Cartes de développement Elliptika

Description

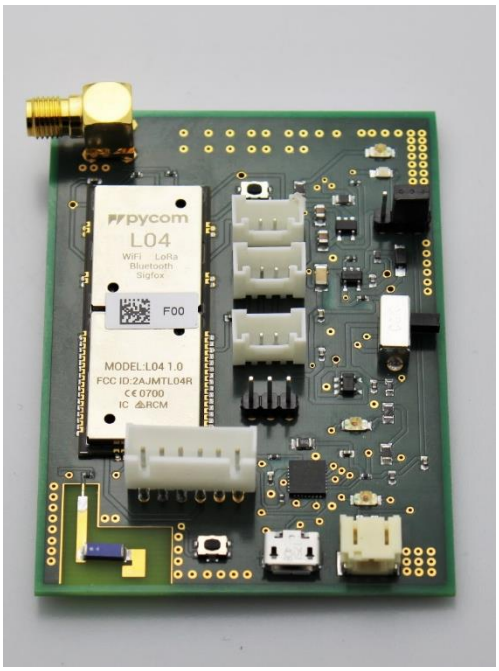
La carte modulaire Elliptika_IoT est notre dernière carte de développement, basée sur un module pycom qui inclut le module ESP32 d'Espressif, conçu pour les projets IoT. La carte de développement a été conçue avec des composants et du matériel de soudure de haute qualité pour permettre un prototypage facile. Nos cartes ne nécessitent pas de soudure ni même de câbles. Plug and play n'importe quel capteur directement sur la carte et commencez à exécuter votre application. Avec la carte modulaire Elliptika_IoT, le développement devient rapide, flexible et avec une consommation d'énergie ultra-faible.

Caractéristiques

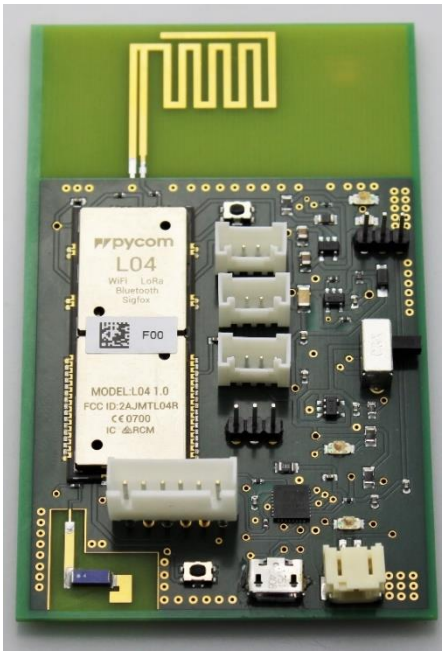
- USB - 1x port Micro USB : Port série USB vers UART.
- Chargeur de batterie LiPo intégré pour charger votre batterie.
- Alimentation - 3.3 à 5v
- LORA_antenne avec fréquence de fonctionnement de 868 MHz
- Wifi_antenna avec fréquence de fonctionnement de 2.4/5GHz
- Régulateur de tension
- LED pour la charge et pour le débogage
- Interface UART
- Interface I2C
- Interface SPI
- Commutateur : charge de la batterie ou alimentation de la carte
- Dimensions 95mm*56mm

Nos produits : voici un lien où nous présentons nos cartes électroniques

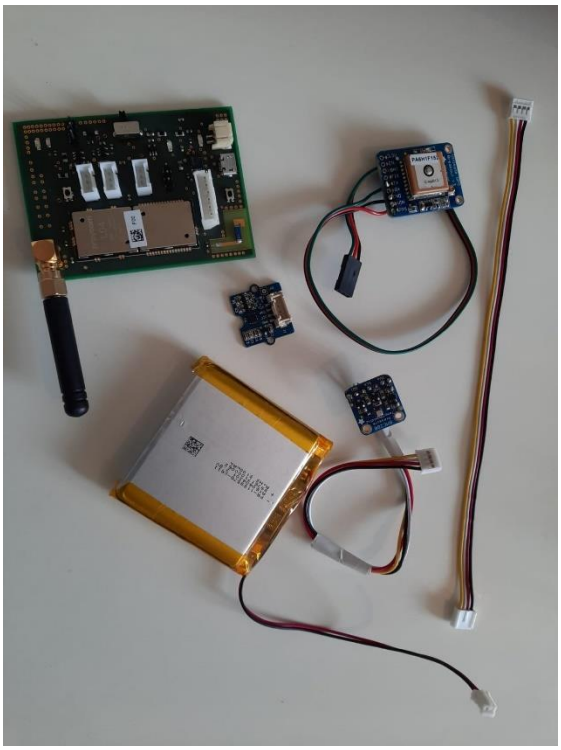
Version : 1

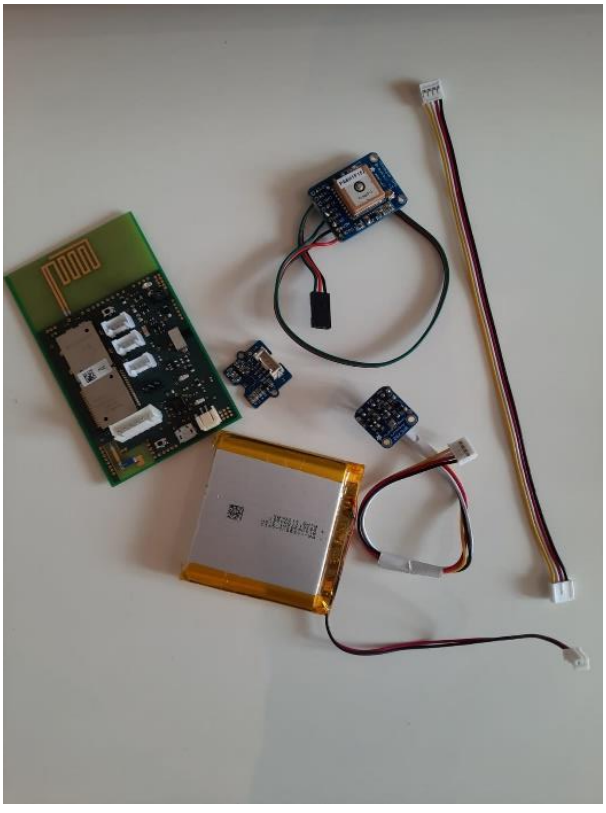


Version : 2

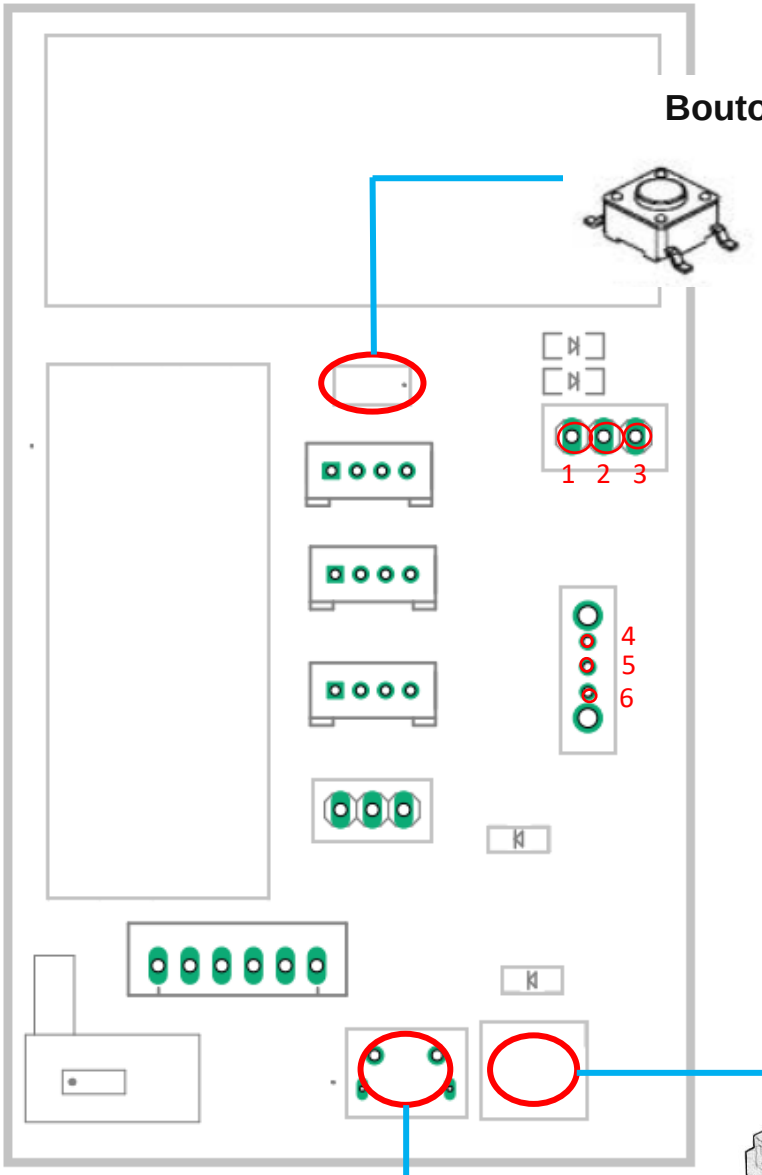


Kit _cartes_capteurs_alimentation





Connexions des cartes électroniques

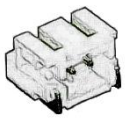


Bouton Reset

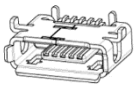
Tableau 1 :

Connexions	Caractéristiques
1-2	Charge de la batterie
2-3	Activation de la connexion usb/uart pour programmer la carte
4-5	Alimentation du circuit
5-6	Charge de la batterie

Connecteur de batterie



Connecteur usb



Pins

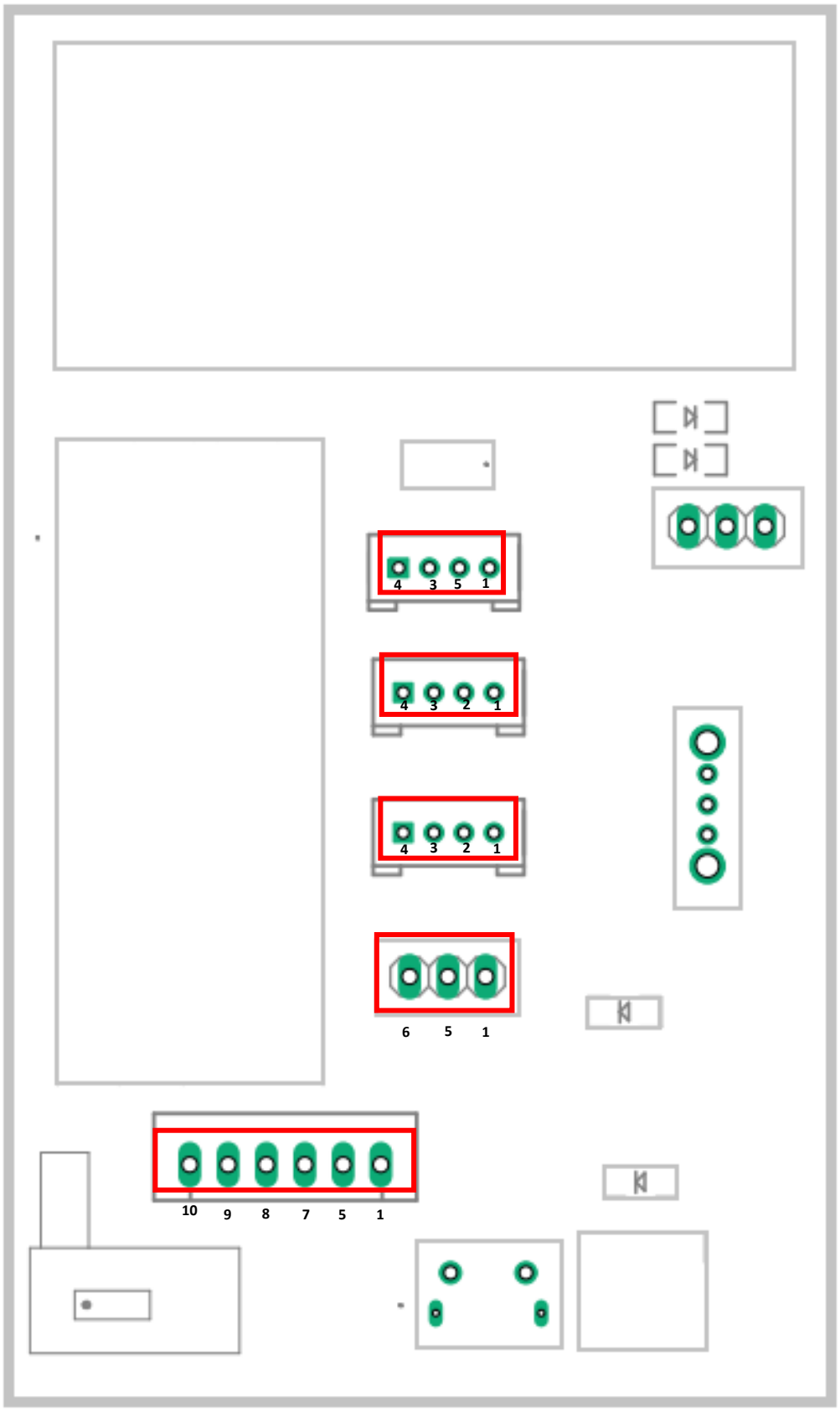


Tableau 2

Interfaces	Alimentation
I2C_1	5V
I2C_2	3V
I2C_3	3V
UART	5V
SPI	5V

Tableau 3

Pin	signal	fonction	ADC
1	GND	ground	-
2	3V3	Power supply 3.3V	-
3	SDA P9	Serial Data Line	-
4	SCL P10	Serial Clock Line	-
5	5V	Power supply 5V	-
6	RXD P3	Receive	-
7	CS P8	Chip Select	-
8	SCL P19	Clock	Resource disponible
9	MOSI P20 SDI Data in	Master Output Slave Input	Resource disponible
10	MISO P21 SD0 DATA OUT	Master Input Slave Output	-

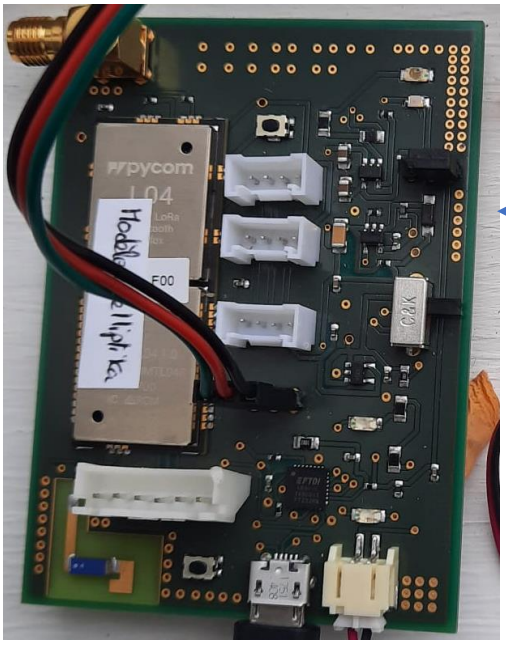
Interfaçage du PA6H1F152 avec la carte électronique

Câblage :

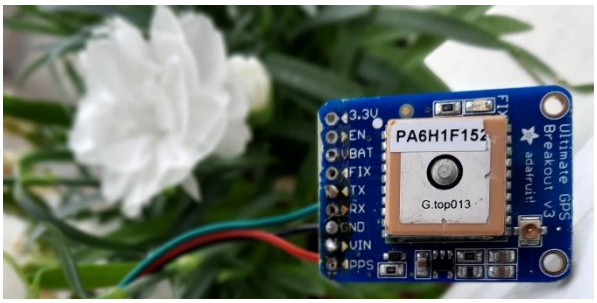
Rouge : Connecter VIN à +5V

Noir : Connecter GND au Ground

Vert : Connecter GPS TX au RX sur la carte



Câblage



capteur	Constructeur	lien
gps PA6H1F152	adafruit	adafruit-ultimate-gps.pdf

Notez Bien : Il est nécessaire que le capteur GPS Soit déporté à l'extérieur pour avoir les résultats voulus

Programmes

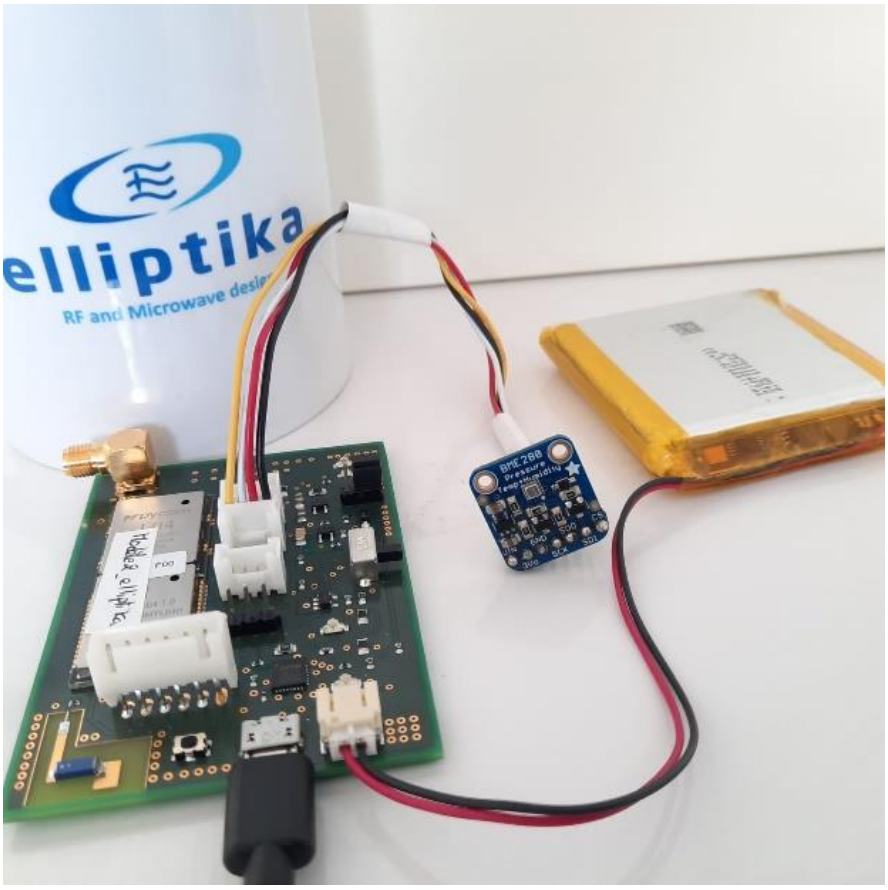
main.py

```
from machine import Pin
import pycom
import time
from machine import UART
import ubinascii
import machine
from machine import RTC
import math
import gc
import pa6h
uart = UART(1, baudrate=9600, pins=('P4','P3')) # init with given baudrate
while True:
   dataReceived=uart.read()
   while dataReceived == None :
      dataReceived=uart.read()
   time.sleep(3)
   trame=pa6h.lecture_donnees(uart) # lit les donnees
   print("trame :", trame)
   heure=pa6h.extraire_heure(trame)
   print("heure : "+heure)
```

pa6h.py

```
# retourne une trame GPGGA lue par la liaison uart
def lecture_donnees(uart):
    test=b''
    while test!=b'GPGGA':        # verifie que la trame lue est la trame voulue, sinon lit la prochaine
        trame=uart.readline()    # lit une ligne
        test=trame[1:6]
    trame=trame.decode("utf-8")   # permet de travailler avec des strings
    return trame
# retourne la partie de la trame relative a l'heure
def extraire_heure(trame):
    heure=trame[7:17]
    return heure
# retourne la partie de la trame relative a la latitude
def extraire_latitude(trame):
    latitude=trame[18:27]+trame[28]
    return latitude
# retourne la partie de la trame relative a la longitude
def extraire_longitude(trame):
    longitude=trame[30:40]+trame[41]
    return longitude
# retourne la latitude et la longitude dans une seule chaine
def extraire_position(trame):
    return extraire_latitude(trame)+' - '+extraire_longitude(trame)
```

Interfaçage du BM280 avec la carte



Câblage

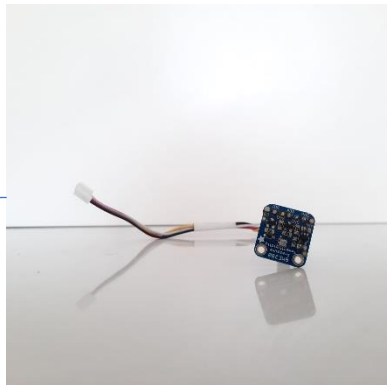
rouge: Connecter VIN au : 3.3V, 5V

noir: Connecter GND au Ground & SDO

blanc: Connecter SDA au SDI

jaune: Connecter SCL au SCK

cablage



Capteur	Constructeur	lien
BM280	**adafruit**	ADAFRUIT

Programmes

Main.py

```
import bme280
import micropython
import pycom
import ubinascii
i2c=bme280.instancier_i2c()

bme280.parametrer_capteur(i2c)# paramètre les registres de contrôle du capteur
buf=bme280.lecture_donnees(i2c) # lit les données et les place dans un buffer
# découpage du buffer
temp=buf[3:6]
# calculs des valeurs a partir des données brutes
temp=bme280.calcul_temp(temp,i2c)
# affichage
print("temp :",temp,"degC")    print("trame :", trame)
   heure=pa6h.extraire_heure(trame)
   print("heure : "+heure)
```

Bm280.py

```
from machine import I2C
import micropython
import pycom
import ubinascii
BME280_I2C_ADDRESSE=0x76 # si SD0 a la masse

# BME280_I2C_ADDRESSE=0x77 # si SD0 a 3.3V

##### Adresses registres BME280 ####

BME280_WHOAMI_REG=0xD0

BME280_TEMPERATURE_CALIB_REG_START=0x88

BME280_PRESSION_CALIB_REG_START=0x8E

BME280_HUMIDITE_CALIB_REG_1=0xA1

BME280_HUMIDITE_CALIB_REG_2_START=0xE1

BME280_CONTROLE_REG_MESURE=0xF4

BME280_CONTROLE_REG_HUMIDITE=0xF2

BME280_DONNEES_REG_START=0xF7
```

```
# transforme des donnees non signees en valeurs signees
def unsigned_to_signed(value):
    if ((value&0x8000)==0x8000):
        value=(-1)*(0xFFFF-value+1)
        return value
    else:
        return value
# initialise la connexion i2c
def instancier_i2c():
    i2c = I2C(0, I2C.MASTER, baudrate=100000)
    #par défaut P9 -> SDI et P10 -> SCK et SD0 a la masse
    return i2c
# ecrit des valeurs dans les registres de controle du capteur (cf doc BME280 pour les details)
def parametrer_capteur(i2c):
    i2c.writeto_mem(BME280_I2C_ADDRESSE, BME280_CONTROLE_REG_MESURE, 0x27) # ecriture parametres d'acquisition


# lit la valeur du registre WhoAmI, et la compare a la valeur theorique
def test_capteur(i2c):
    buf=bytearray(1)
    i2c.readfrom_mem_into(BME280_I2C_ADDRESSE, BME280_WHOAMI_REG, buf) # WhoAmI, value = 0x60
    WhoAmI=int.from_bytes(buf, 0)
    return WhoAmI==0x60
# lit les donnees brutes dans le registre correspondant
def lecture_donnees(i2c):
    buf=bytearray(8)
    i2c.readfrom_mem_into(BME280_I2C_ADDRESSE, BME280_DONNEES_REG_START, buf) # donnees
    return buf
```

```
# calcule la valeur de la temperature a partir des donnees brutes

def calcul_temp(temp,i2c):

    temp=(int.from_bytes(temp, 0))>>4

    buf=bytearray(6)

    i2c.readfrom_mem_into(BME280_I2C_ADDRESSE,
BME280_TEMPERATURE_CALIB_REG_START, buf) # lit les coefficients

    buf_bis=int.from_bytes(buf, 0)

    octet0=(buf_bis&0xFF0000000000)>>40

    octet1=(buf_bis&0xFF00000000)>>32

    octet2=(buf_bis&0xFF000000)>>24

    octet3=(buf_bis&0xFF0000)>>16

    octet4=(buf_bis&0xFF00)>>8

    octet5=buf_bis&0xFF

    dig_t1=(octet1<<8)|octet0

    dig_t2=(octet3<<8)|octet2

    dig_t3=(octet5<<8)|octet4

    var_1=(((temp>>3)-(dig_t1<<1))*dig_t2)>>11

    var_2=(((((temp>>4)-dig_t1)*((temp>>4)-(dig_t1)))>>12)*(dig_t3))>>14

    global t_fine

    t_fine=var_1+var_2

    temp=((t_fine*5+128)>>8)/100

    return temp
```

Interfaçage du HMC5883L 3-Axis avec la carte

Câblage

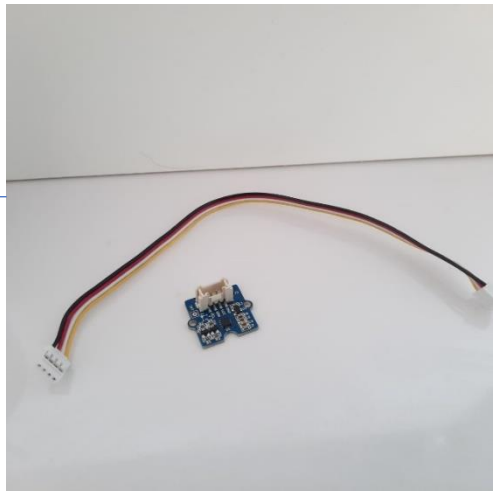
Rouge: Connecter VIN au : 3.3V, 5V

Noir: Connecter GND au Ground & SDO

Blanc: Connecter SDA au SDI

Jaune: Connecter SCL au SCK

Câblage



Capteur	Constructeur	Lien
HMC5883L 3-Axis	Adafruit	ADAfruit

Programmes

Main.py

```
import hmc5883l
import time
import gc

############################################################################
#               VARIABLES GLOBALES
############################################################################

m = hmc5883l.HMC5883L()

############################################################################
#               BOUCLE PRINCIPALE
############################################################################

while True:
        m.readAxes() #lit les donnees
        m.heading()  #corrige les donnees au format [0;360]°C
        print(m)
        time.sleep_ms(1000)
        gc.collect() #appel du garbage collector
```

hmc5883l.py

```
from machine import I2C
from array import array
import math
import gc
import time

###########################################################################
#               REGISTRES
###########################################################################

HMC5883L_sampling_mode_continous = bytes([0x00])
HMC5883L_sampling_mode_single    = bytes([0x01])
HMC5883L_sampling_mode_idle      = bytes([0x02])

HMC5883L_samples_1 = 0
HMC5883L_samples_2 = 32
HMC5883L_samples_4 = 64
HMC5883L_samples_8 = 96

HMC5883L_rate_00_75 = 0
HMC5883L_rate_01_50 = 4
HMC5883L_rate_03_00 = 8
HMC5883L_rate_07_50 = 12
HMC5883L_rate_15_00 = 16

HMC5883L_rate_75_00 = 20


HMC5883L_measurement_mode_bias_disabled = 0
HMC5883L_measurement_mode_bias_positive = 1
HMC5883L_measurement_mode_bias_negative = 2
HMC5883L_gauss_gain = {
        "0.88": [0, 0.73],
        "1.3":  [32, 0.92],
        "1.9":  [64, 1.22],
        "2.5":  [96, 1.52],
        "4.0":  [128, 2.27],
        "4.7":  [160, 2.56],
        "5.6":  [192, 3.03],
        "8.1":  [224, 4.35]
        }
def complement2toInt(value, len):
        if (value & (1 << len - 1)):
                value = value - (1<<len)
        return value
```

```
class HMC5883L():

	def __init__(self, busI2C=None, port=0, sensor_address=30, gauss="1.3", declinationDegrees=-1,
declinationMinutes=21):
		if busI2C==None:
			self.bus = I2C(port, I2C.MASTER, baudrate=100000) # max 400000 MR7 can
change it
		else:
			self.bus = busI2C
		self.address = sensor_address
		self.setDeclination(declinationDegrees, declinationMinutes)
		self.headingDeg = None
		self.__data = bytearray([0]*6)
		self.x = 0
		self.y = 0
		self.z = 0
		self.__error = -4096
		self.wasError = 0
		self.samples = HMC5883L_samples_8
		self.rate = HMC5883L_rate_15_00
		self.bias = HMC5883L_measurement_mode_bias_disabled
		self.setRegA()
		self.gauss=gauss
		self.gauss_mask, self.gauss_scale=HMC5883L_gauss_gain[gauss]
		self.bus.writeto_mem(self.address, 0x01, bytes([self.gauss_mask]))
		self.setMode(HMC5883L_sampling_mode_continous)
		time.sleep_ms(67) #1/15 Hz

	def setDeclination(self, degrees, minutes):
		self.declDegrees = degrees
		self.declMinutes = minutes
		self.declination = (degrees + minutes / 60.0) * math.pi / 180.0
	def setRegA(self):
		self.bus.writeto_mem(self.address, 0x02, bytes([self.bias | self.samples | self.rate]))
	def setSamples(self, samples):
		self.samples = samples
		self.setRegA()

	def setRate(self, rate):
		self.rate = rate
		self.setRegA()

	def setBias(self, bias):
		self.bias = bias
		self.setRegA()

	def setMode(self, mode):
		self.bus.writeto_mem(self.address, 0x02, mode)
```

```
def declination(self):
        return (self.declDegrees, self.declMinutes)

def convert(self, data, offset):
        val = complement2toInt(data[offset] << 8 | data[offset+1], 16)
        if val == self.__error: return None
        return round(val * self.gauss_scale, 4)

def readAxes(self):
        self.wasError = 0
        self.bus.readfrom_mem_into(self.address, 0x03, self.__data)
        #self.x = self.convert(self.__data, 0)
        self.x = self.__data[0] << 8 | self.__data[0+1]
        if (self.x & (1 << 16 - 1)):
                self.x-= (1<<16)
        if self.x == self.__error:
                self.x=None
                self.wasError = 1
        else:
                self.x=round(self.x * self.gauss_scale, 4)

        #self.z = self.convert(self.__data, 2)
        self.z = self.__data[2] << 8 | self.__data[2+1]
        if (self.z & (1 << 16 - 1)):
                self.z-= (1<<16)
        if self.z == self.__error:
                self.z=None
                self.wasError = 1
        else:
                self.z=round(self.z * self.gauss_scale, 4)
        #self.y = self.convert(self.__data, 4)
        self.y = self.__data[4] << 8 | self.__data[4+1]
        if (self.y & (1 << 16 - 1)):
                self.y-= (1<<16)
        if self.y == self.__error:
                self.y=None
                self.wasError = 1
        else:
                self.y=round(self.y * self.gauss_scale, 4)
```

```
	def heading(self):
		'''
		1° to 2° compass heading accuracy
		first call self.readAxes()
		'''
		headingRad = math.atan2(self.y, self.x)
		headingRad += self.declination

		# correct to range 0-360
		if headingRad < 0:
			headingRad += 2 * math.pi
		elif headingRad > 2 * math.pi:
			headingRad -= 2 * math.pi

		self.headingDeg = headingRad * 180 / math.pi

	def __str__(self):
		'''
		first call:
		self.readAxes()
		self.heading()
		'''
		return "X: " + str(self.x) + ", Y: " + str(self.y) + ", Z: " + str(self.z) + " - Heading: " +
str(self.headingDeg) + ", Declination: " + str((self.declDegrees,self.declMinutes)) + "\n"
```

Liens :

Tutoriel 1 : Begin Chating with your Board

Tutoriel 2 : LORA Using The Board with TTN+Node-red .