Spécification du projet d’IOT

eThermoGenerator

# Description technique de l’objet

## Concept

Un appartement possède plusieurs sources d’énergies réutilisables; l’une d’entre-elles est ma chaleur. Les plaques de cuisson, le four ou encore le chauffage sont généralement les éléments générant le plus de chaleur. A l’aide d’un procédé, il est possible de réutiliser cette chaleur pour générer de l’énergie électrique réutilisable.

C’est ce **cycle** que souhaite concrétiser eTG.



**NB** : L’objet réalisée ne reflète pas réellement notre imagination du concept, on a dû faire face à certaines contraintes matérielles. De ce fait on s'appuiera sur l’objet réalisé.

Certains éléments ont tout de même été incorporés à ce document pour se donner une idée de l’objet tel que nous l‘avions imaginé.

## Objet

eTG est un ensemble composé de 2 parties :

* un eTG Bank : appareil permettant de générer de l'électricité depuis une source de chaleur
* un eTG Mother : centrale permettant de contrôler les différents eTG Bank dispersés dans l’appartement.

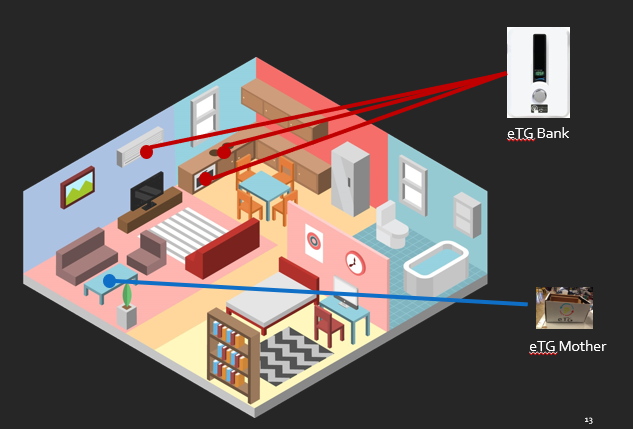
### Utilisation

L’utilisateur place les différents eTG Bank sur chaque source de chaleur dans l’appartement. Ensuite il allume la centrale (eTG Mother), spécifie à l’aide d’un bouton le nombre de batteries qu’il souhaite charger.

L’ordre est envoyé à chaque eTG Bank qui va lancer un procédé de transformation selon sa capacité à récupérer une tension issu de la différence de température. Ce procédé est effectué grâce au modules peltiers intégrés dans le dispositif eTG Bank.

Une fois les batteries chargées, l’eTG bank informe l’eTG Mother qui se chargera d’allumer une lampe et de signaler à l’utilisateur que l’opération est terminée.

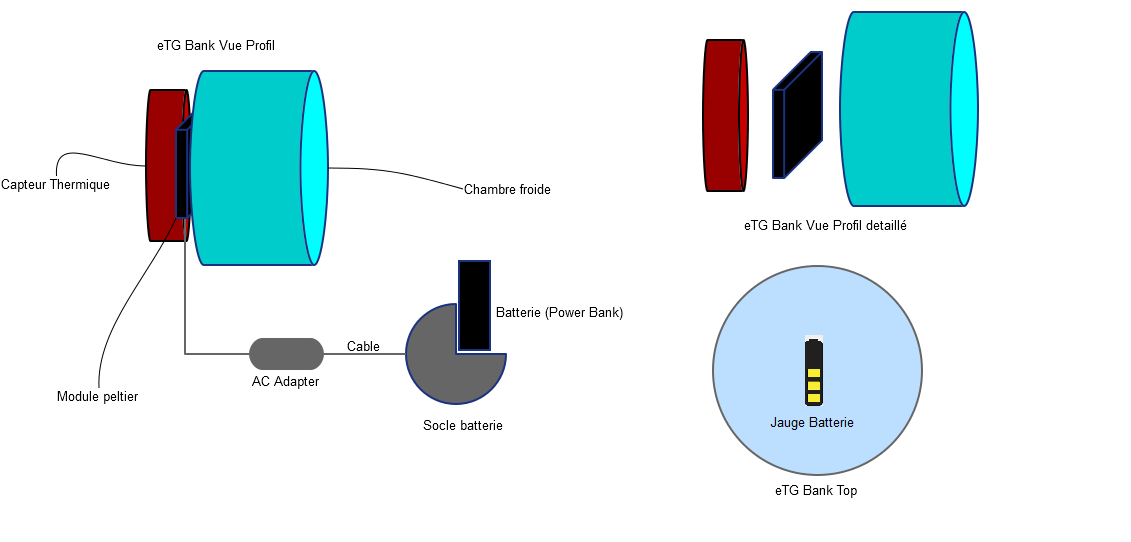
Il est également possible de laisser l’eTG Mother lancer les chargements de manière automatisée. Dans ce cas cela dépendra de la température disponible.



### Perspective matériele

Le kit eTG ressemblerait à ceci :

eTG Bank (implémentation possible)



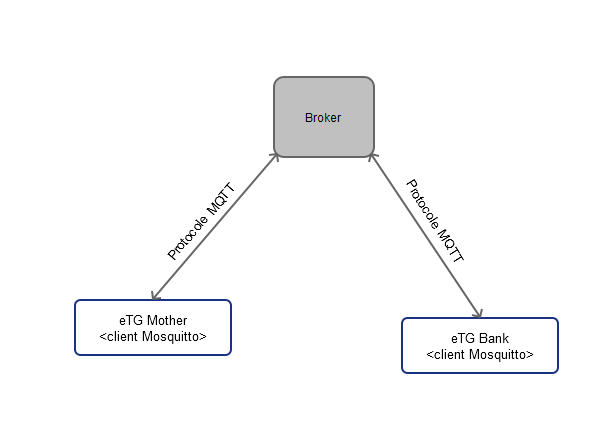
eTG Mother

|  |  |
| --- | --- |
|  |  |
|  |

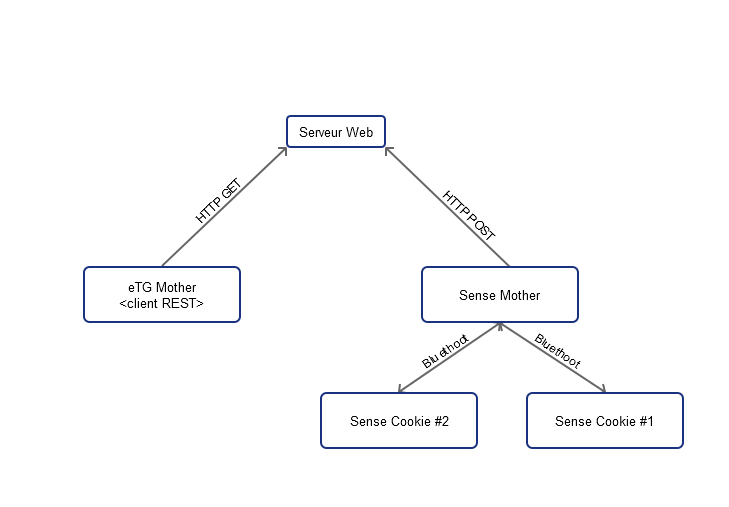
### Perspective logicielle

Il y’a deux types de perspectives:

* La perspective “réelle” : comme on l’avait imaginée en vrai



* La perspective “simulation” : utilisée pour la démonstration



# Architecture

## Architecture réelle

On part sur une architecture de type Publish&Subscribe

# 

Dans ce cas on aurait un eTG Mother qui correspondrait à un client Mosquitto (installé sur un Raspberry) qui s’abonnerait à chaque topic correspondant à chaque eTG Bank.

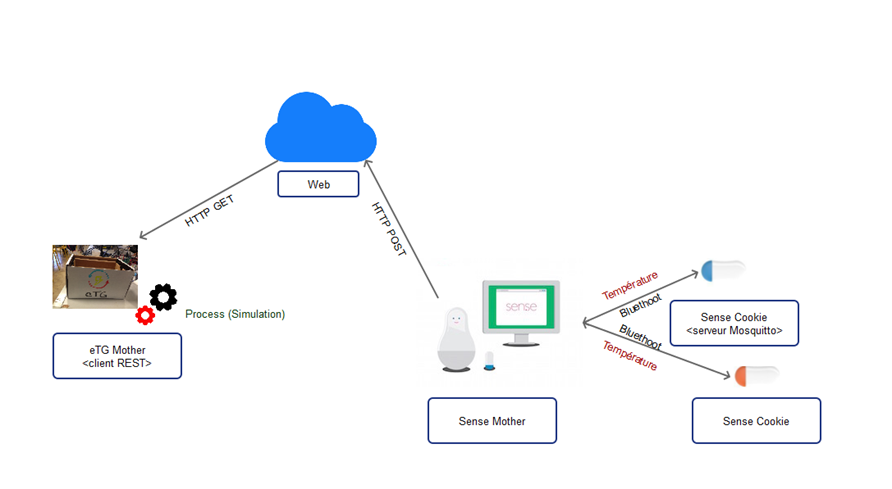
l’eTG Bank serait aussi un client Mosquitto abonné au topic correspondant à sa Mother.

Lors du lancement du programme, l’eTG Mother publie sur son topic les informations qu’elle souhaite faire exécuter l’eTG Bank.

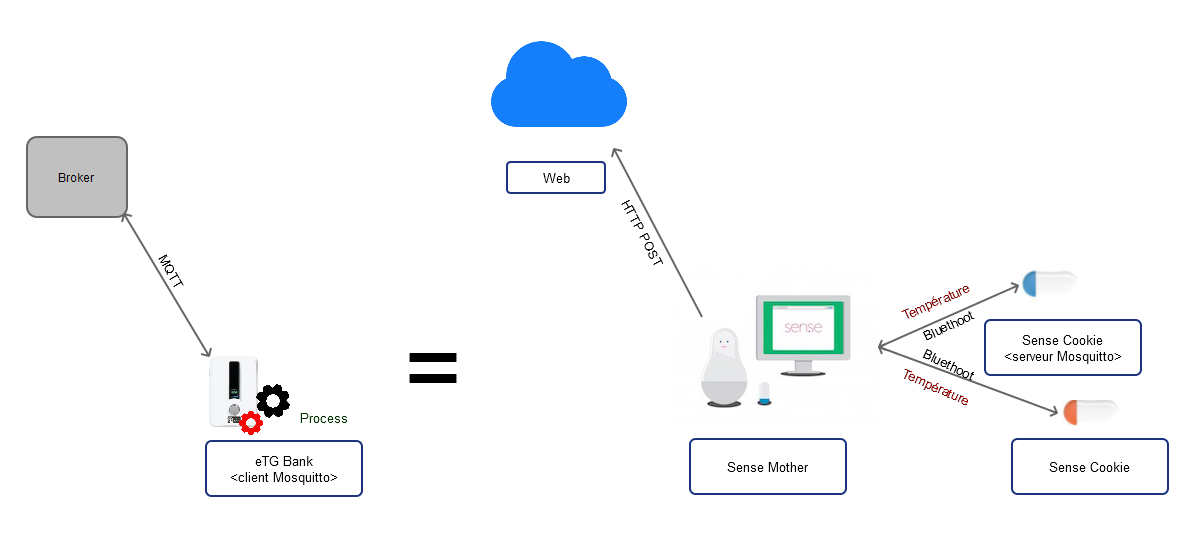
Une fois l’action réalisée ou pendant l’action l’eTG Bank publie sur son topic les informations que l’eTG Mother va récupérer.

## Architecture simulation

On part sur une architecture de type API REST



Dans ce cas là, on remplace tout la partie eTG Bank par le dispositif Mother



Les cookies vont récolter un certain nombre d’informations qu’on va récupérer à traver la Mother. La Mother va envoyer ces données sur un serveur Web et on aura plus qu’à récupérer via API REST les températures de chaque cookie.

Ainsi pour la simulation, à partir de la température on effectue un calcul côté eTG Mother afin de déterminer la quantité électrique potentielle générée selon une température donnée.

Les calculs effectués se basent sur les équations de l’effet Seebeck (Source: <https://fr.wikipedia.org/wiki/Effet_Seebeck>).

# Données manipulées

## Input

Les données envoyés par l’utilisateur sont issues des boutons sur l’interface de l’eTG Mother.

## Output

Les données récupérées sont les températures des différents cookies, elles sont récupérés grâce aux cookies de la Mother et sous format JSON à travers un serveur Web.

# 

# Code

Le langage utilisé est Python

Fichier battery.py pour les informations relatif aux batteries “””

|  |
| --- |
| class Battery(object):  def \_\_init\_\_(self, id = 0, label = "", max = 0, charging\_time = 0):  self.id = id  self.label = label  self.etat = 0  self.init = 0  self.max = max  self.charging\_time = charging\_time |

Fichier sense.py pour les méthodes du Mother (Sense)

|  |
| --- |
| import sense  from random import randrange  import time  import Battery  # Authentication  def authentication():  # sense.api\_key = '4a5f9881e53ba0a3cf237fa1c20737a0dbfe5e99'  # You can get your key using the API.  sense.api\_key = sense.User.api\_key(username='kellydiote', password='master2isc')  return sense  # Get the first cookie of your account  def getCookies():  user = sense.User.retrieve(expand=['devices'])  return [d for d in user.devices if d.resource.slug == 'cookie']  def subscribe(cookie):  # Create a subscription in order to receive new temperature events.  # You will be notified of a new event on a URL of your choice,  sub = sense.Subscription.create(  label='%s motion feed subscription' % cookie.label,  gatewayUrl='https://server2000.eu/events/',  subscribes= ['/nodes/%s/feeds/temperature/' % cookie.uid]  )  assert sub.uid is not None  def getCookieTemperature(cookie):  # Once you know UIDs you can instantiate objects without querying the API.  feed = sense.Node(cookie.uid).feeds(type='temperature')  # Get the last 5 events of the motion feed  ev = feed.events.list(limit=1)  #print(ev.objects[0].data.centidegreeCelsius)  return (ev.objects[0].data.centidegreeCelsius /100)  def getChargingTime(cookieTemperature):  cold = randrange(-10,10)  #la tension  dV=abs( (cookieTemperature-cold)\*2.1)  #l'intensite  I= (dV/100)  #temps de chargement = 1,2 \* capacite de la batterie (en mA) / courant du chargeur (en mA)  return (1.2 \* 10) /(I\*1000) |

Fichier etg\_mother.py pour les méthodes relatives à l’eTG Mother

|  |
| --- |
| def launch\_charge(init\_bn\_battery\_to\_charge=3, p\_batteries = [], charging\_time = 10):  batteries = p\_batteries  print(init\_bn\_battery\_to\_charge)  nb\_battery\_charged = 0  fini= False  while(nb\_battery\_charged < init\_bn\_battery\_to\_charge):  for bat in batteries:  for timer in [0, 100, bat.charging\_time]:  if (bat.etat < bat.max):  bat.etat += 1  percent = ((bat.etat \* 100) / bat.max)  print(" La Batterie " + bat.label + " est charge a " + str(percent) + " % ")  if (bat.etat >= bat.max):  nb\_battery\_charged += 1  print('Batterie ' + bat.label + ' est charge')  if nb\_battery\_charged == init\_bn\_battery\_to\_charge :  fini=True  if(fini):  break  time.sleep(1)  if(fini):  break  if (fini):  break  if(fini):  break |

Fichier config.py pour toutes les configs utilisés par l’eTG Mother

|  |
| --- |
| import RPi.GPIO as GPIO  import time  class Config(object):  # Define GPIO to LCD mapping  LCD\_RS = 26 #7  LCD\_E = 13 #8  LCD\_D4 = 6 #25  LCD\_D5 = 12 #24  LCD\_D6 = 5 #23  LCD\_D7 = 25 #18  LEDPinRed = 22  buttonPin = 4  buttonPinCount =18  # Define some device constants  LCD\_WIDTH = 16 # Maximum characters per line  LCD\_CHR = True  LCD\_CMD = False  LCD\_LINE\_1 = 0x80 # LCD RAM address for the 1st line  LCD\_LINE\_2 = 0xC0 # LCD RAM address for the 2nd line  # Timing constants  E\_PULSE = 0.0005  E\_DELAY = 0.0005  def \_\_init\_\_(self):  ###################  GPIO.setmode(GPIO.BCM)  # Setup the pin the LED is connected to  GPIO.setup(self.LEDPinRed, GPIO.OUT)  # Setup the button  GPIO.setup(self.buttonPin, GPIO.IN, pull\_up\_down = GPIO.PUD\_UP)  GPIO.setup(self.buttonPinCount, GPIO.IN, pull\_up\_down = GPIO.PUD\_UP)  GPIO.setwarnings(False)  GPIO.setmode(GPIO.BCM) # Use BCM GPIO numbers  GPIO.setup(self.LCD\_E, GPIO.OUT) # E  GPIO.setup(self.LCD\_RS, GPIO.OUT) # RS  GPIO.setup(self.LCD\_D4, GPIO.OUT) # DB4  GPIO.setup(self.LCD\_D5, GPIO.OUT) # DB5  GPIO.setup(self.LCD\_D6, GPIO.OUT) # DB6  GPIO.setup(self.LCD\_D7, GPIO.OUT) # DB7  # Initialise display  self.lcd\_init()  def lcd\_toggle\_enable(self):  # Toggle enable  time.sleep(self.E\_DELAY)  GPIO.output(self.LCD\_E, True)  time.sleep(self.E\_PULSE)  GPIO.output(self.LCD\_E, False)  time.sleep(self.E\_DELAY)    def lcd\_byte(self, bits, mode):  # Send byte to data pins  # bits = data  # mode = True for character  # False for command  GPIO.output(self.LCD\_RS, mode) # RS  # High bits  GPIO.output(self.LCD\_D4, False)  GPIO.output(self.LCD\_D5, False)  GPIO.output(self.LCD\_D6, False)  GPIO.output(self.LCD\_D7, False)  if bits&0x10==0x10:  GPIO.output(self.LCD\_D4, True)  if bits&0x20==0x20:  GPIO.output(self.LCD\_D5, True)  if bits&0x40==0x40:  GPIO.output(self.LCD\_D6, True)  if bits&0x80==0x80:  GPIO.output(self.LCD\_D7, True)  # Toggle 'Enable' pin  self.lcd\_toggle\_enable()  # Low bits  GPIO.output(self.LCD\_D4, False)  GPIO.output(self.LCD\_D5, False)  GPIO.output(self.LCD\_D6, False)  GPIO.output(self.LCD\_D7, False)  if bits&0x01==0x01:  GPIO.output(self.LCD\_D4, True)  if bits&0x02==0x02:  GPIO.output(self.LCD\_D5, True)  if bits&0x04==0x04:  GPIO.output(self.LCD\_D6, True)  if bits&0x08==0x08:  GPIO.output(self.LCD\_D7, True)  # Toggle 'Enable' pin  self.lcd\_toggle\_enable()  def lcd\_init(self):  # Initialise display  self.lcd\_byte(0x33,self.LCD\_CMD) # 110011 Initialise  self.lcd\_byte(0x32,self.LCD\_CMD) # 110010 Initialise  self.lcd\_byte(0x06,self.LCD\_CMD) # 000110 Cursor move direction  self.lcd\_byte(0x0C,self.LCD\_CMD) # 001100 Display On,Cursor Off, Blink Off  self.lcd\_byte(0x28,self.LCD\_CMD) # 101000 Data length, number of lines, font size  self.lcd\_byte(0x01,self.LCD\_CMD) # 000001 Clear display  time.sleep(self.E\_DELAY)  def lcd\_string(self, message, line):  # Send string to display  message = message.ljust(self.LCD\_WIDTH," ")  self.lcd\_byte(line, self.LCD\_CMD)  for i in range(self.LCD\_WIDTH):  self.lcd\_byte(ord(message[i]),self.LCD\_CHR)  #allumer les led  def light\_led(self):  return |

Fichier run.py pour le lancement et le déroulement du programme

|  |
| --- |
| import time  import sense  import datetime  import RPi.GPIO as GPIO  from SensTestV2 import \*  from Battery import Battery  from testFinale import Config  init\_bn\_battery\_to\_charge = 0  #init  config = Config()  try:  buttonOnOff = True  ledState = False  add\_battery\_button = True  count = 0  stateOnOff = False  init = False  end\_init = False  batA = Battery()  batB = Battery()  batC = Battery()  batteries = []  # Send some test  print("Come on man, press the button!")  config.lcd\_string(time.ctime(), Config.LCD\_LINE\_1)  while True:  buttonOnOff = GPIO.input(Config.buttonPin)  # ON  if buttonOnOff == False and stateOnOff == False and end\_init == False:  #GPIO.output(Config.LEDPinRed, True)  print("LED ON")  config.lcd\_string("Rasbperry Pi On", Config.LCD\_LINE\_1)  config.lcd\_string("choisir nb batterie", Config.LCD\_LINE\_2)  #launch eTG  authentication()  cookies = getCookies()  charging\_time = []  for cookie in cookies:  subscribe(cookie)  timeForCharge = getChargingTime(getCookieTemperature(cookie))  print(timeForCharge)  charging\_time.append(timeForCharge)  timeeforcharge\_formathms = str(datetime.timedelta(seconds=((timeForCharge \* 100) \* 60)))  print('il faudra ' + timeeforcharge\_formathms + ' minutes pour charger le ' + cookie.label)  end\_init = True  # get batteries  batA = Battery(id=1, label=cookies[0].label, max=10, charging\_time=charging\_time[0])  batB = Battery(id=2, label=cookies[1].label, max=10, charging\_time=charging\_time[1])  batC = Battery(id=3, label=cookies[2].label, max=10, charging\_time=charging\_time[2])  if not (batA == None) or (batB == None) or (batC == None):  batteries = [batA, batB, batC]  stateOnOff = True  time.sleep(0.5)  if buttonOnOff == False and stateOnOff == True and end\_init == False:  GPIO.output(Config.LEDPinRed, False)  print("LED OFF")  config.lcd\_string("Rasbperry Pi OFF", Config.LCD\_LINE\_1)  config.lcd\_string("Au revoir (^\_^)", Config.LCD\_LINE\_2)  # for simulation fin utilisation battery - reset  for bat in batteries:  if bat.etat == 100:  bat.etat = 0  stateOnOff = False  init = False  time.sleep(0.5)  # si allume  if stateOnOff == True:  init = True  # init  add\_battery\_button = GPIO.input(Config.buttonPinCount)  if add\_battery\_button == False:  init\_bn\_battery\_to\_charge += 1  if init\_bn\_battery\_to\_charge > 3:  init\_bn\_battery\_to\_charge = 1  config.lcd\_string("Nbr de chargeur", Config.LCD\_LINE\_1)  config.lcd\_string(str(init\_bn\_battery\_to\_charge), Config.LCD\_LINE\_2)  print("Number OF led: ", init\_bn\_battery\_to\_charge)  time.sleep(0.5)  buttonOnOff = GPIO.input(Config.buttonPin)  if buttonOnOff == False:  end\_init = True    elif end\_init == True:  #end\_init = False  # launch charge  if not (batteries == []):  batteries = [batA, batB, batC]  launch\_charge(init\_bn\_battery\_to\_charge = init\_bn\_battery\_to\_charge, p\_batteries=batteries)  if init\_bn\_battery\_to\_charge == 1:  GPIO.output(Config.LEDPinRed, True)  config.lcd\_string("Battry Cum", Config.LCD\_LINE\_1)  config.lcd\_string("100% (^\_^)", Config.LCD\_LINE\_2)    # for simulation fin utilisation battery - reset  init\_bn\_battery\_to\_charge = -1  for bat in batteries:  if bat.etat == 100:  bat.etat = 0  print(bat.id)  end\_init = False  except KeyboardInterrupt:  pass  finally:  config.lcd\_byte(0x01, Config.LCD\_CMD)  config.lcd\_string("Goodbye!", Config.LCD\_LINE\_1)  GPIO.cleanup() |

# Readme (cahier de recettes)

## Installation

### Pré-requis

Il faut un Raspberry et un kit Sense Mother avec quelques cookies.

### Raspberry

#### Installation librairies

Afin d’utiliser le raspberry avec un board (led , écran lcd , etc…) il faut télécharger la librairie GPIO.

sudo apt-get install python-dev python-rpi.gpio

Afin de communiquer avec la Mother, il faut installer l’API Sense ce qui permettra à un user Sense de publier et/ou de souscrire à des topics sur un ou plusieur cookie.

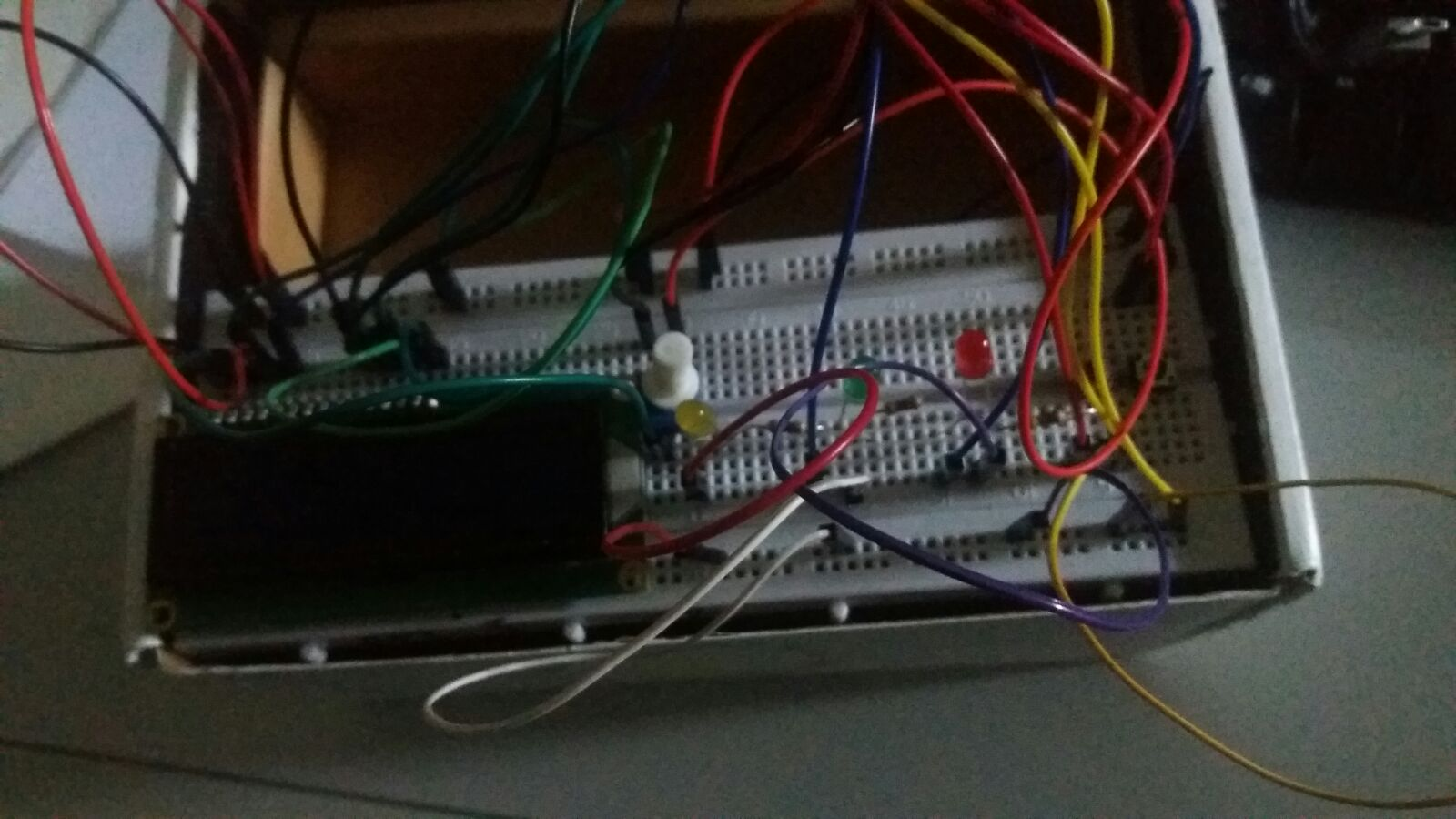
sudo pip install sense-python-client

Si la commande pip n’existe pas regarder ce tutoriel pour l’installer:

Cf(<https://www.saltycrane.com/blog/2010/02/how-install-pip-ubuntu/> )

#### Branchement du Board

Tuto : <http://www.circuitbasics.com/raspberry-pi-lcd-set-up-and-programming-in-python/>



### Sense Mother

Pour suivre les activités de la Mother il faut se connecter au Senseboard via le lien suivant :

Cf( <https://sen.se/login/> )

Les identifiants sont:

login: Kellydiote

password : master2isc

Pour utiliser l’API chaque user à un clé unique d’identification, pour le user “kellydiote” la clé est la suivante :

key : 4a5f9881e53ba0a3cf237fa1c20737a0dbfe5e99