Final project of Internet of things course

A real GSM IOT device

Piccardo Giovanni 4481533

Università di Genova
Laurea magistrale in Artificial Intelligence

June 2022

Project context

In the April 2021 I decided to buy an electric scooter. I used to park it outside and only when needed I took home the batteries to charging them. Recently I found a close place into a car box near home to park it, where I can charge it directly but some problems arise:

1. **How much I should pay to the box owner for charging of the scooter?**
2. **How to stop automatically the charging when the batteries are full?**

Figure 1: My electric scooter

As solution to those problems I decided to build a prototype of an internet of thing device, able to connect to the network, able to communicate some information as power consumes during charging.

Where and How The Device works

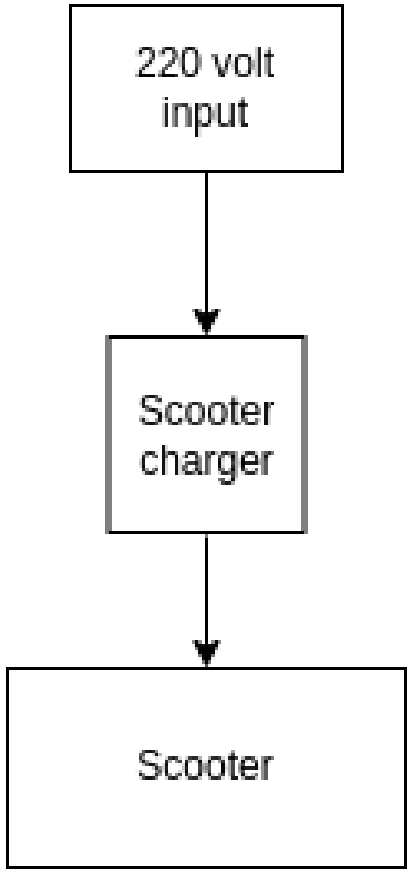


Figure 2: Original charging setup

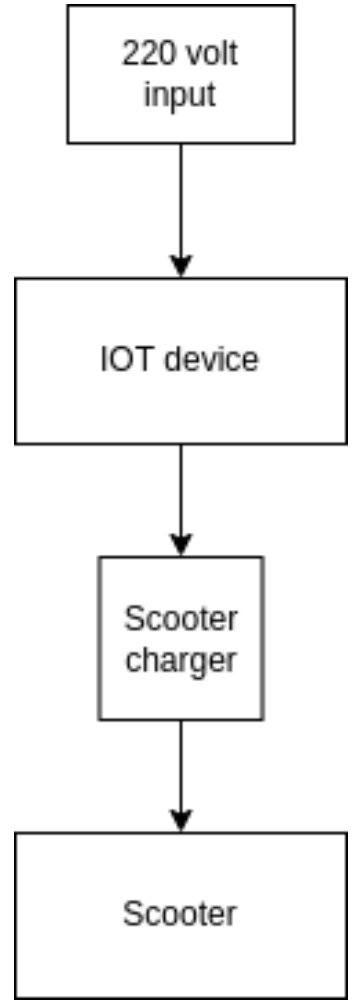


Figure 3: The new charging setup

Where and How The Device works

In the figures 2 we can see the original way to charge the scooter, instead in the figure 3 we see how the IOT device edits the original setup to be able to control the charging procedure and how it is able to collect data from the charging.

Project Architecture

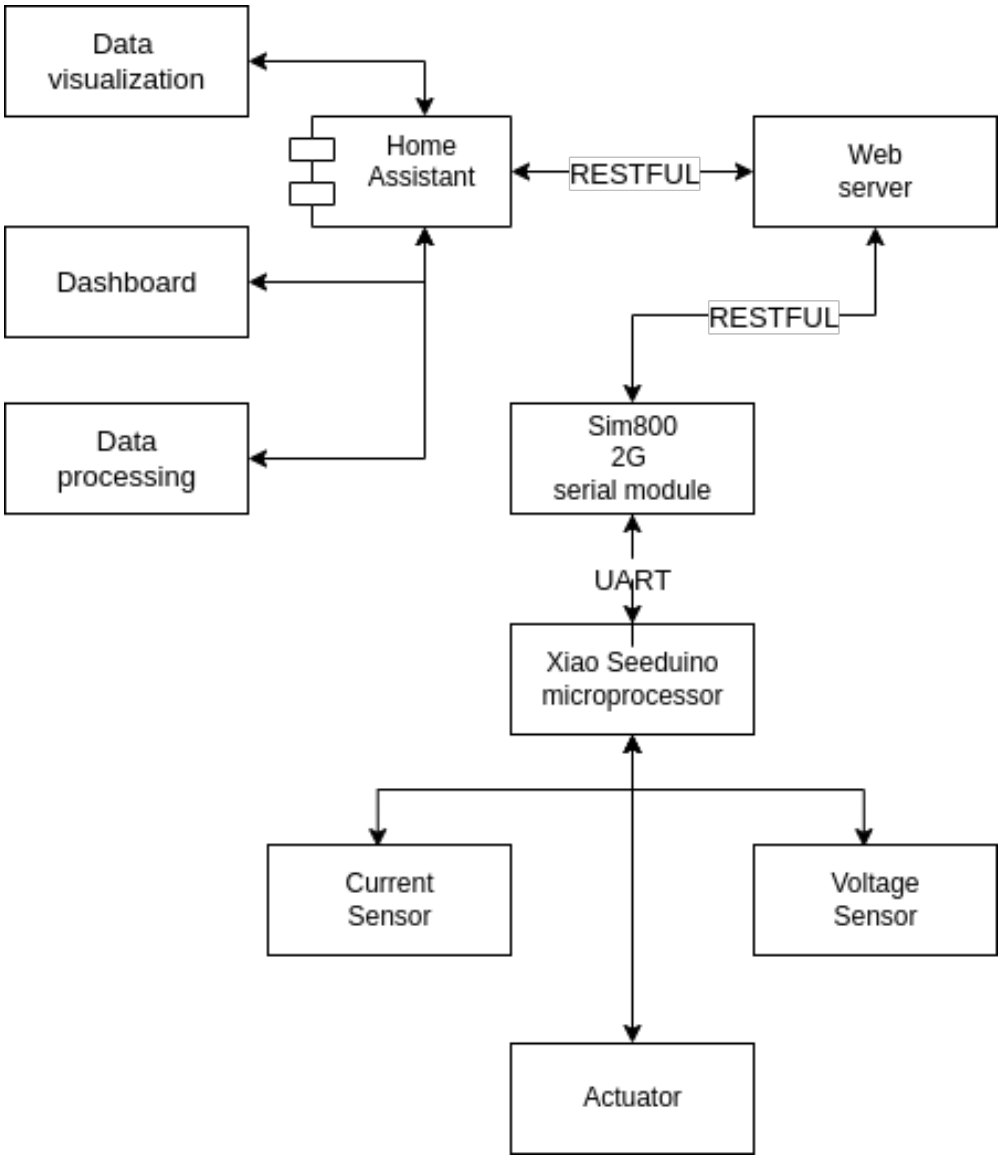


Figure 4: IOT device architecture

As can be seen in the figure 4 the prototype is composed by different parts, divided in:

- communication
- data collecting
- microprocessor

In the following slides I explain all the parts which compose the device

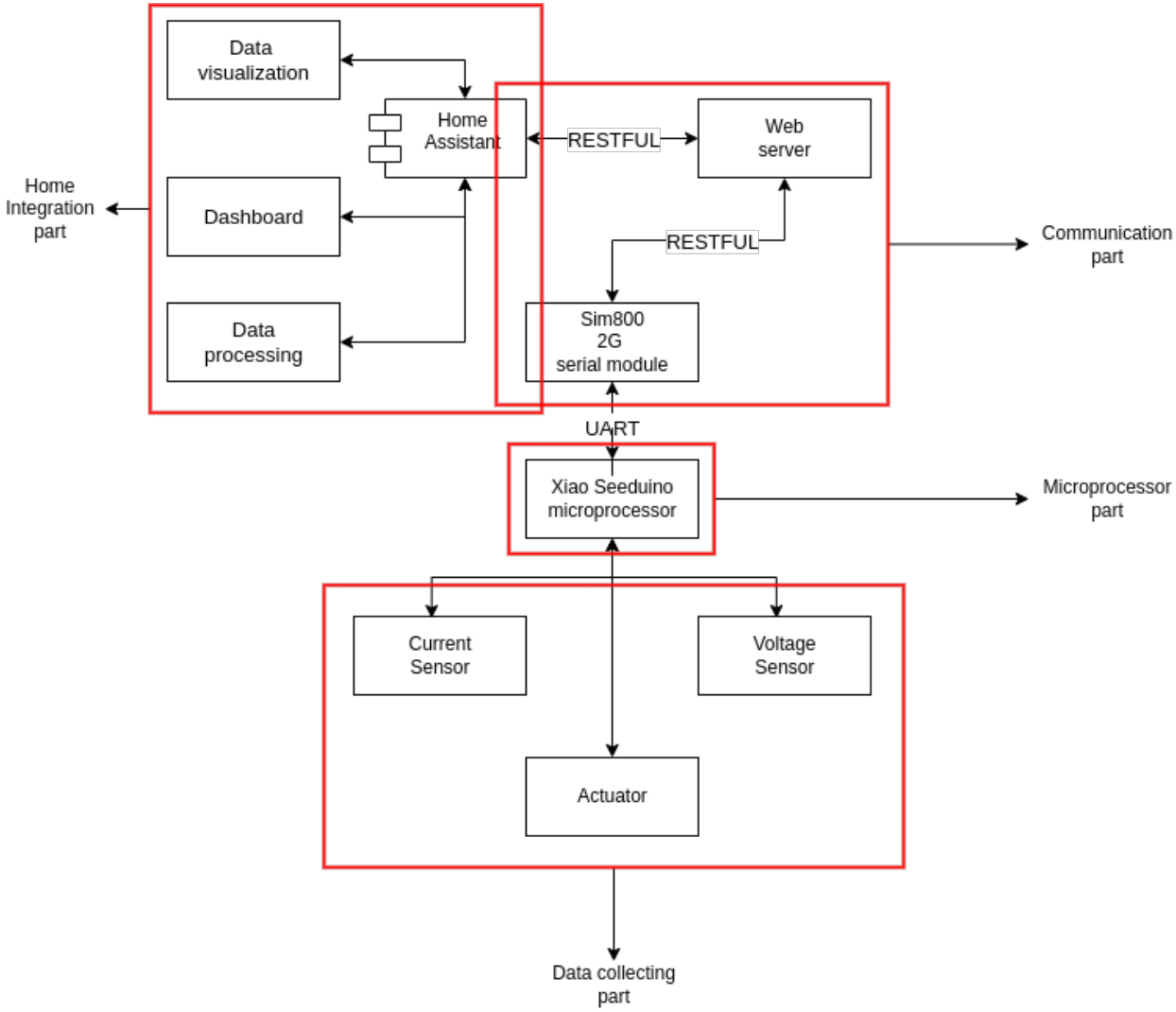


Figure 5: IOT device architecture explained

Project Architecture: IOT device

As shown in the figures 6 and 7 a logic circuit is very useful to design physical device which needs electronic components, showing how the components are connected. The device presented in those slides is divided in: an high voltage part (220V) and an low voltage part (4V). This separation helps to prevent any overvoltage problems and it's important for the security. The microprocessor and the GSM module are both in the low voltage part. The sensors are in both the low voltage part (power supply) and the high voltage part (measurement).

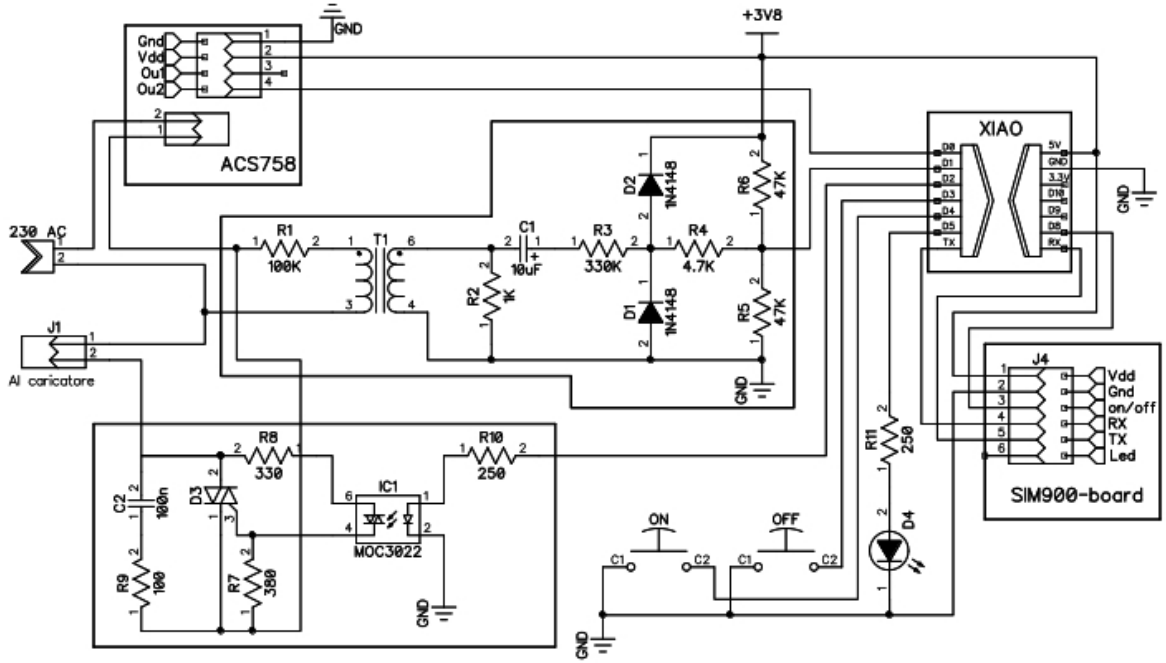


Figure 6: Logic circuit scheme of the IOT device

In this scheme the power supply is not represented. It which converts the 220 input volts into 4 volts needed for the low voltage components of the IOT device (Microprocessor, GSM module, led)

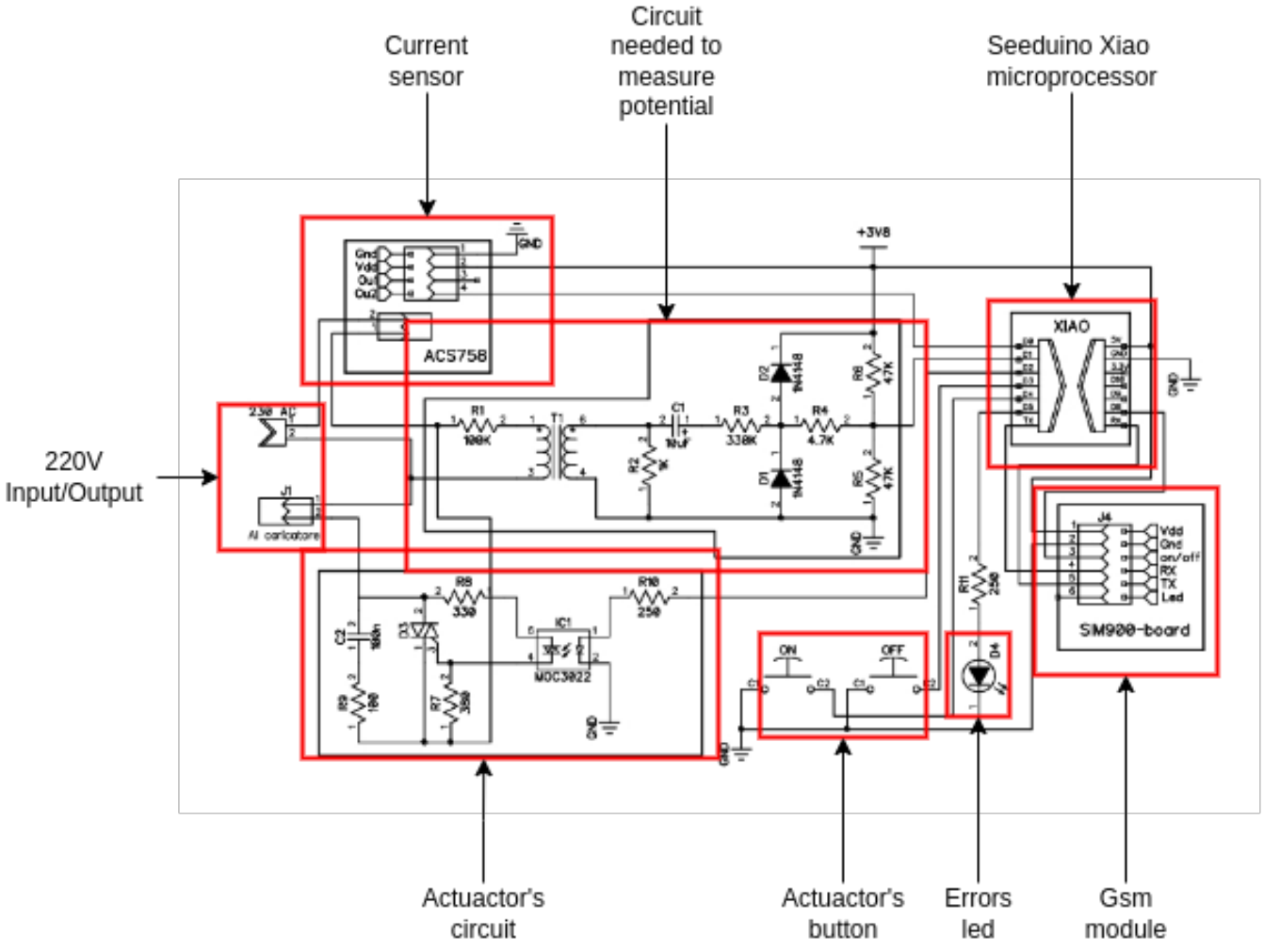


Figure 7: Logic circuit scheme of the IOT device

Project Architecture: The Microprocessor

The most important part is the microprocessor, which is coded to collect, to pre-process and to send the collected data. For the prototype I used the Seeeduino XIAO [22] a little 32 bit ARM board with good performance but it needs less power. It is designed in a tiny size and can be used for wearable devices and small projects. This microprocessor has 14 GPIO PINs, which can be used for 11 digital interfaces, 10 PWM interfaces, DAC output/ ADC input, I2C interface, SPI interface, UART interface, ecc.

Figure 8: Seeeduino XIAO microprocessor

Project Architecture: Data collecting: current sensor

The sensors used for this project permit to the device to collect data on the measured current (Ampers) and potential (Volts). Both the sensors need the Analog to Digital Converter (ADC), a built-in circuit in the microprocessor which is able to convert an analogic signal into a digital signal.
The current have to pass throw this sensor (figure 9) to be measured so the sensor given a know vcc input volt (typically 5V) it gives in output a value between the range $\{\frac{Vcc}{2}, Vcc\}$. When there is no current the output is $\frac{Vcc}{2}$, otherwise the output is $\frac{Vcc}{2} + 0.4Vcc * \mathrm{A}$.
Example: Input: 2A, output: $\frac{Vcc}{2} + 2 \times 0.4 = 3.3V$

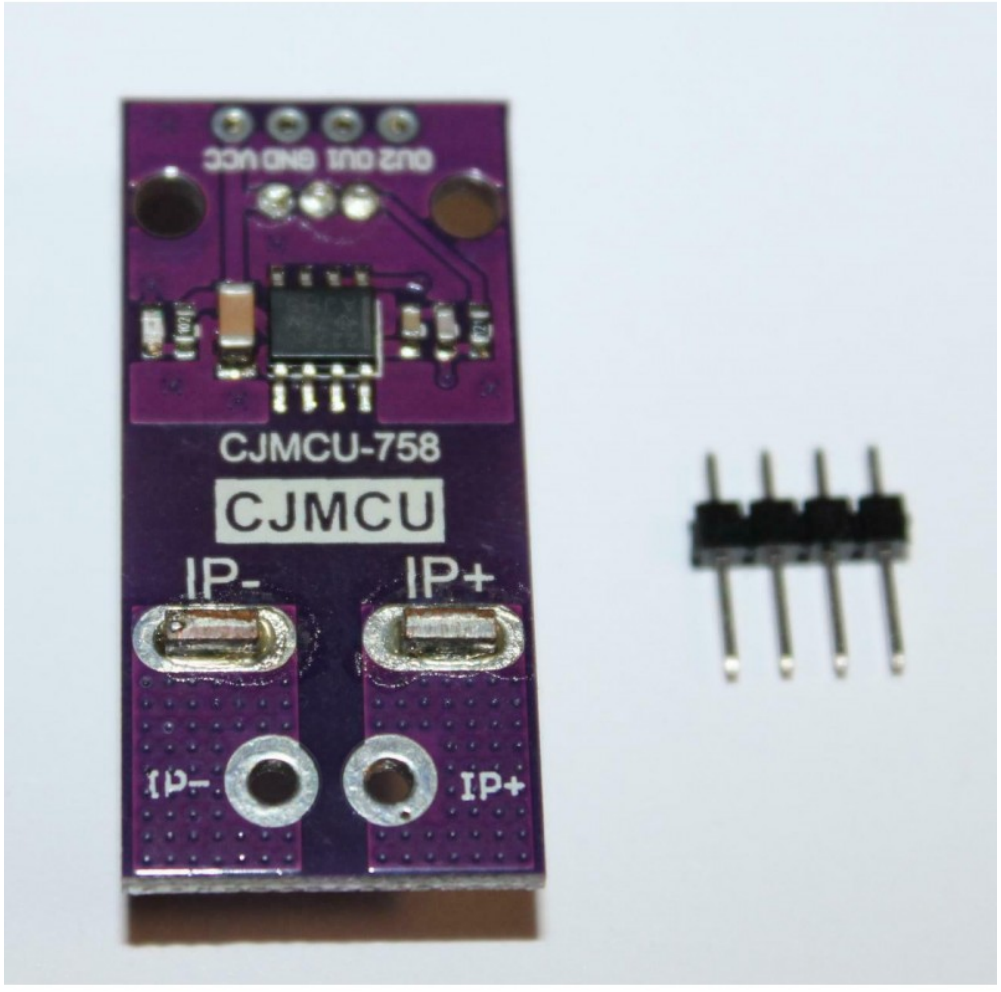


Figure 9: The current sensor [2]

Project Architecture: Data collecting: voltage sensor

Measuring the instantaneous voltage value of the electric line is needed to compute the Kwh used to estimate the cost of the scooter recharges (the energy supplier proposes the energy cost in $\frac{€}{\text{Kwh}}$). To build this sensor I used a circuit [26] (figure 10) which transforms the 220 volt into 12 volt, then the C1 capacitor removes the continuous part of the input signal, the resistors R3 and R4 produce a voltage between 3.3 volt and 0 volt to center the signal in the middle of the measuring range of the ADC. Finally the diods D1 and D2 and the resistor R1 limit the voltage and the current to the limits of the microprocessor's pin (few tens of microampere).

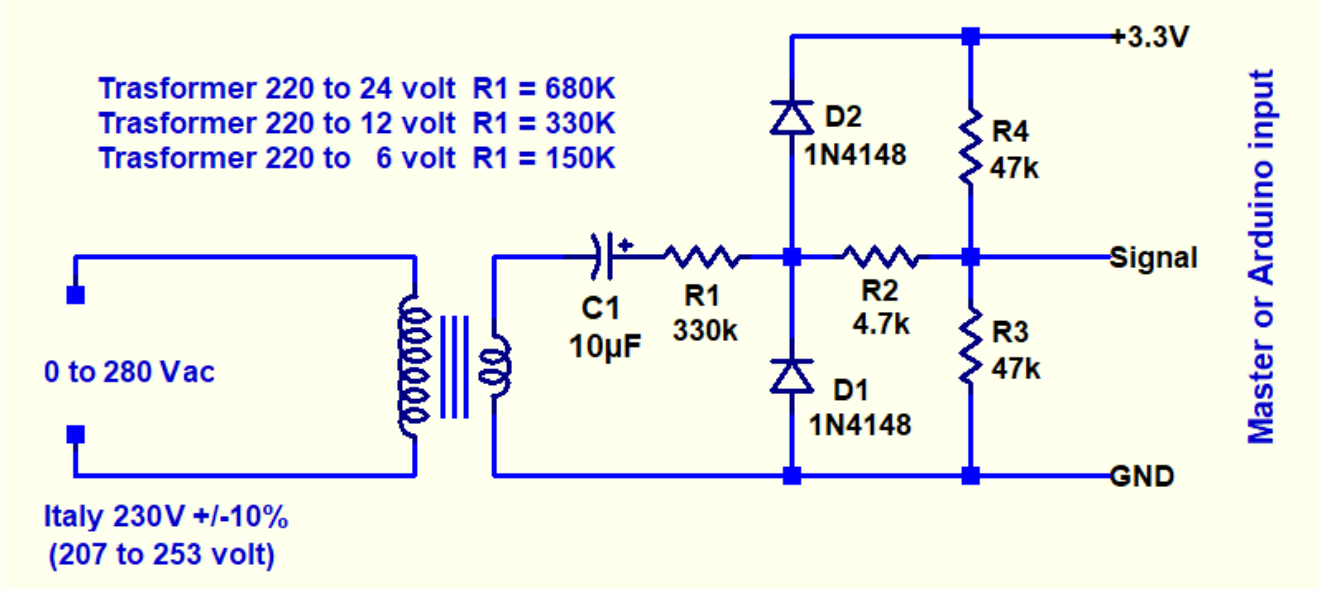


Figure 10: Logic circuit of the voltage sensor [26]

Project Architecture: Communication

The Sim800 2G GSM module [23] (figure 11) permits to communicate with the rest of the world via AT commands using the UART [25]. The garage box is far from my house so the Wi-Fi solution was impossible to adopt. To solve this problem I decided to implement a GSM module, a module ables to connect to the cellular network using a sim as a classical smartphone. To use this module an IOT subscription is needed. The one I bought included 500Mb, which is enough to send simple data. The HTTP [11] protocol isn't the only one which this module implements but it implements other protocols as the NTP [15] (network time protocol) used to obtain the current timestamp, very useful to label data with the correct time value.

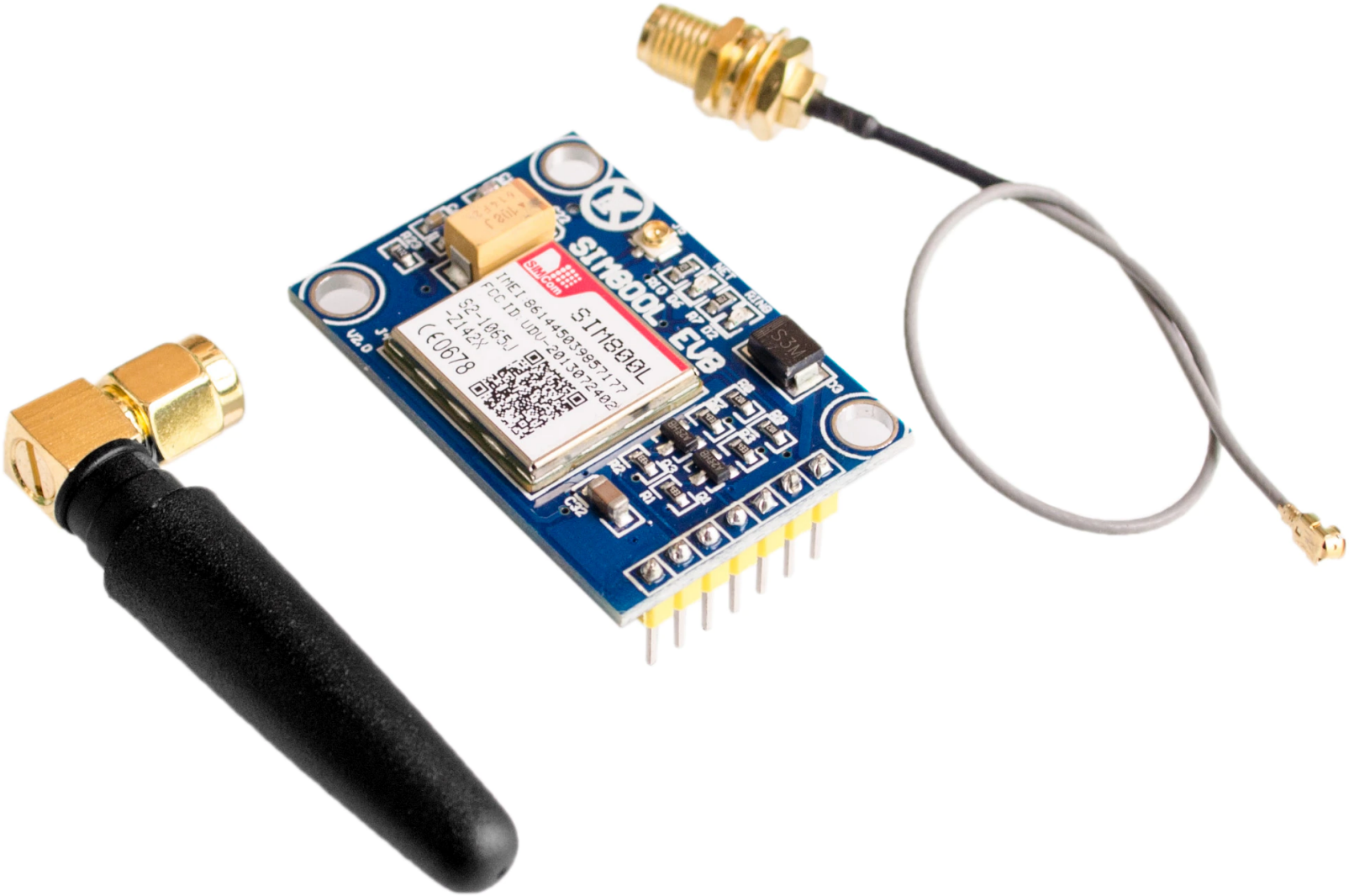


Figure 11: The Sim800 GSM 2G module [23]

Communication Architecture

The prototype implements a very basic REST [16] communication architecture. This device is not directly accessible via GSM[8] so I need a known server which permits to update data (via HTTP-POST) from the device. I decided to implement this middleware server using python [20] and Flask [6] as web server, which permits in a few lines to implement a REST server.

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"
```

Listing 1: Flask minimal working example

The Middleware Server

The middleware server responds only to few responses:

- A **POST** method to upload data from the IOT device
- A **GET** method to receive data from the database's server

Programing The Microprocessor

To program the microprocessor I used Circuitpython[5] This python version is very basic, so a lots python constructors are not implemented due to the few space available on the board. For example I needed the watchdog timer, which controls the hardware routine that in case of problems (infinite loop, deadlocks, ...) automatically restarts the board, very important for the embedded programing. In the Seeduino Xiao the watchdog isn't implemented so I used the Arduino [3]'s library (C based) to implement it into the Circuitpython's source code. Even if Circuitpython is very basic the possibility to use python in embedded programing is very useful and interesting because python is a high level and very simple language, as can be seen from the difference between the code presented in the listings 2 and 3.

Example: blinking led Arduino vs Circuitpython

```
void setup() {
    pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
    digitalWrite(LED_BUILTIN, HIGH);
    delay(500);
    digitalWrite(LED_BUILTIN, LOW);
    delay(500);
}
```

Listing 2: Led blinking in Arduino

```
import board
import digitalio as dIO
led = dIO.DigitalInOut(board.A1)
led.direction = dIO.Direction.OUTPUT

while True:
    led.value = not led.value
    time.sleep(0.5)
```

Listing 3: Led blinking in Circuitpython

Main Program diagram

The main program (the flowchart in the figure 12) starts after the microprocessor's boot. It starts with the initialization of the SIM800 GSM module, where the module is setted to the HTTP mode, then the module is connected to a known NPT server to get the current date and time using the network time protocol built-in, the timestamp received will be used to calculate the current Kilo watt per hour[12], using the difference between each data's timestamp. In case of any error the led on the device's case is programmed to blink. After the initialization the main program enters into an infinite loop where it gets the data from the sensors, it saves the new just obtained values only if the new data's values are differed (higher than 10% or lower than 10%) from the 10% than the previous data's values.

Only if the filesystem is full the data are sent to the middleware server. When the button is pushed the charger's actuator is activated, so the 220V can pass to the Scooter charger. During the charing process the charger actuator is switched off when the current sensor value is lower than 10 Watt.

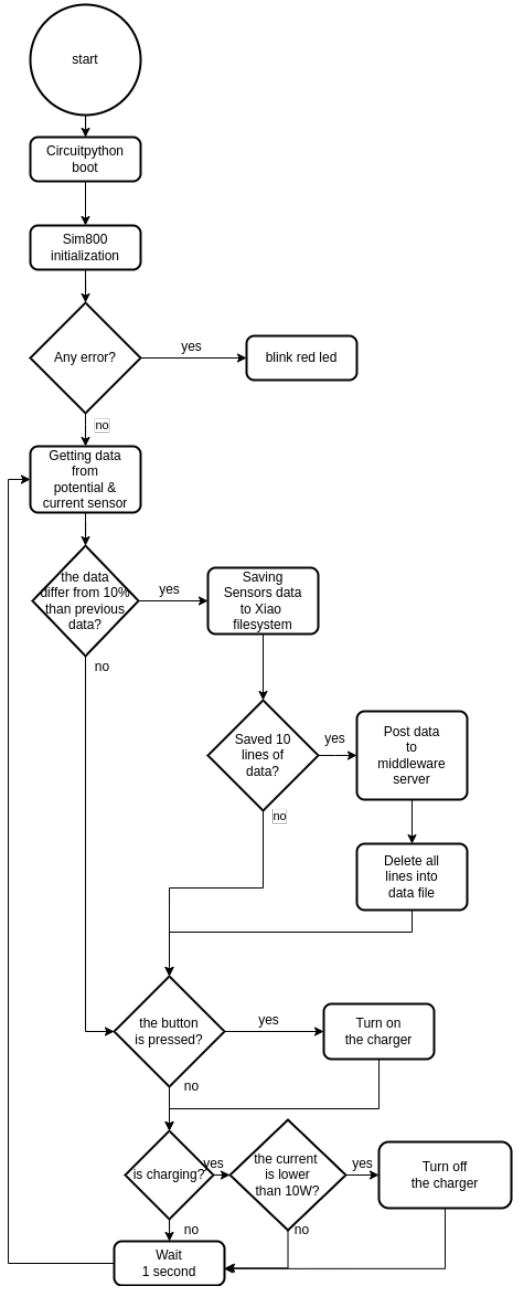


Figure 12: Main Program flowchart diagram

Security problems

As any type of device is connected to the network is very important to consider its security and the security into the communication protocol. The main problems to consider and resolves are:

- REST protocol security
- As 220V pass throw the prototype it must be closed in a box to avoid any injury

REST Security solutions

To resolve the REST security I decided to implement a client verification using an api key, which must be provided on each request to the server otherwise the request is denied. In this way only who knows the api key can communicate and can update and download data on/from the middleware server.

Hardware Security solutions

To avoid any possible injury problem due to the 220V potential exposure I decided to 3d print a box to enclose the prototype. I created a 3d model of the case calculating the best possible arrangement of each part which compose the prototype, also to avoid any heating problem.

3D printing process

To print the case of the prototype I executed a few steps:

1. Modeling in Blender [4] the case
2. Slicing the model in Prusa-slicer[19]
3. Sending the generated gcode [7] to the printer
4. Starting the printer process and wait the printing time, checking if any problems occur

In the following slides I show how I designed and how I 3d printed the IOT device's case

Step 1: 3d modelling the device's case

I used Blender, (see figure 13), a very complete open source 3D software, which in a few steps permits to generate anything.
I started with measuring the sizes of each component needed to build the device then I kept a basic cube from which I removed the top face to create in a second moment the box's closer, then I scaled the mesh [14] (a figure composed by different vertices, edges and faces in a 3d world) to the device's hardware parts sizes. Finally I added some holes to secure the hardware components to the case and one another hole for the external GSM module aerial.

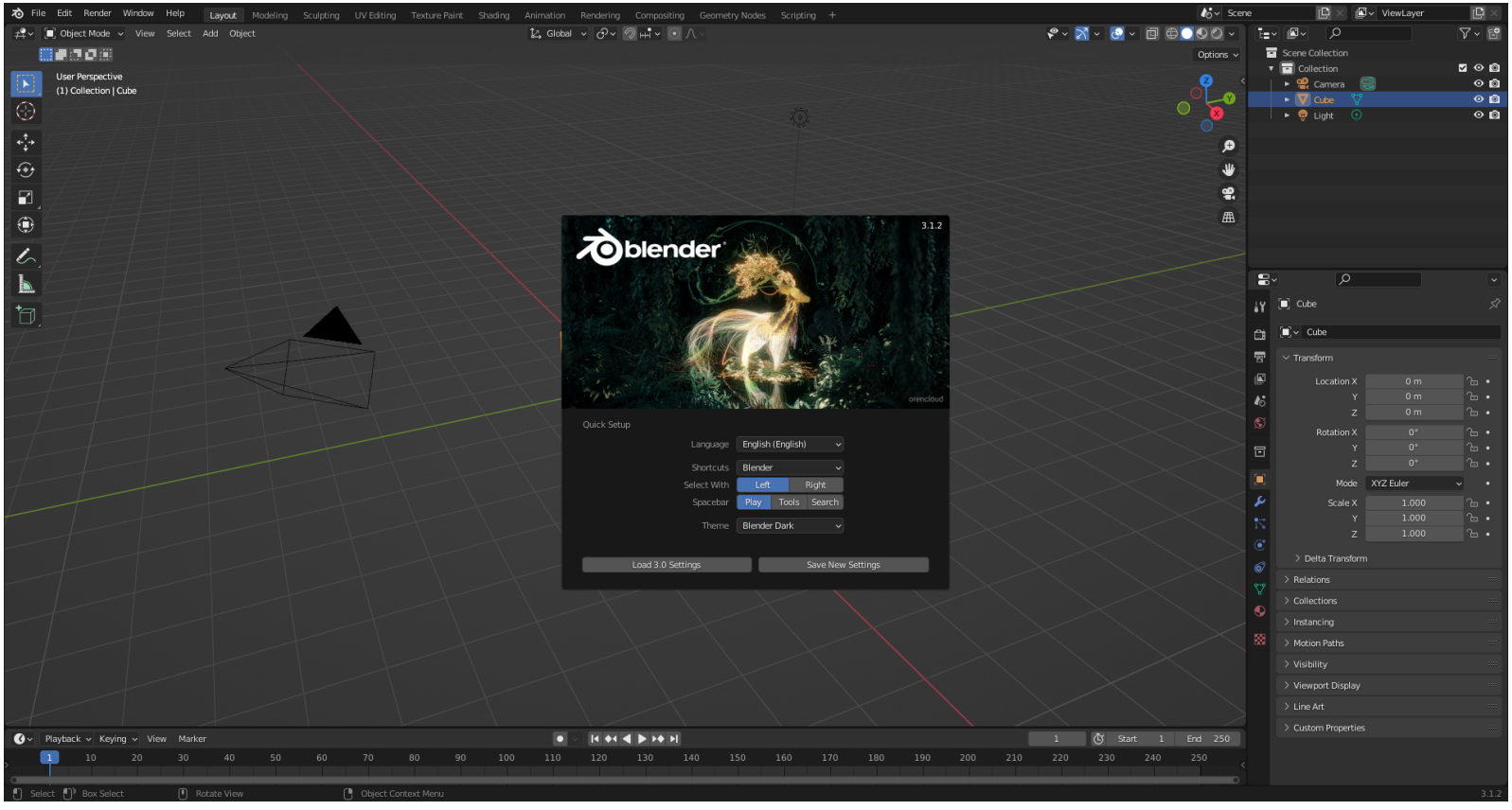


Figure 13: Blender screenshot

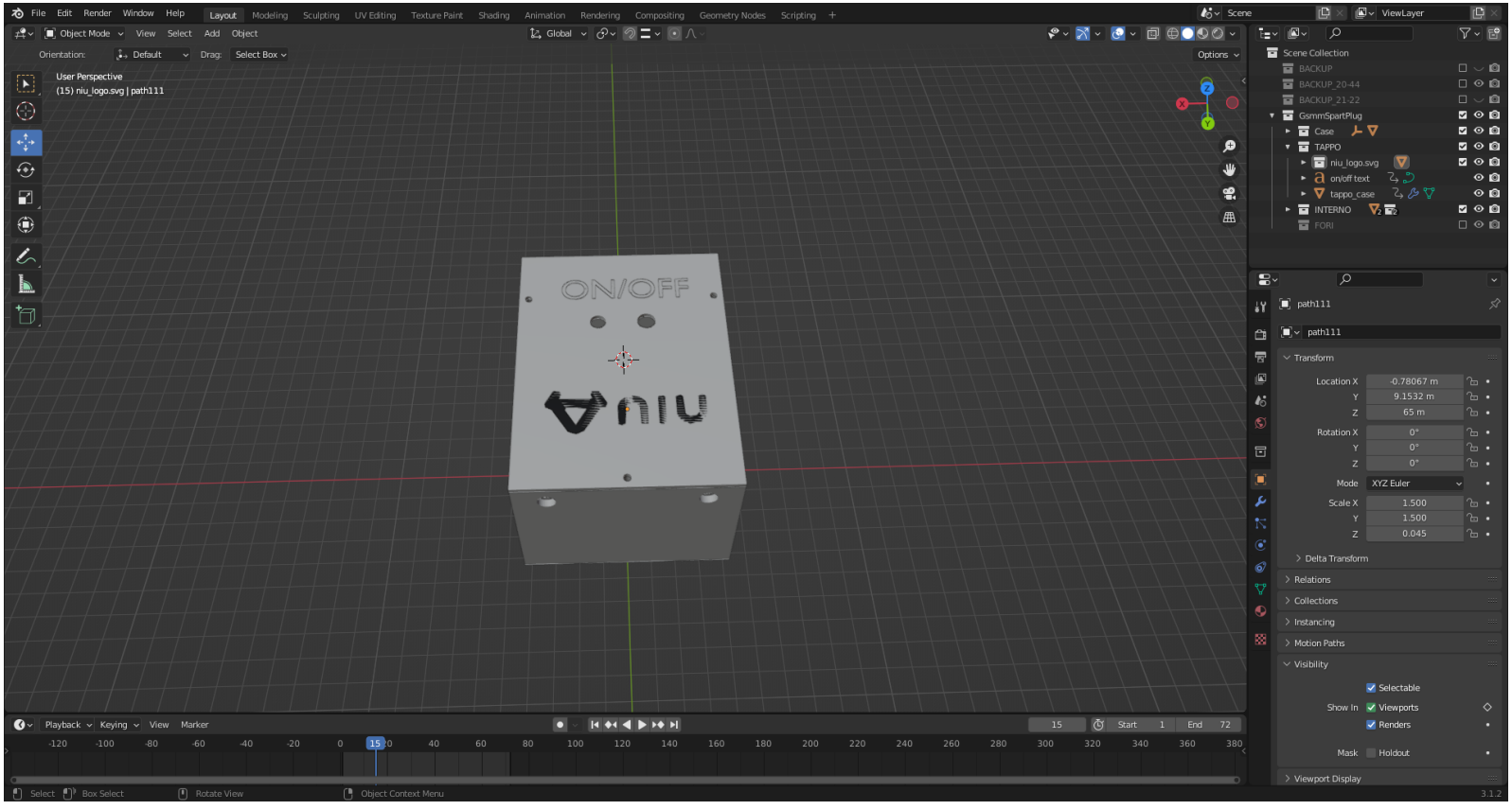


Figure 14: Blender view of the device's case

Step 2: slicing the generated model

The 3d printer has an internal parser which understands only the gcode language, a specific language for CNC and 3D printers. This code are converted into extruder movements and filament extrusions on the printing bed. To produce gcode I have to use a slicer software (figure 15), a tool which given an stl model (3d model composed only by its vertices, edges and faces) generates the gcode file. This is possible because as the name say the slicer slices [24] the object into a lots of layers (the layer high depends on the resolution selected for the model, usually I use 0.3 mm for object with big size, a compromise between a quick print time and a good resolution. Of course much lower is the resolution value, much more layers are generated so the printing time increase. In this case the printing time of the prototype's case took 6 hours more or less.

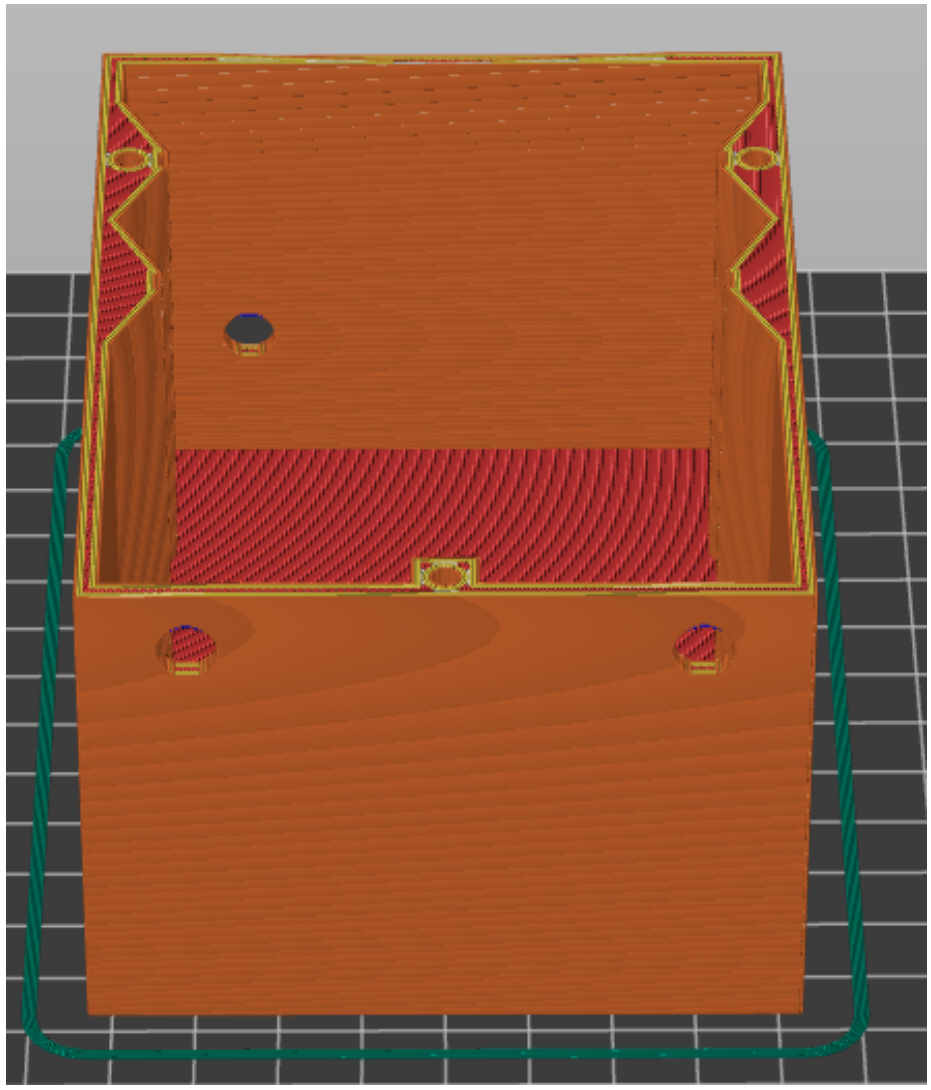


Figure 15: IOT device case sliced using Prusa-slicer

Step 3: Sending the gcode to the 3d printer

To 3d print any type of object is needed a 3d printer [1], a tool formed by three axis (one for each axis in the 3d space), an extruder heated to at least 170 degrees where in it passes a filament typically a plastic material (PLA is the most common), the axis steppers' and the motherboard. The steppers are motors but very particular because they are able to move only by a little angle (1.8 degrees) so the combined movements are very precise. The motherboard can be of different type, for example now the ARM processor are very used. In my 3d printer the motherboard is an Arduino Mega and on it is flashed Marlin [13], a very powerful C++ firmware for 3d printing. The 3d printer is controlled by the USB port but to avoid any problem with my laptop I use an Orange Pi [18] (a single board computer like Raspberry Pi [21]) and on it is installed Octoprint [17] (figure 16), a very interesting webserver for 3d printers. In this way I can send the gcode file via REST to the printer in a few seconds and I can control it using the webpage.

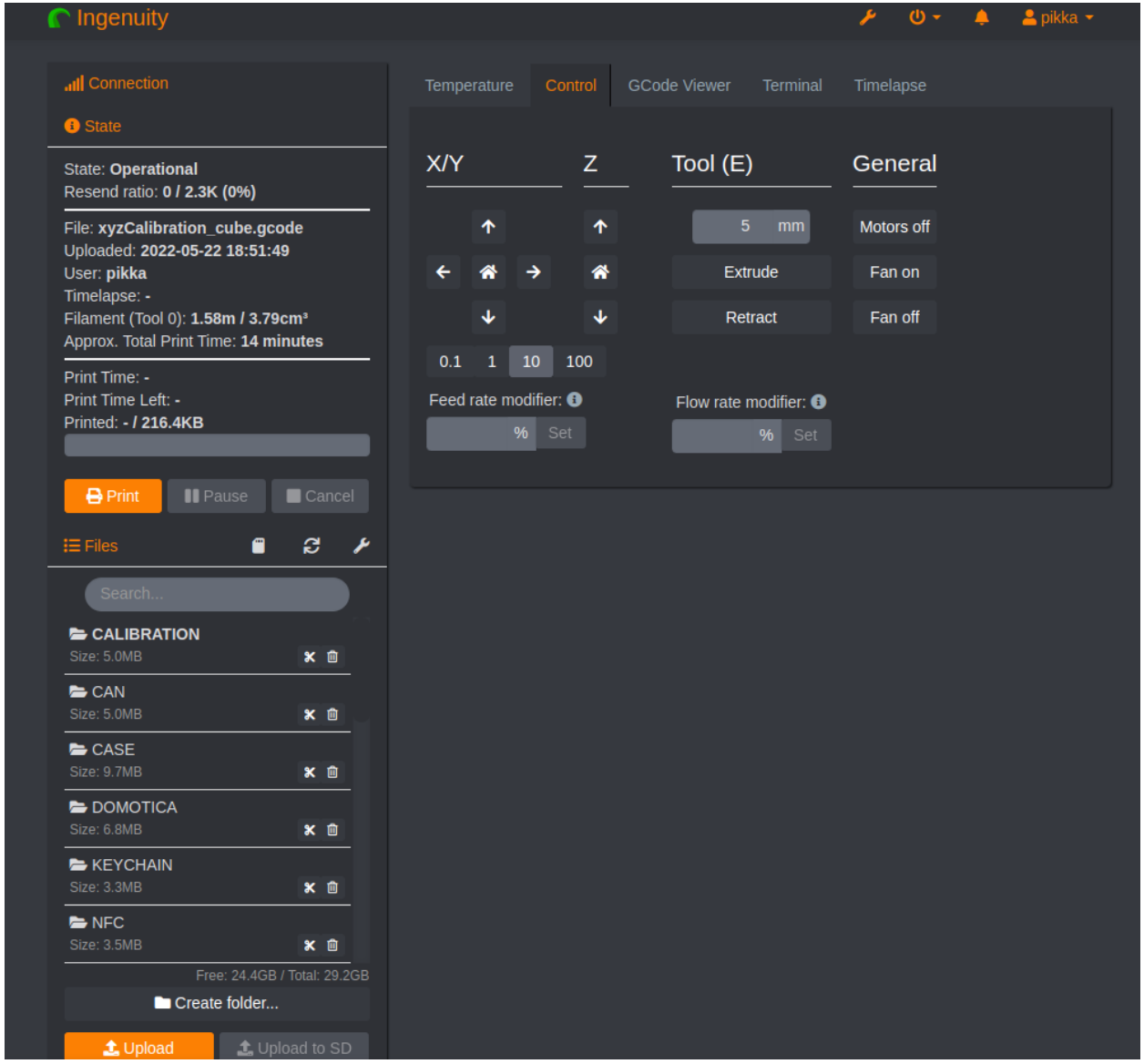


Figure 16: Screnshot of the Octoprint server webpage

Step 4: Starting the printer process and wait the printing time, checking if any problems occur

The last step of the 3d printing process is the most slow due to the printing time needed to print anything. Even if I use Octoprint, so the 3d printer is completely autonomous by my computer, I have always to check it because any type of problem can arise due to the 3d printer precision errors, filament jamming, ... (An example of a possible 3d printing error in the figure 17). Each time I print anything is very common the situation in which I have to restart the 3d printing a lots of times due to different type of problems occurred in the process.

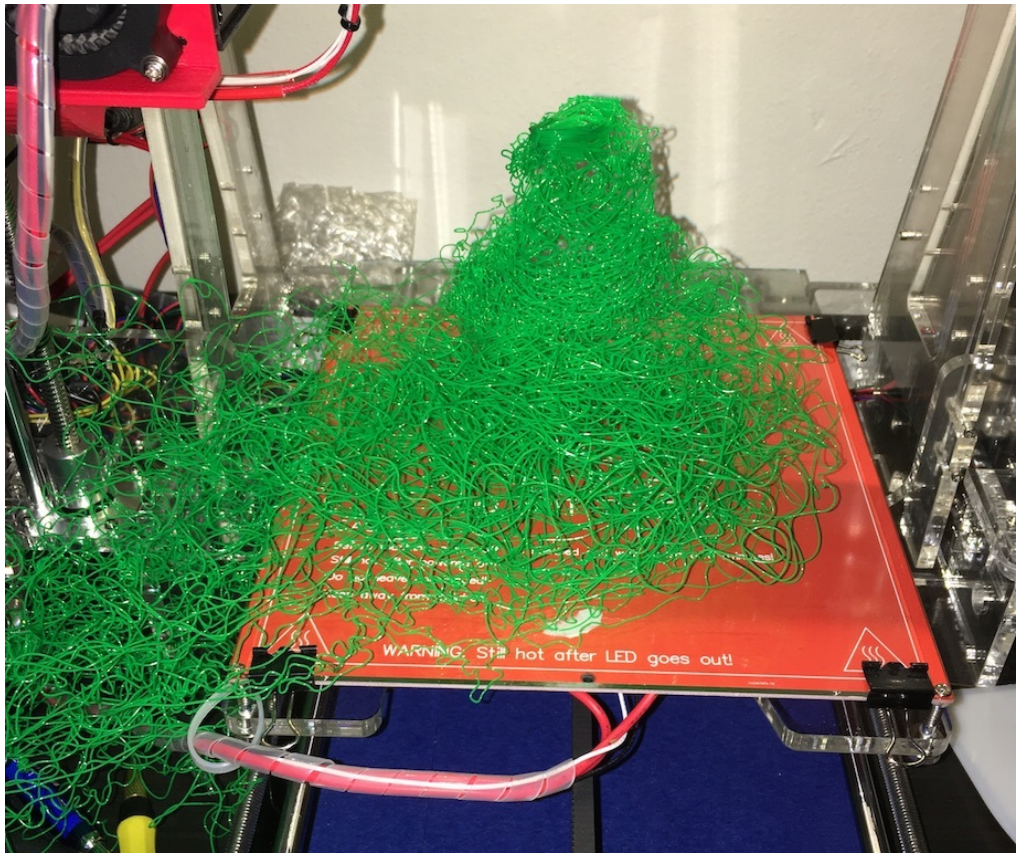


Figure 17: An example of a possible problem in 3d printing

Data Agration & Data Visualization

To aggregate data and visualize them I decided to integrate them into my home automation [10]. In particular I use Home assistant [9] a powerful tool (installable on al type of computer, very useful into a Raspberry Pi) to aggregate and visualize data from smart and IOT devices. In particular Home assistant is used for the house, it permits to integrate a lot of smart devices in a few steps, to generate automations and scripts which help to the use smart of our houses. To integrate and show the prototype data I used the REST protocol, generating a custom integration for Home assistant, generating a sensor for the current and the voltage, an computed sensor for the current power (measured Watt, calculated by Amperes $*$ Volts), the weekly summation of the Kwh of data and the weekly summation costs of based on an input text which define the cost of a Kwh.

Data Agration & Data Visualization II

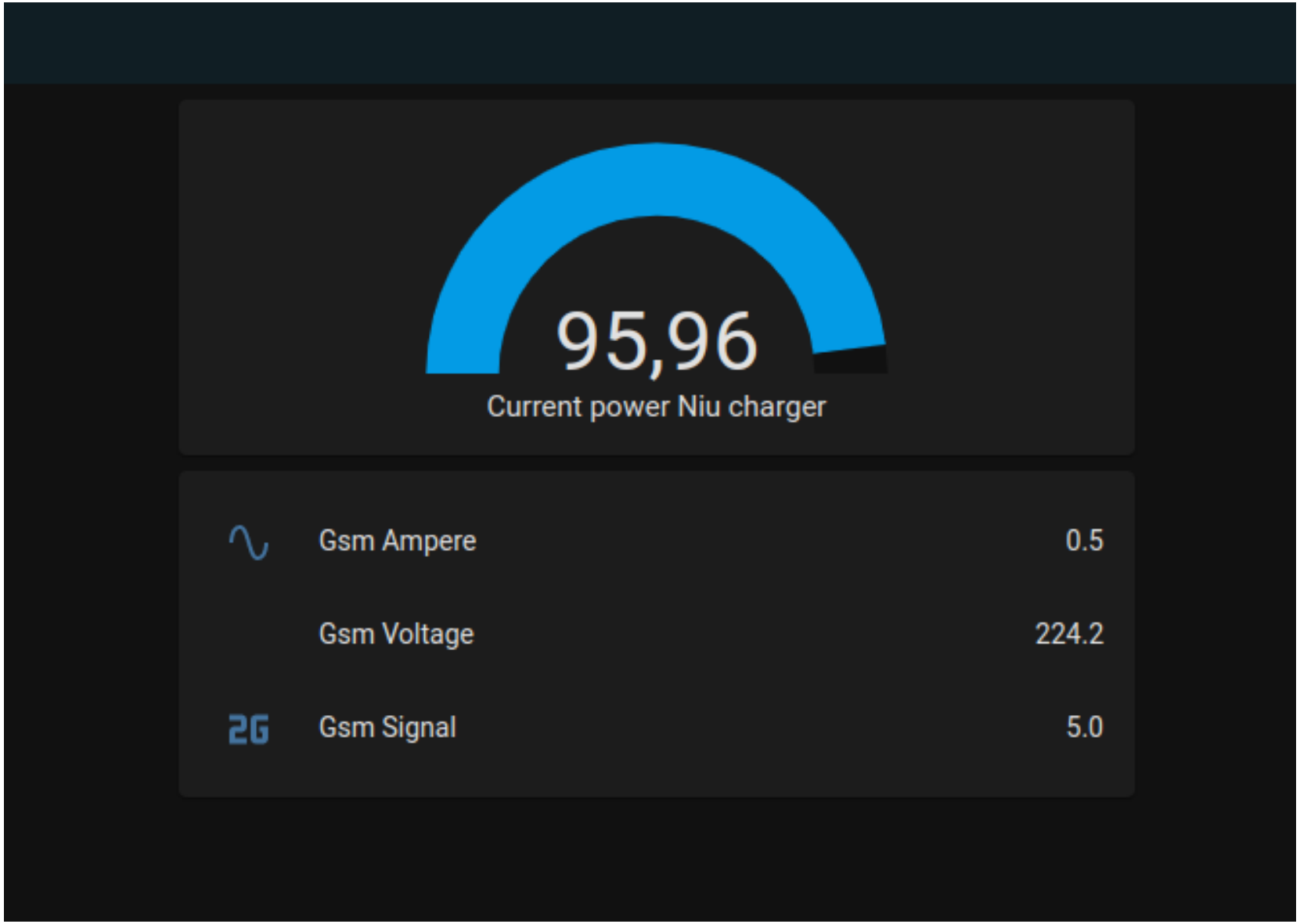
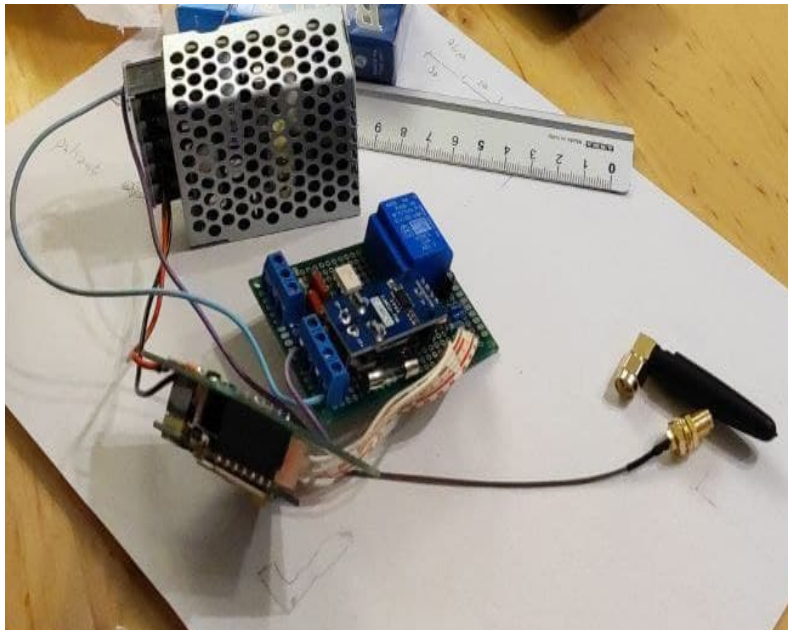
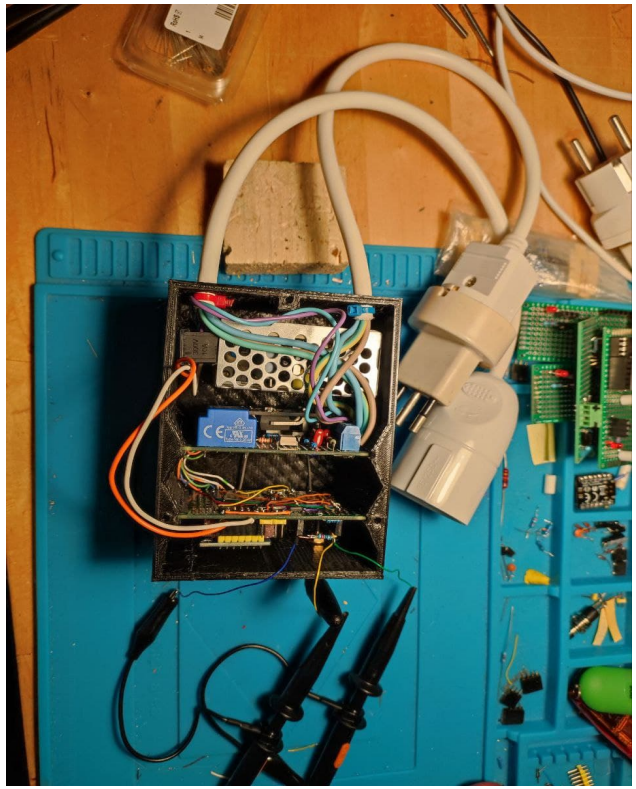
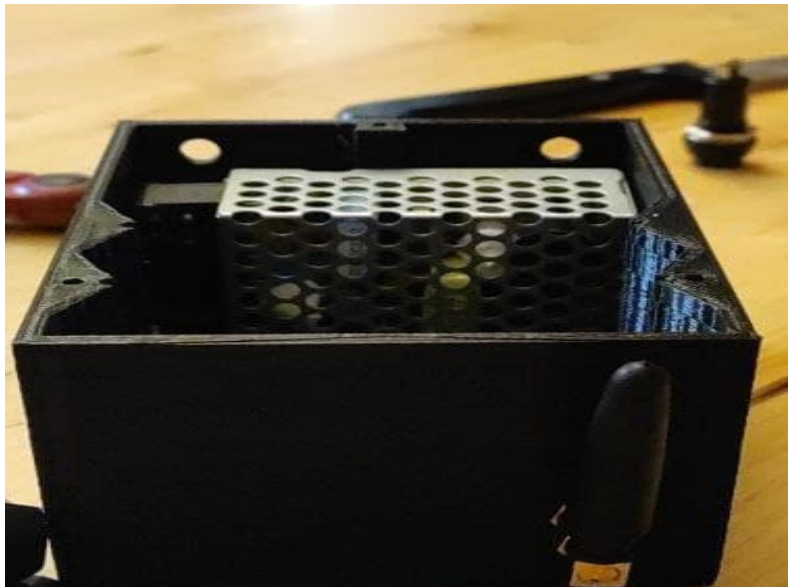


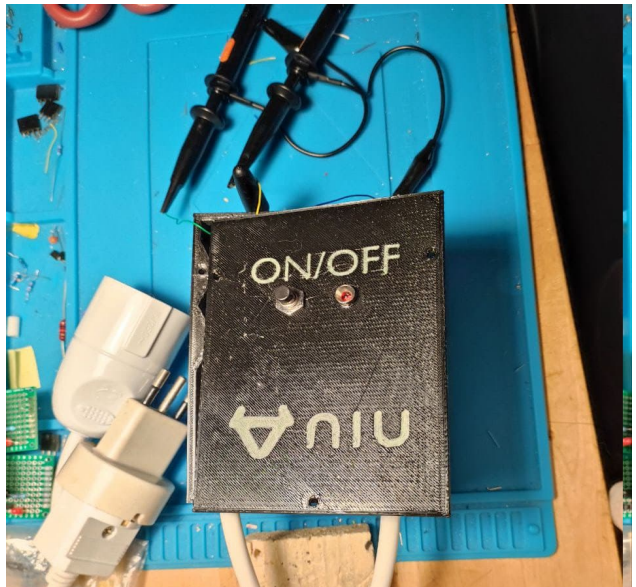
Figure 19: Data Visualization on testing data using Home Assistant

Some Photos Of The IOT Device









Conclusions

I designed this IOT device to:

1. Trace the scooter charging costs
2. Start instantaneously the scooter charging using an SMS, containing simple text as "start charging", sent by a phone to the number used by the SIM800 module's SIM.
3. Store the charging costs
4. Communicate automatically to the box owner the monthly costs of the scooter charging

The completed parts of the projects are: the device's hardware, the middleware server and the major part of the device's software. The device can collect and send the data but it don't implement yet the charging enabler routine via SMS. I implemented the integration of the data generated by the device using Home Assistant, in which is possible to store and aggregate all the data, in this software I'll implement the box owner notifier.

References I


3d printing on wikipedia.
https://en.wikipedia.org/wiki/3D_printing.


Acs758 current sensor datasheet.
https://dtsheet.com/doc/1564995/acs758-datasheet.


The arduino home page.
https://www.arduino.cc/.


The blender home page.
https://www.blender.org/.


Circuitpython documentation home page.
https://circuitpython.readthedocs.io/.


Flask documentation library home page.
https://flask.palletsprojects.com/.


Gcode on wikipedia.
https://en.wikipedia.org/wiki/G-code.

References II


Gsm communication protocol on wikipedia.
https://en.wikipedia.org/wiki/GSM.


Home assistant home page.
https://www.home-assistant.io/.


Home automation on wikipedia.
https://en.wikipedia.org/wiki/Home_automation.


Http protocol on wikipedia.
https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol.


Kilowatt-hour on wikipedia.
https://en.wikipedia.org/wiki/Kilowatt-hour.


The marlin firmware home page.
https://marlinfw.org/.

References III


Mesh definition on wikipedia.
https://en.wikipedia.org/wiki/Polygon_mesh.


Ntp protocol on wikipedia.
https:
//en.wikipedia.org/wiki/Network_Time_Protocol.


Ntp software architectural style on wikipedia.
https://en.wikipedia.org/wiki/Representational_
state_transfer.


The octoprint home page.
https://octoprint.org/.


The orange pi home page.
http://www.orangepi.org/.


The prusa-slicer home page.
https://www.prusa3d.com/page/prusaslicer_424/.

References IV


Python home page.
https://www.python.org/.


The raspberry pi home page.
https://www.raspberrypi.com/.


The seeeduino xiao home page.
https://wiki.seeedstudio.com/Seeeduino-XIAO/.


Sim800 datasheet and at commands manual.
https:
//www.alldatasheet.com/view.jsp?Searchword=SIM800.


Slicer (3d printing) on wikipedia.
https:
//en.wikipedia.org/wiki/Slicer_(3D_printing).

References V


Uart communication protocol on wikipedia.
https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter.


Voltage sensor scheme in italian.
https://www.theremino.com/wp-content/uploads/files/Theremino_PowerMeter_Help_ITA.pdf.