able to develop the final website with charts and data storage. Moreover, not having root access on the website made us wait from IT service even for simple tasks like creating a database. We decided to stay with our test page for the moment.
The PHP script is simple, it tests what data is received and then updates the files where data are written. This script is available in the dedicated section in annexe.

II.2 - Test protocol

The first step is to test if the PHP script is able to get data and display them in a .html page. To do that, we can simply use a PC and send GET requests to our website. This way, we know exactly what data we send so we know what should be displayed on the website. This first step was very useful as the script wasn’t perfect at the first try, but we improved it and fixed the errors. The final step is to let the ESP32 send its own GET requests to the website, we added the code necessary to trace what the ESP32 was sending, and checked what was received by the website. This way we’ve been able to test and validate our website and the request sending of the ESP32.

II.3 Security improvement

At the actual status of the project, the website works with HTTP. It works well for what we want to do but this protocol is not secured. Every request can be read by everyone, so everyone can know how we do our GET requests, so everyone can change what is displayed on the website. One solution would be to change the HTTP protocol by a HTTPS protocol, which encrypts the requests. We've included this security in our code and it worked but the school didn't provide a website which worked with HTTPS so we had to get back to HTTP protocol.

Chapter III : ESP32 and Arduino interfacing

This chapter will expose how the Arduino Uno and the ESP32 are interfaced, how do they communicate, and first of all why use 2 card.

III.1 - Why use 2 cards ?

In this our, 2 cards are used : an Arduino Uno and an ESP32 embedded on a Heltec card. The reason is simple, we need that our system is as stable as possible, because it's supposed to work all day long, we can't have a system that crashes every hour and needs to be rebooted by the user. After discussion with our tutor about the ESP8266, we decided to use the ESP32 embedded on it's own card.

III.2 - Make them communicate

Now that there are 2 cards in our beehive, we need them to communicate. Indeed, the main role of the Arduino is to process data received by all our sensors, where the main role of the ESP32 is to send these data to the website. Then, we had to imagine a way to make them communicate in an efficient way so we can prevent errors. The simpler way to do that, was to create a special TX/RX protocol both cards can understand and process.

The operation is simple, we want our Arduino to send messages to the ESP32 which contains a data measured by a sensor. Indeed, each message will contain only one data, for example the temperature measured. The Arduino will send these data one after the other in loop, as strings of characters.
So now the Arduino is capable of sending data through a TX/RX transmission. But since the Arduino sends each data one after the other, how the ESP32 can detect the start of a message, and its end ? We decided to make the Arduino send a start character and an end character. This way, the message would be encapsulated and its start and its end would be identifiable by the ESP32.
One problem is still existing. The Arduino can send data, and the ESP32 can identify the start and the end of the message, but how can it detect which type of data is received ? Is it a temperature, a humidity ? It can't know this at the moment. The solution we chose is the add another informative character, a data type character, that would make the ESP32 be able to know what data has been received.
Now our protocol is finally designed, and looks like this :

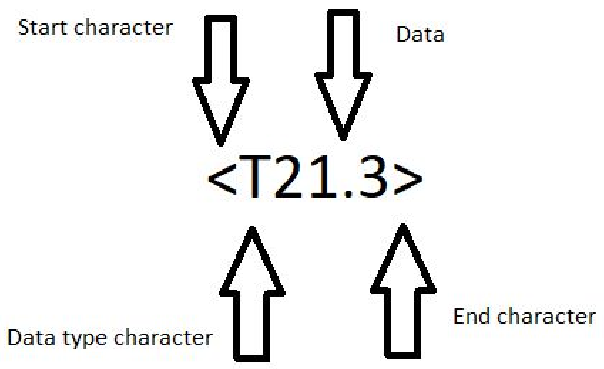


Figure III.2.1 Message example using our protocol

As it can be seen here, the message contains a temperature data, and its value is 21.3 degrees. Thanks to the code we implemented in both cards, we are able to make them communicate and understand each other through this specific protocol. The code is available at the end of this document for more details.

III.3 - Test protocol

We'll see in this part how we have tested and validated our protocol and its implementation in C. The operation is simple and divided in different step. We first need to test if the code works on each card.

For the Arduino, we need to check if the messages it sends are correct. We add the code necessary so we can get the message the Arduino is sending through USB connection with our PC. The Arduino IDE provides a serial monitor which allow us to receive these messages. We make su that the code will send a predetermined message so we are able to know what message we should receive through serial monitor. This part has been quickly validated as it is a simple task for the Arduino.

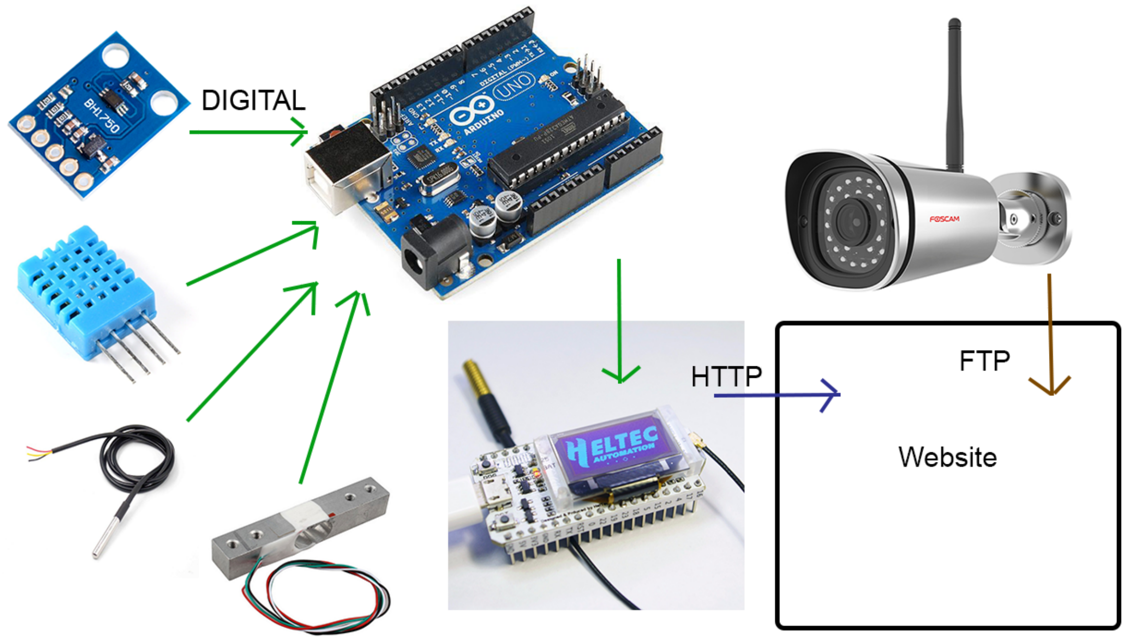
For the ESP32, we need to check if the message received has been correctly identified. We add the code necessary so we can fake the predetermined data reception and check if our algorithm processes it correctly. We use the serial monitor to check if the ESP32 identifies the start and end characters, the data type and finally the value sent. This part has been less quick to validate but we hadn't that much problem, everything has been fixed quickly.

Now that our 2 modules embeds working algorithm, we have to connect them and see if it's still working fine. We program the Arduino so it can send various data (with different data types) with fake value (for test purpose) and we connect it with the ESP32. We know exactly what the Arduino is sending through serial monitor, and we have to check if the ESP32 gets every data and processes it correctly. The main problem of this part was to find a good delay between each message because if we let the system run for a long time, it started to get corrupted data. After some test, we've been able to quickly find a perfect delay. We now can let the system run for hours without getting one error. The system is now tested and validated.

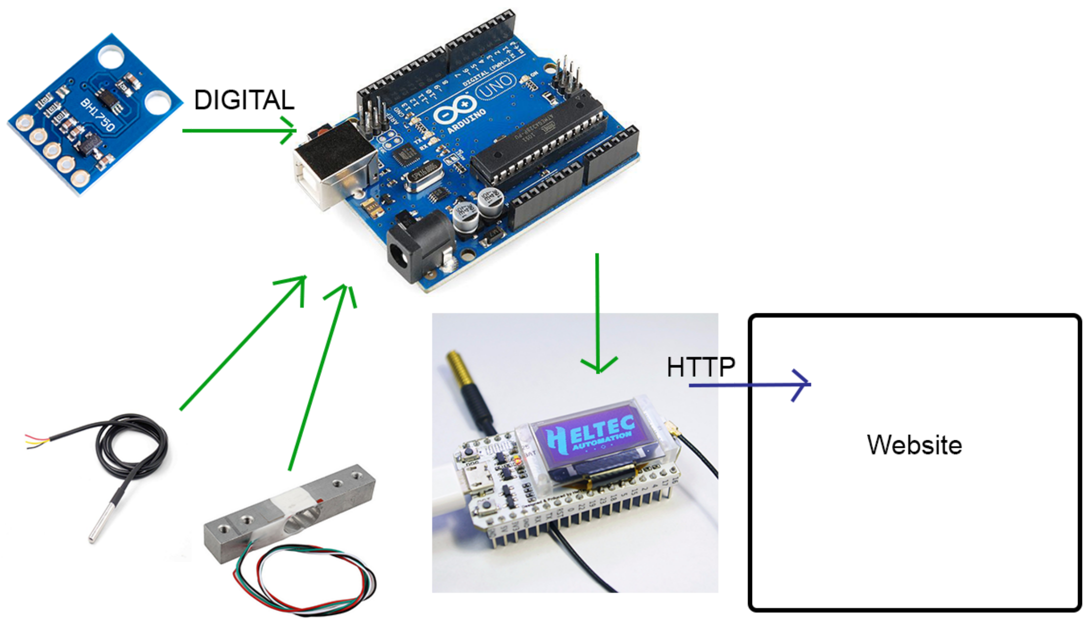
Chapter IV : Physical implementation of the system

IV.1 - Architecture

Here is the general Architecture of our project. Each square represents a function described in the first chapter of this report. This architecture is what the system will be at its final status.



At the moment the architecture of our system looks like this :



We can see that the system does not implement neither the camera nor the humidity sensors and pressure. But it implements the weight sensor, temperature, light and is capable of sending data to our website. For more details, please refer to the chapter VI.

IV.2 - The board

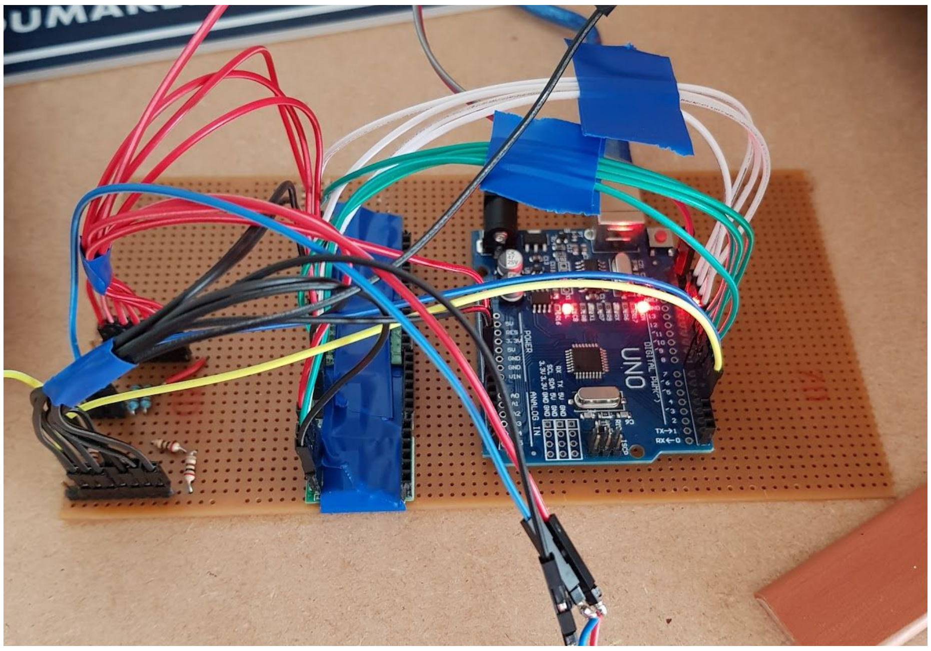


Figure IV.2.1 Board

The board is divided in 3 parts. On the left of the picture, we have a little circuit that brings together all the power cable for our sensors (+5V and GND) and some resistors necessary in order to use the temperature sensor and our TX/RX communication between the Arduino and ESP32. Then, on the center of the board, we have our 4 HX711 necessary in order to interface our weight sensors with our Arduino. finally, on the right side we have the Arduino board which welcomes all the sensors.
We can see that everything is built to be replaced easily. Indeed, this connected hive is only a prototype at the moment, this is not a final state, so we've built everything to be easy to replace in case of change.

IV.2.1 Intermediate circuit

Here is a detailed schematic for this side of the board. Like said just before, this circuit is here to power all the sensors and to implements necessary resistors.

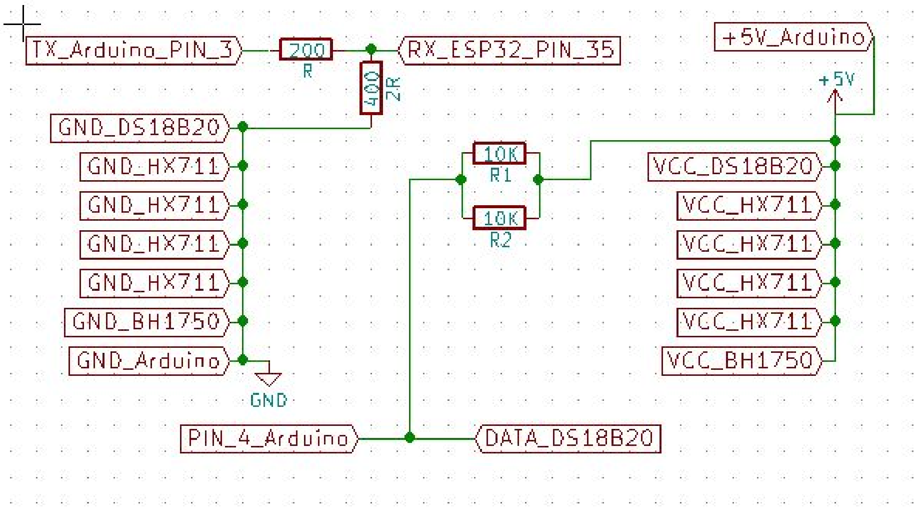
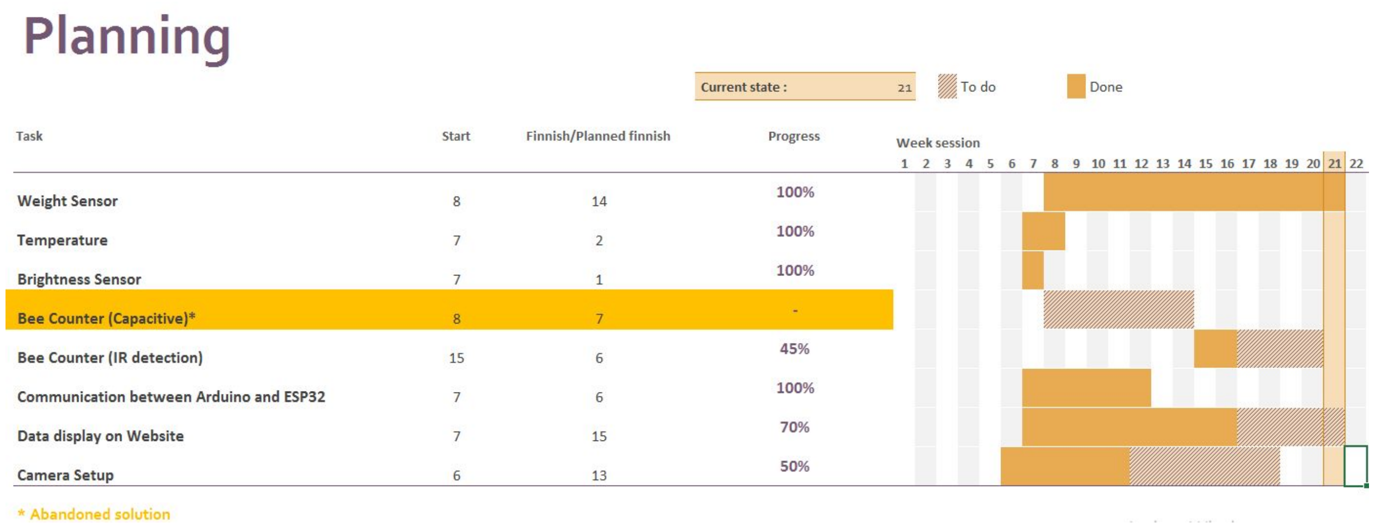


Figure IV.2.1.1 Detailed schematic

For the detailed mapping of the sensors on the Arduino, please refer to the dedicated section in annexe.

Chapter V : Project planning

As in any project, it is very important to have a planning during the lifecycle of our projects. Throughout the BeeConnect project, we tried to keep a record of all the tasks we have done during the sessions.
Before rushing to move ahead with execution or reporting results, we gathered all the tasks we needed to accomplish from the specifications we were given. Afterwards, we assigned each member of the team with two or three tasks to complete in an estimated time. Each member was free to set the timing and deadlines for their tasks.



Planning

Current state : 21 ▨ To do 🟧 Done

Task	Start	Finnish/Planned finnish	Progress
Weight Sensor	8	14	100%
Temperature	7	2	100%
Brightness Sensor	7	1	100%
Bee Counter (Capacitive)*	8	7	-
Bee Counter (IR detection)	15	6	45%
Communication between Arduino and ESP32	7	6	100%
Data display on Website	7	15	70%
Camera Setup	6	13	50%

* Abandoned solution

Figure V.1 Final project planning

Chapter VI : Actual project status

At the moment, most of the specifications are met, but not all of them. According to them, our system should collect data about the temperature, the humidity, the atmospheric pressure, the weight and finally the brightness. At the moment, the system is able to provide data about the temperature, the brightness and the weight. It doesn't implement the 2 other sensors simply because all the input of the Arduino Uno are full. A simple solution would be to use an Arduino Mega, which provides more pins.
Concerning the data sending over internet, the system is fully operational. The main problem for us is the Polytech's network because it filters too much things and it's hard to bypass everything. Moreover, our website is able to receive these data, but we need to develop a website that could store these data and process them in order to display different charts. Indeed, at actual state, the website can only display the last data which has been received.
Concerning the use of the STM32, we had to find the librairies adapted from Arduino in order to use our sensors with this card. Almost every sensors' libraries had been already adapted to this card but not the library of the DS18B20, the temperature sensor. We didn't have the time to adapt it so we decided to get back on the Arduino.

Also, we were not able to deliver the bee counter. We spent a large amount of time on the capacitive plates sensor solution which turned out to be not suitable for the hive giving the dimensions of the counter and what we wanted to detect (i.e the bees). Our work on this first counter has cost us a lot of time which impacted negatively on the global planning of the project.
Therefore, even if we had finally decided to focus on another implementation, based on infrared detection, we were not able to pass the tests and calibration of one sample of the counter, meaning one "theoretical tube entrance".

Finally, we neglected one aspect in the planning which is the physical implementation of the entire system on the hive. Our focus was more on trying to make the sensors work and the website up. We had several "side tasks" to accomplish in order to set up the smart hive such as: organizing all the circuitry on one board and designing a wood box that would later contain the Arduino board and the circuitry, placing the sensors in the right places, insulating wires, jumpers, sensors, holes on the hives…

This slackness taught us to be more careful about these side tasks in the future and to not neglect them as they are the difference between a prototype and the final product.

Conclusion

This report is a work in progress of an entire semester on the BeeConnect project. Our goal was to take a simple beehive and turn it into a smart hive that can provide users with information regarding its health and status.
We started by working on a bibliographic research in order to get more familiar with the project, setting the goals and exploring multiple solutions to chose the most suitable for the hive.

After that, we tackled the practical aspect of the development of our system which had many learning outcomes in both technical and management aspects.
Working in full autonomy allowed us to have a more precise idea of an engineer job, and a better understanding of the lifecycle of a project and the importance of planning and monitoring the tasks accomplished.

Finally, we wanted to express more our interest in the BeeConnect project and the desire of some of the team members to continue its development next year and achieve the specification we were not able to meet.

Annexes

I - Sensors mapping on Arduino

BH1750

SCL -> SCL
SDA -> SDA

HX711

	Pin number	Pin number	Pin number	Pin number
DT (Green)	5	6	7	8
SCK (White)	9	10	11	12

The white wire from a weight sensor goes to the pin A+. The green wire goes to the pin A-. VCC on E+, GND on E-.

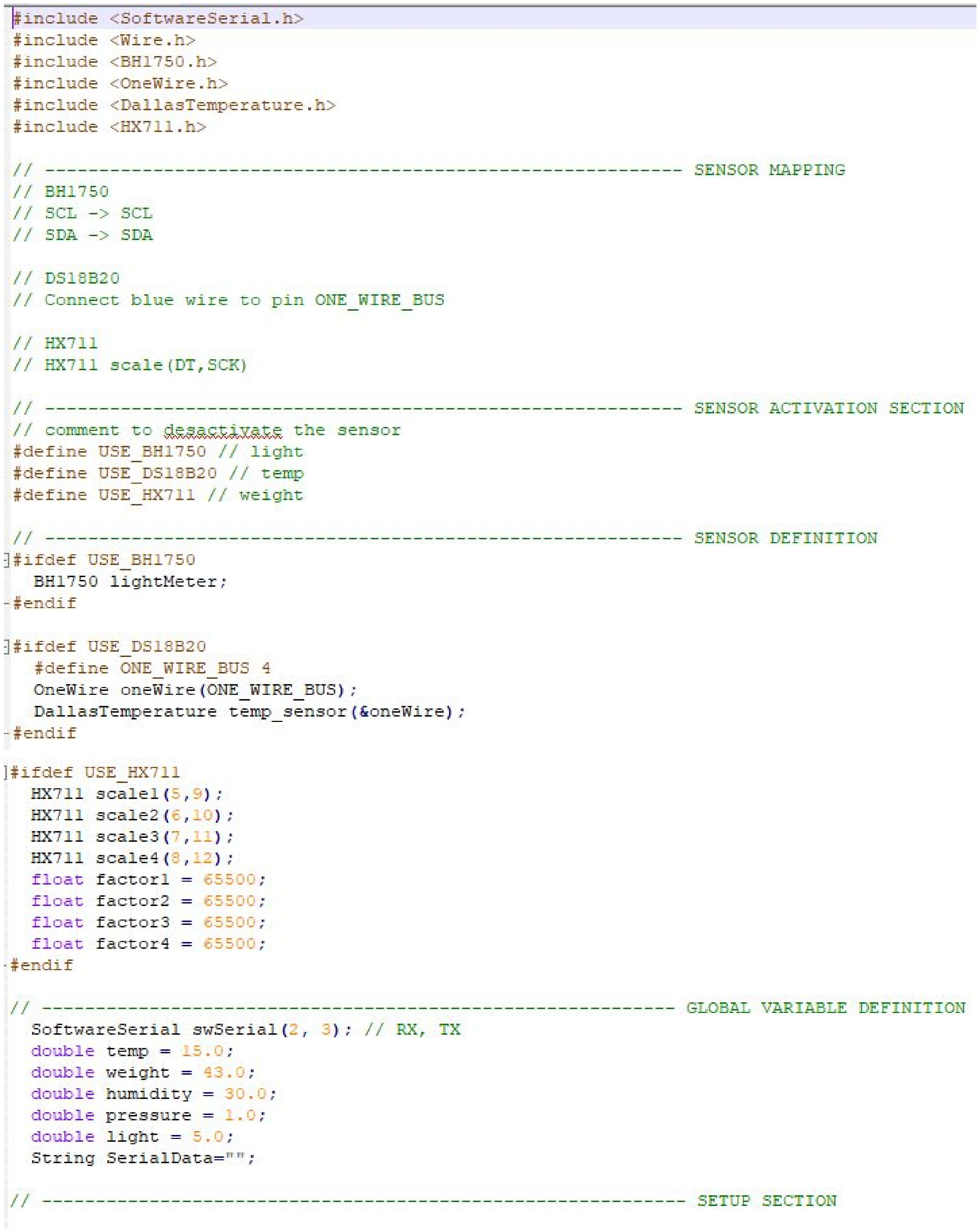
DS18B20

Data (blue) connected to intermediate circuit to the two parallel mounted resistors of 10K ohm (blue wire). The second blue wire goes to the pin 4 on the Arduino.

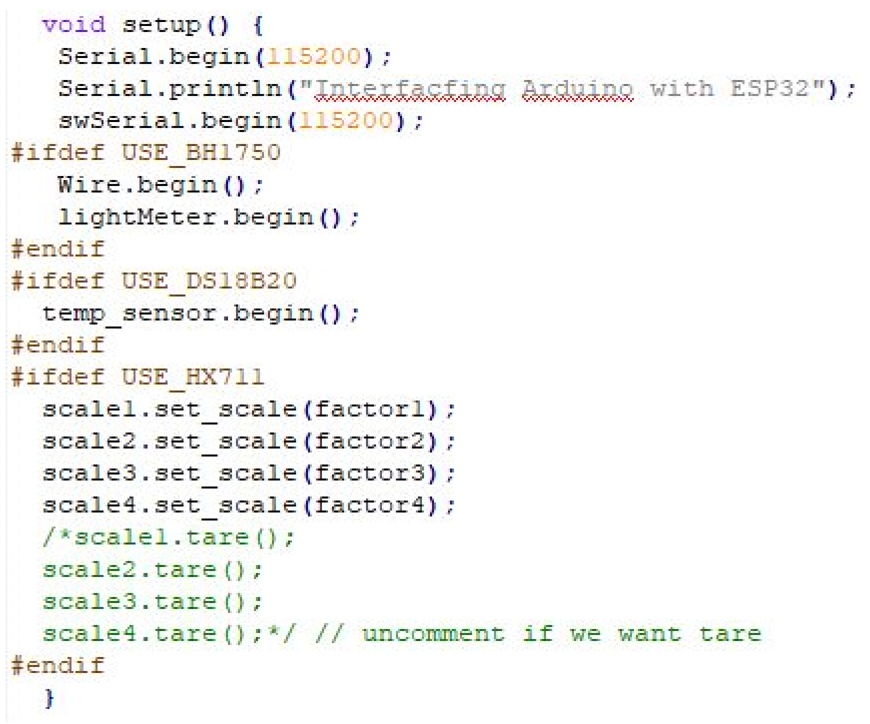
TX/RX Arduino/ESP32

Yellow wire goes from pin 3 (Arduino) to the intermediate circuit, the pin closer to GND rack. A second yellow cable goes from the pin above the first one, to the pin 34 of the ESP32.

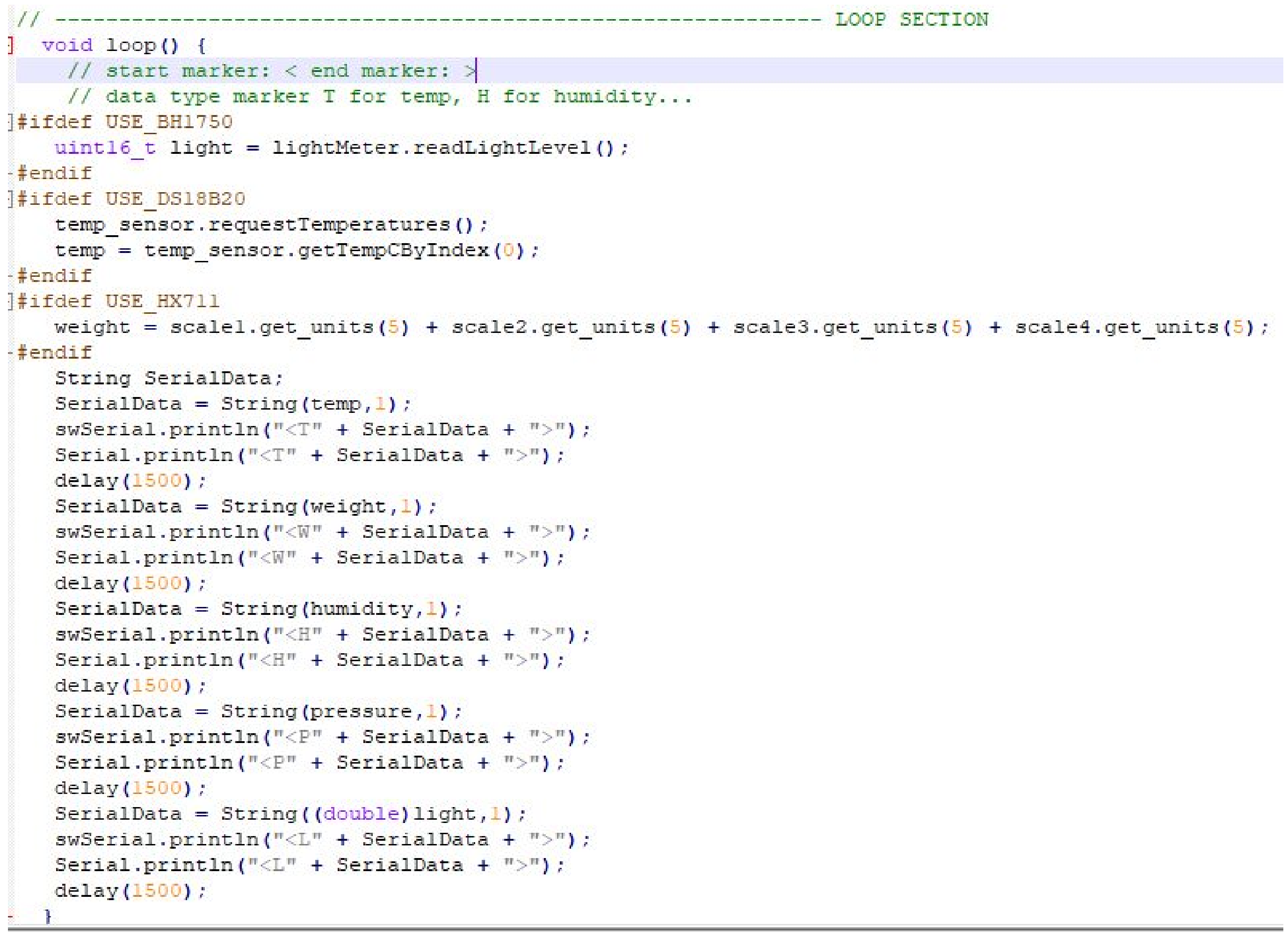
II - Embedded software on Arduino



```
#include <SoftwareSerial.h>
#include <Wire.h>
#include <BH1750.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <HX711.h>

// ------------------------------------------------------------ SENSOR MAPPING
// BH1750
// SCL -> SCL
// SDA -> SDA

// DS18B20
// Connect blue wire to pin ONE_WIRE_BUS

// HX711
// HX711 scale(DT,SCK)

// ------------------------------------------------------------ SENSOR ACTIVATION SECTION
// comment to desactivate the sensor
#define USE_BH1750 // light
#define USE_DS18B20 // temp
#define USE_HX711 // weight

// ------------------------------------------------------------ SENSOR DEFINITION
#ifdef USE_BH1750
  BH1750 lightMeter;
#endif

#ifdef USE_DS18B20
  #define ONE_WIRE_BUS 4
  OneWire oneWire(ONE_WIRE_BUS);
  DallasTemperature temp_sensor(&oneWire);
#endif

#ifdef USE_HX711
  HX711 scale1(5,9);
  HX711 scale2(6,10);
  HX711 scale3(7,11);
  HX711 scale4(8,12);
  float factor1 = 65500;
  float factor2 = 65500;
  float factor3 = 65500;
  float factor4 = 65500;
#endif

// ------------------------------------------------------------ GLOBAL VARIABLE DEFINITION
  SoftwareSerial swSerial(2, 3); // RX, TX
  double temp = 15.0;
  double weight = 43.0;
  double humidity = 30.0;
  double pressure = 1.0;
  double light = 5.0;
  String SerialData="";

// ------------------------------------------------------------ SETUP SECTION
```

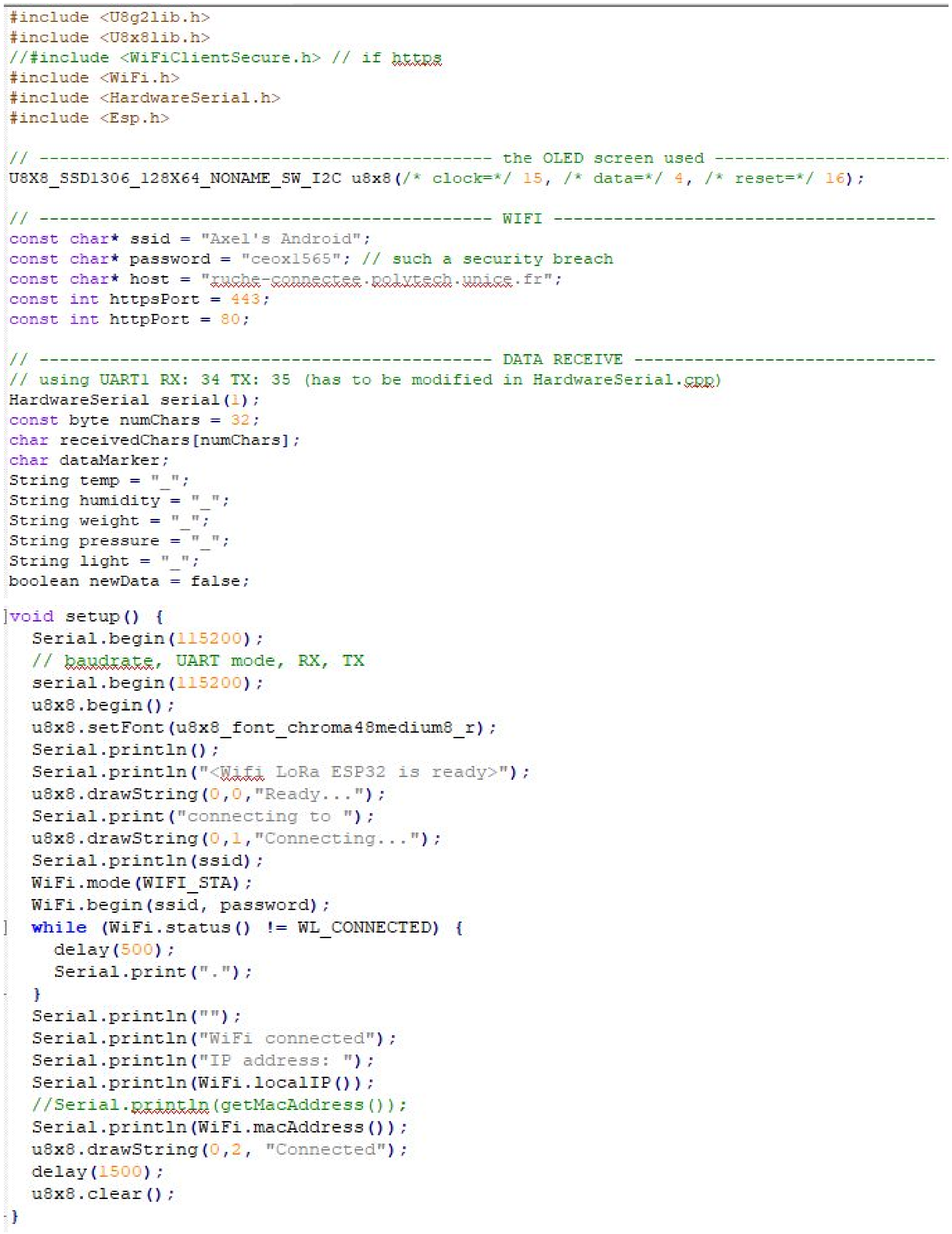


```
  void setup() {
   Serial.begin(115200);
   Serial.println("Interfacfing Arduino with ESP32");
   swSerial.begin(115200);
#ifdef USE_BH1750
   Wire.begin();
   lightMeter.begin();
#endif
#ifdef USE_DS18B20
  temp_sensor.begin();
#endif
#ifdef USE_HX711
  scale1.set_scale(factor1);
  scale2.set_scale(factor2);
  scale3.set_scale(factor3);
  scale4.set_scale(factor4);
  /*scale1.tare();
  scale2.tare();
  scale3.tare();
  scale4.tare();*/ // uncomment if we want tare
#endif
  }
```

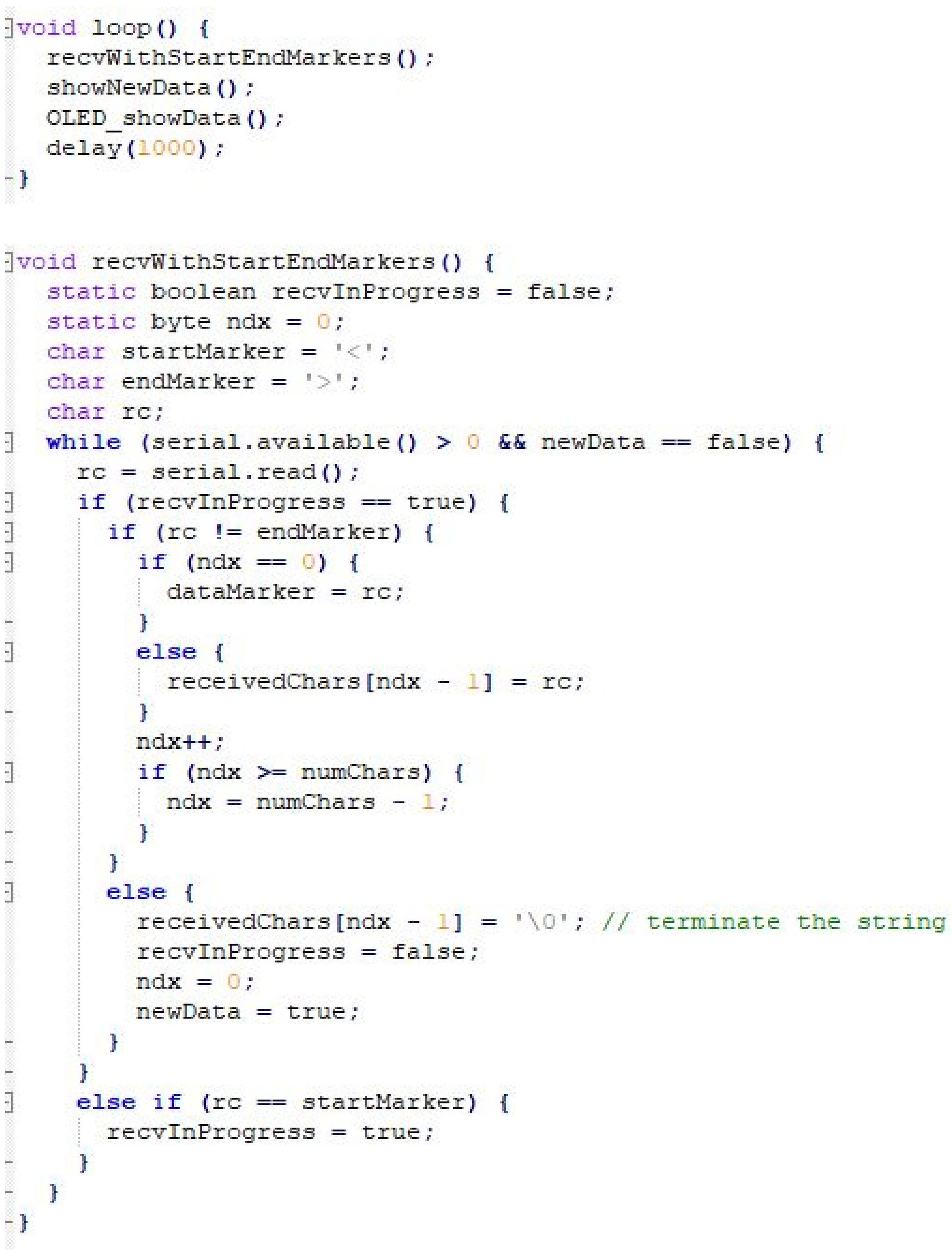


```
// ------------------------------------------------------------ LOOP SECTION
  void loop() {
    // start marker: < end marker: >
    // data type marker T for temp, H for humidity...
#ifdef USE_BH1750
   uint16_t light = lightMeter.readLightLevel();
#endif
#ifdef USE_DS18B20
   temp_sensor.requestTemperatures();
   temp = temp_sensor.getTempCByIndex(0);
#endif
#ifdef USE_HX711
   weight = scale1.get_units(5) + scale2.get_units(5) + scale3.get_units(5) + scale4.get_units(5);
#endif
   String SerialData;
   SerialData = String(temp,1);
   swSerial.println("<T" + SerialData + ">");
   Serial.println("<T" + SerialData + ">");
   delay(1500);
   SerialData = String(weight,1);
   swSerial.println("<W" + SerialData + ">");
   Serial.println("<W" + SerialData + ">");
   delay(1500);
   SerialData = String(humidity,1);
   swSerial.println("<H" + SerialData + ">");
   Serial.println("<H" + SerialData + ">");
   delay(1500);
   SerialData = String(pressure,1);
   swSerial.println("<P" + SerialData + ">");
   Serial.println("<P" + SerialData + ">");
   delay(1500);
   SerialData = String((double)light,1);
   swSerial.println("<L" + SerialData + ">");
   Serial.println("<L" + SerialData + ">");
   delay(1500);
  }
```

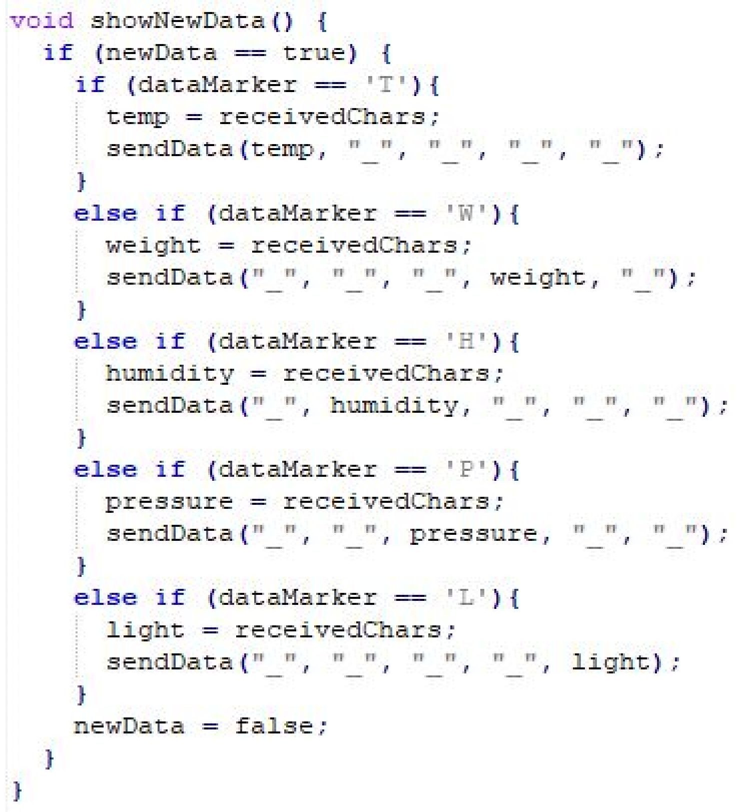
III - Embedded software on ESP32



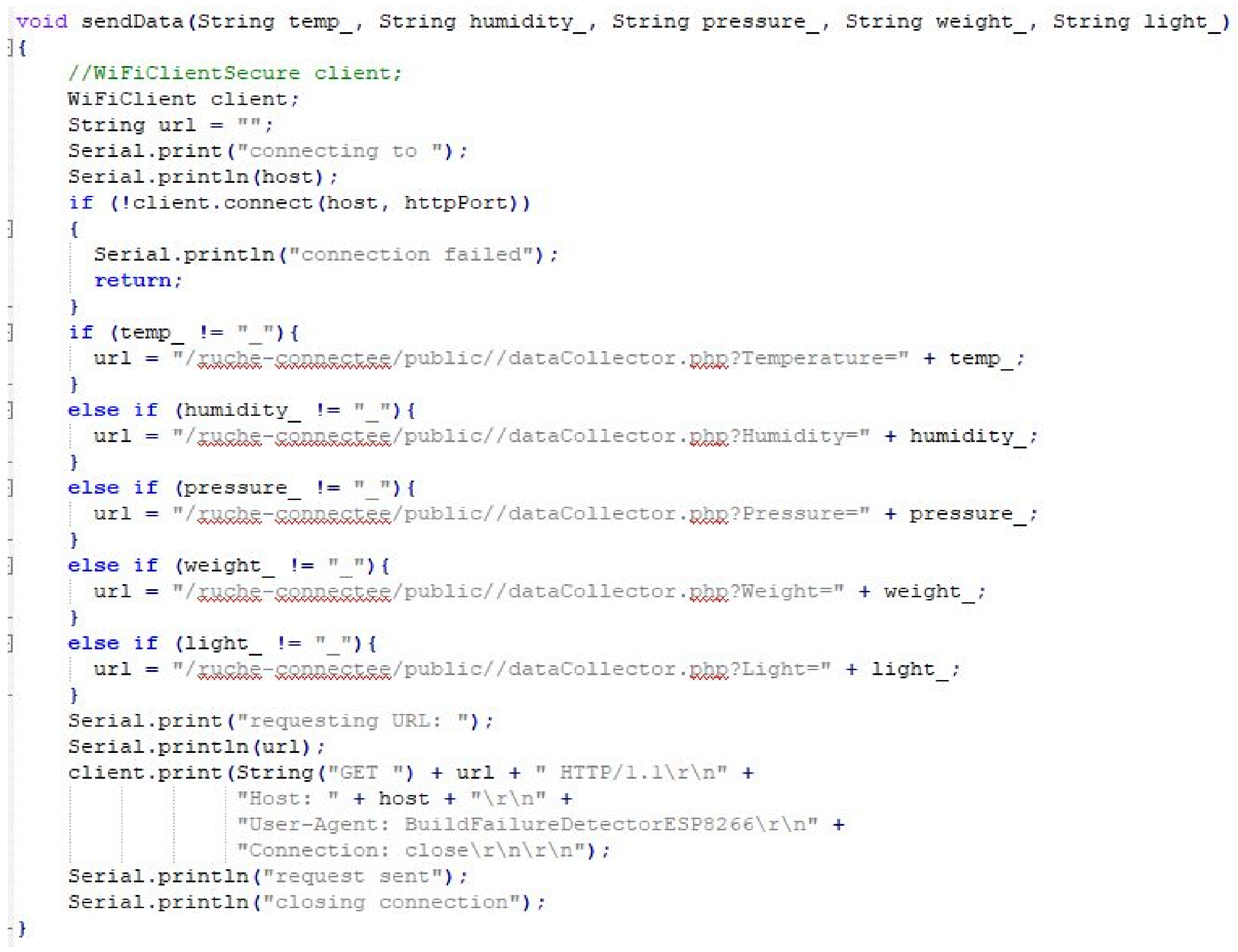
```
#include <U8g2lib.h>
#include <U8x8lib.h>
//#include <WiFiClientSecure.h> // if https
#include <WiFi.h>
#include <HardwareSerial.h>
#include <Esp.h>

// ---------------------------------------------- the OLED screen used -----------------------
U8X8_SSD1306_128X64_NONAME_SW_I2C u8x8(/* clock=*/ 15, /* data=*/ 4, /* reset=*/ 16);

// ---------------------------------------------- WIFI ------------------------------------
const char* ssid = "Axel's Android";
const char* password = "ceox1565"; // such a security breach
const char* host = "ruche-connectee.polytech.unice.fr";
const int httpsPort = 443;
const int httpPort = 80;

// ---------------------------------------------- DATA RECEIVE ----------------------------
// using UART1 RX: 34 TX: 35 (has to be modified in HardwareSerial.cpp)
HardwareSerial serial(1);
const byte numChars = 32;
char receivedChars[numChars];
char dataMarker;
String temp = "_";
String humidity = "_";
String weight = "_";
String pressure = "_";
String light = "_";
boolean newData = false;

void setup() {
  Serial.begin(115200);
  // baudrate, UART mode, RX, TX
  serial.begin(115200);
  u8x8.begin();
  u8x8.setFont(u8x8_font_chroma48medium8_r);
  Serial.println();
  Serial.println("<Wifi LoRa ESP32 is ready>");
  u8x8.drawString(0,0,"Ready...");
  Serial.print("connecting to ");
  u8x8.drawString(0,1,"Connecting...");
  Serial.println(ssid);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  //Serial.println(getMacAddress());
  Serial.println(WiFi.macAddress());
  u8x8.drawString(0,2, "Connected");
  delay(1500);
  u8x8.clear();
}
```



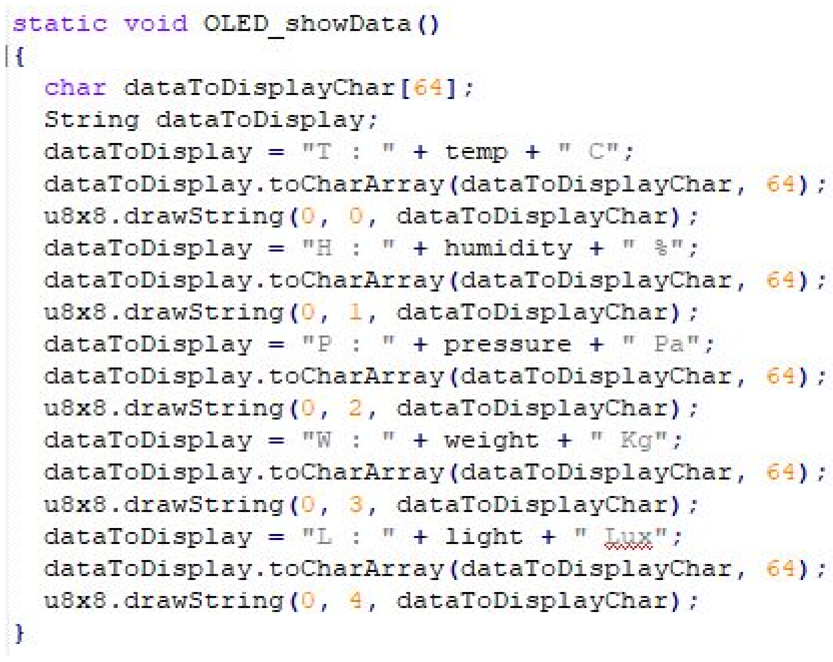
```
void loop() {
  recvWithStartEndMarkers();
  showNewData();
  OLED_showData();
  delay(1000);
}


void recvWithStartEndMarkers() {
  static boolean recvInProgress = false;
  static byte ndx = 0;
  char startMarker = '<';
  char endMarker = '>';
  char rc;
  while (serial.available() > 0 && newData == false) {
    rc = serial.read();
    if (recvInProgress == true) {
      if (rc != endMarker) {
        if (ndx == 0) {
          dataMarker = rc;
        }
        else {
          receivedChars[ndx - 1] = rc;
        }
        ndx++;
        if (ndx >= numChars) {
          ndx = numChars - 1;
        }
      }
      else {
        receivedChars[ndx - 1] = '\0'; // terminate the string
        recvInProgress = false;
        ndx = 0;
        newData = true;
      }
    }
    else if (rc == startMarker) {
      recvInProgress = true;
    }
  }
}
```



```
void showNewData() {
  if (newData == true) {
    if (dataMarker == 'T'){
      temp = receivedChars;
      sendData(temp, "_", "_", "_", "_");
    }
    else if (dataMarker == 'W'){
      weight = receivedChars;
      sendData("_", "_", "_", weight, "_");
    }
    else if (dataMarker == 'H'){
      humidity = receivedChars;
      sendData("_", humidity, "_", "_", "_");
    }
    else if (dataMarker == 'P'){
      pressure = receivedChars;
      sendData("_", "_", pressure, "_", "_");
    }
    else if (dataMarker == 'L'){
      light = receivedChars;
      sendData("_", "_", "_", "_", light);
    }
    newData = false;
  }
}
```

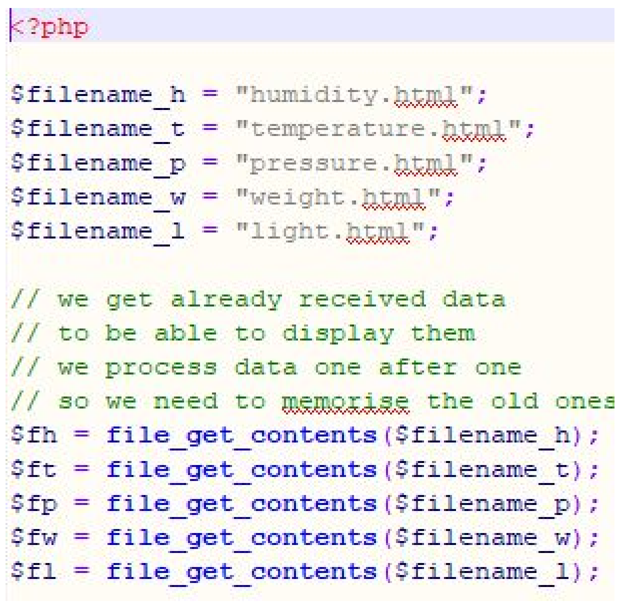


```
void sendData(String temp_, String humidity_, String pressure_, String weight_, String light_)
{
    //WiFiClientSecure client;
    WiFiClient client;
    String url = "";
    Serial.print("connecting to ");
    Serial.println(host);
    if (!client.connect(host, httpPort))
    {
      Serial.println("connection failed");
      return;
    }
    if (temp_ != "_"){
      url = "/ruche-connectee/public//dataCollector.php?Temperature=" + temp_;
    }
    else if (humidity_ != "_"){
      url = "/ruche-connectee/public//dataCollector.php?Humidity=" + humidity_;
    }
    else if (pressure_ != "_"){
      url = "/ruche-connectee/public//dataCollector.php?Pressure=" + pressure_;
    }
    else if (weight_ != "_"){
      url = "/ruche-connectee/public//dataCollector.php?Weight=" + weight_;
    }
    else if (light_ != "_"){
      url = "/ruche-connectee/public//dataCollector.php?Light=" + light_;
    }
    Serial.print("requesting URL: ");
    Serial.println(url);
    client.print(String("GET ") + url + " HTTP/1.1\r\n" +
                         "Host: " + host + "\r\n" +
                         "User-Agent: BuildFailureDetectorESP8266\r\n" +
                         "Connection: close\r\n\r\n");
    Serial.println("request sent");
    Serial.println("closing connection");
}
```



```
static void OLED_showData()
{
  char dataToDisplayChar[64];
  String dataToDisplay;
  dataToDisplay = "T : " + temp + " C";
  dataToDisplay.toCharArray(dataToDisplayChar, 64);
  u8x8.drawString(0, 0, dataToDisplayChar);
  dataToDisplay = "H : " + humidity + " %";
  dataToDisplay.toCharArray(dataToDisplayChar, 64);
  u8x8.drawString(0, 1, dataToDisplayChar);
  dataToDisplay = "P : " + pressure + " Pa";
  dataToDisplay.toCharArray(dataToDisplayChar, 64);
  u8x8.drawString(0, 2, dataToDisplayChar);
  dataToDisplay = "W : " + weight + " Kg";
  dataToDisplay.toCharArray(dataToDisplayChar, 64);
  u8x8.drawString(0, 3, dataToDisplayChar);
  dataToDisplay = "L : " + light + " Lux";
  dataToDisplay.toCharArray(dataToDisplayChar, 64);
  u8x8.drawString(0, 4, dataToDisplayChar);
}
```

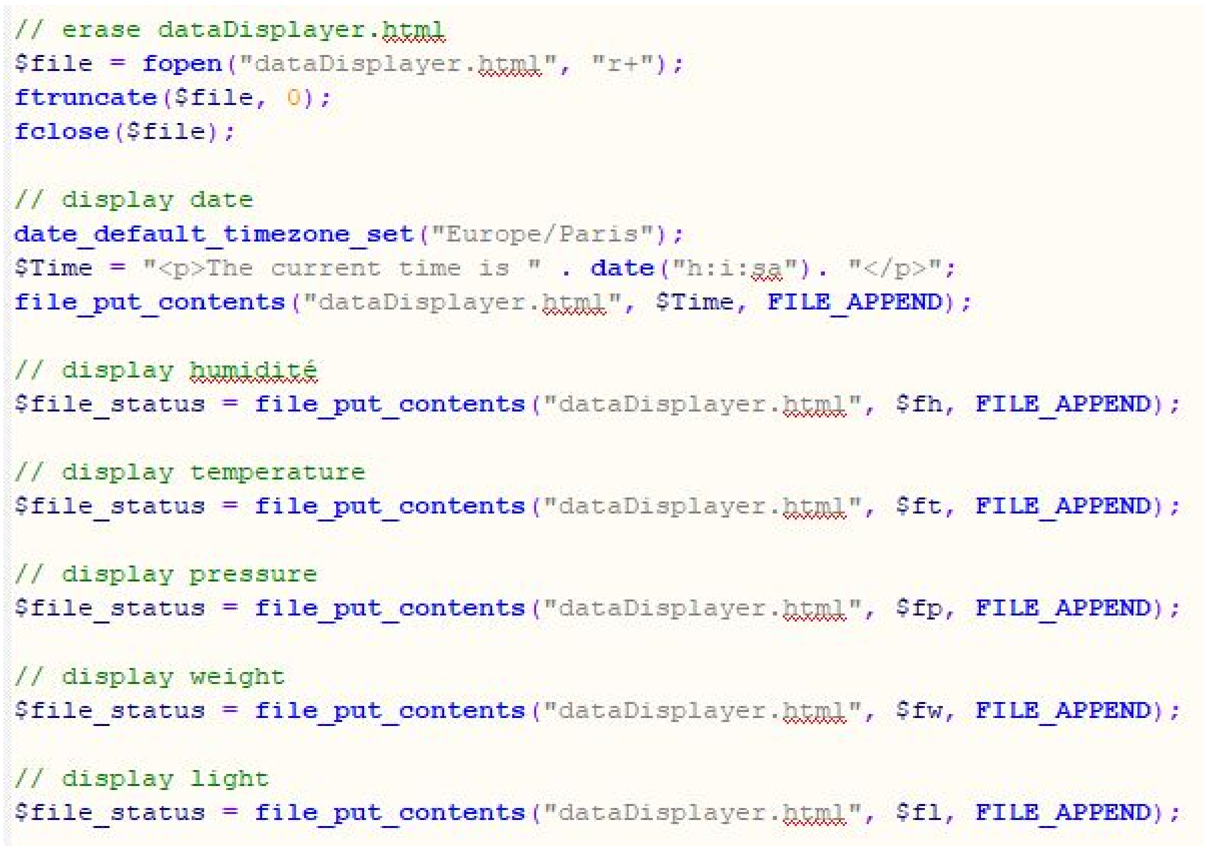
IV - PHP script



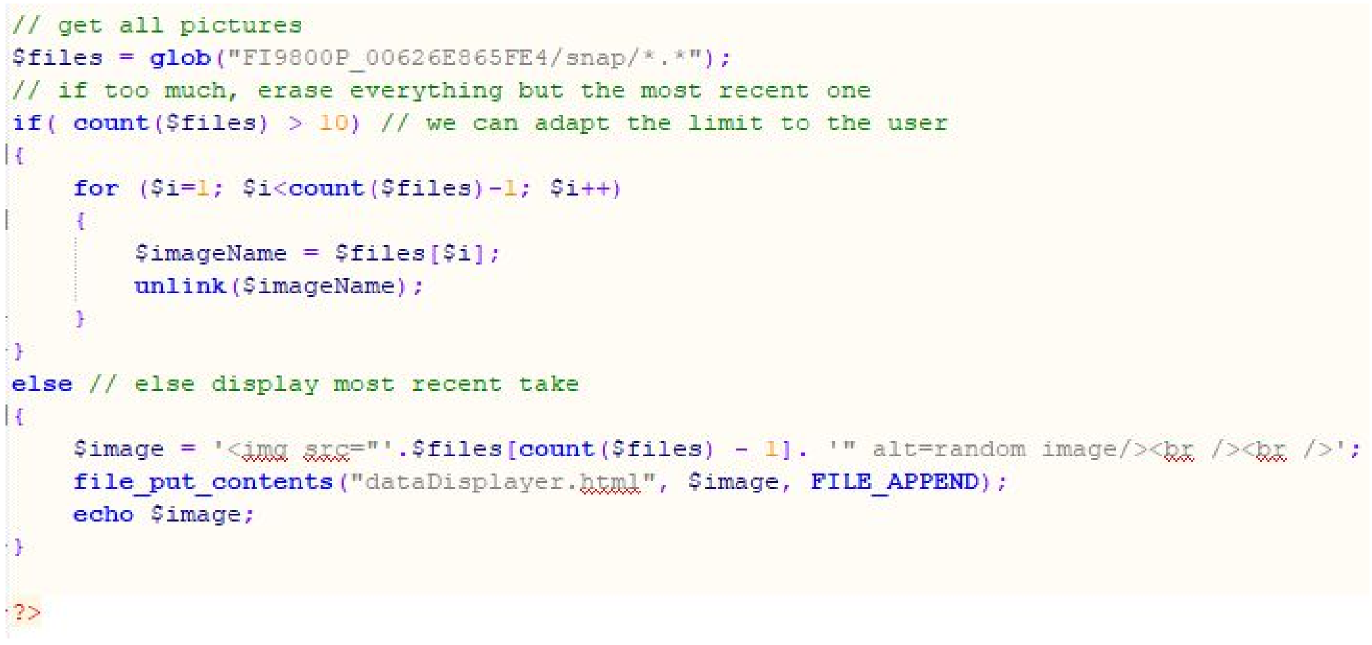
```
<?php

$filename_h = "humidity.html";
$filename_t = "temperature.html";
$filename_p = "pressure.html";
$filename_w = "weight.html";
$filename_l = "light.html";

// we get already received data
// to be able to display them
// we process data one after one
// so we need to memorise the old ones
$fh = file_get_contents($filename_h);
$ft = file_get_contents($filename_t);
$fp = file_get_contents($filename_p);
$fw = file_get_contents($filename_w);
$fl = file_get_contents($filename_l);
```

```php
if( $_GET["Humidity"] ) // humidity received, update of $fh
{
    $file = fopen($filename_h, "r+");
    ftruncate($file, 0);
    fclose($file);
    $humidity = $_GET['Humidity'];
    $StringToAppend = "<p> The Humidity is : " . $humidity . "% </p><br/>";
    $file_status = file_put_contents($filename_h, $StringToAppend, FILE_APPEND);
    if ($file_status != false)
    {
        $fh = file_get_contents($filename_h);
    }
    else
    {
        echo "FAILED";
    }
}
else if ( $_GET["Temperature"] ) // update $ft
{
    $file = fopen("temperature.html", "r+");
    ftruncate($file, 0);
    fclose($file);
    $temperature = $_GET['Temperature'];
    $StringToAppend = "<p> The Temperature is : " . $temperature . " Celcius </p><br/>";
    $file_status = file_put_contents($filename_t, $StringToAppend, FILE_APPEND);
    if ($file_status != false)
    {
        $ft = file_get_contents($filename_t);
    }
    else
    {
        echo "FAILED";
    }
}
```

```php
else if ( $_GET["Pressure"] ) // update $fp
{
    $file = fopen("pressure.html", "r+");
    ftruncate($file, 0);
    fclose($file);
    $pressure = $_GET['Pressure'];
    $StringToAppend = "<p> The Atmospheric Pressure is : " . $pressure . " </p><br/>";
    $file_status = file_put_contents($filename_p, $StringToAppend, FILE_APPEND);
    if ($file_status != false)
    {
        $fp = file_get_contents($filename_p);
    }
    else
    {
        echo "FAILED";
    }
}
else if ( $_GET["Weight"] ) // update $fw
{
    $file = fopen("weight.html", "r+");
    ftruncate($file, 0);
    fclose($file);
    $weight = $_GET['Weight'];
    $StringToAppend = "<p> The Weight is : " . $weight . " kg </p><br/>";
    $file_status = file_put_contents($filename_w, $StringToAppend, FILE_APPEND);
    if ($file_status != false)
    {
        $fw = file_get_contents($filename_w);
    }
    else
    {
        echo "FAILED";
    }
}
else if ( $_GET["Light"] ) // update $fl
{
    $file = fopen("light.html", "r+");
    ftruncate($file, 0);
    fclose($file);
    $light = $_GET['Light'];
    $StringToAppend = "<p> The Brightness is : " . $light . " LUX </p><br/>";
    $file_status = file_put_contents($filename_l, $StringToAppend, FILE_APPEND);
    if ($file_status != false)
    {
        $fl = file_get_contents($filename_l);
    }
    else
    {
        echo "FAILED";
    }
}
```



```php
// erase dataDisplayer.html
$file = fopen("dataDisplayer.html", "r+");
ftruncate($file, 0);
fclose($file);

// display date
date_default_timezone_set("Europe/Paris");
$Time = "<p>The current time is " . date("h:i:sa"). "</p>";
file_put_contents("dataDisplayer.html", $Time, FILE_APPEND);

// display humidité
$file_status = file_put_contents("dataDisplayer.html", $fh, FILE_APPEND);

// display temperature
$file_status = file_put_contents("dataDisplayer.html", $ft, FILE_APPEND);

// display pressure
$file_status = file_put_contents("dataDisplayer.html", $fp, FILE_APPEND);

// display weight
$file_status = file_put_contents("dataDisplayer.html", $fw, FILE_APPEND);

// display light
$file_status = file_put_contents("dataDisplayer.html", $fl, FILE_APPEND);
```



```php
// get all pictures
$files = glob("FI9800P_00626E865FE4/snap/*.*");
// if too much, erase everything but the most recent one
if( count($files) > 10) // we can adapt the limit to the user
{
    for ($i=1; $i<count($files)-1; $i++)
    {
        $imageName = $files[$i];
        unlink($imageName);
    }
}
else // else display most recent take
{
    $image = '<img src="'.$files[count($files) - 1]. '" alt=random image/><br /><br />';
    file_put_contents("dataDisplayer.html", $image, FILE_APPEND);
    echo $image;
}

?>
```