Sommaire :

-Installer la Raspberry et la connecter à constellation

-Présentation du module balance

-Package python

-partage

-explication

Installer la Raspberry  et la connecter à constellation :

Télécharger Raspbian sur le site Raspberry

Une fois Raspbian lancé et vous êtes avec l’interface graphique du compte « pi »

Lancer ces lignes de codes pour mettre à jour votre Rasbian

|  |
| --- |
| sudo apt-get update |
| sudo apt-get install rpi-update |

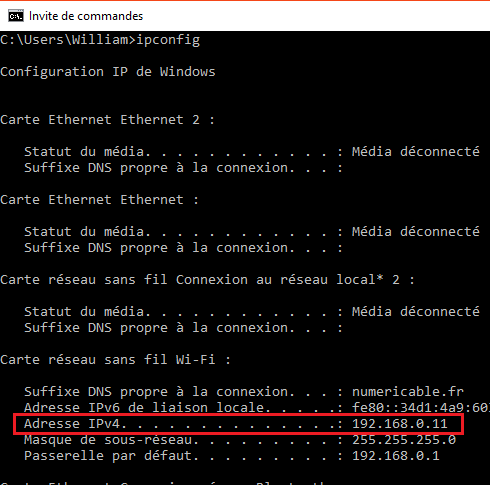
Suivre le tutoriel suivant : <https://developer.myconstellation.io/getting-started/ajouter-des-sentinelles/>

NB : pour se connecter à une constellation en WI-FI, il est demandé l’URI du serveur. Elle correspond à l’adresse : http://<votreAdresseIP>:8088/

Pour récupérer votre adresse IP, ouvrez l’invite de commande windows, et tapez « ipconfig », puis entrez :

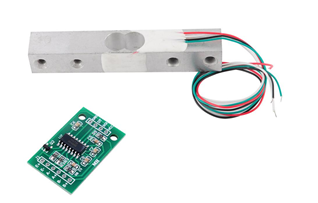
Les informations sur les adresses IP de votre poste sur ce réseau s’affiche, vous devrez utiliser l’adresse IPv4.

Soit dans le cas présent : <http://192.168.0.11:8088/>

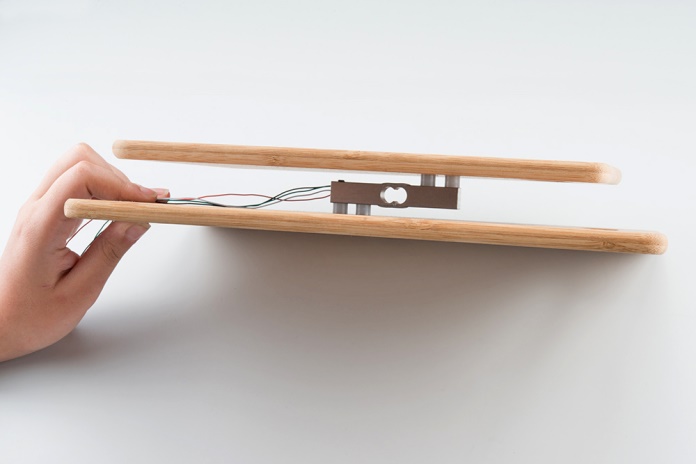
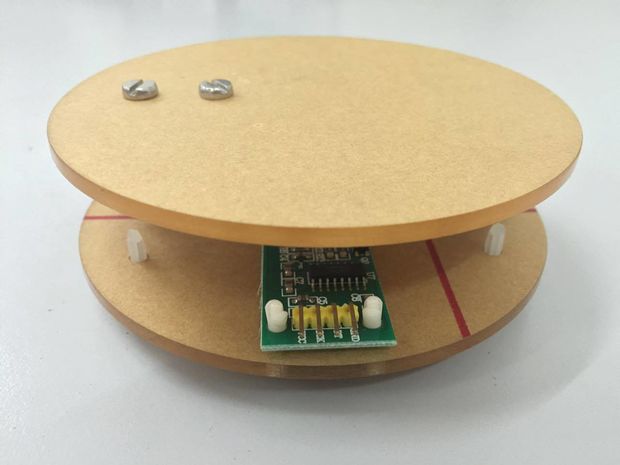


Notez que cette adresse change chaque fois que vous changez de réseau, si c’est votre cas, il vous faudra donc réinstaller la sentinelle.

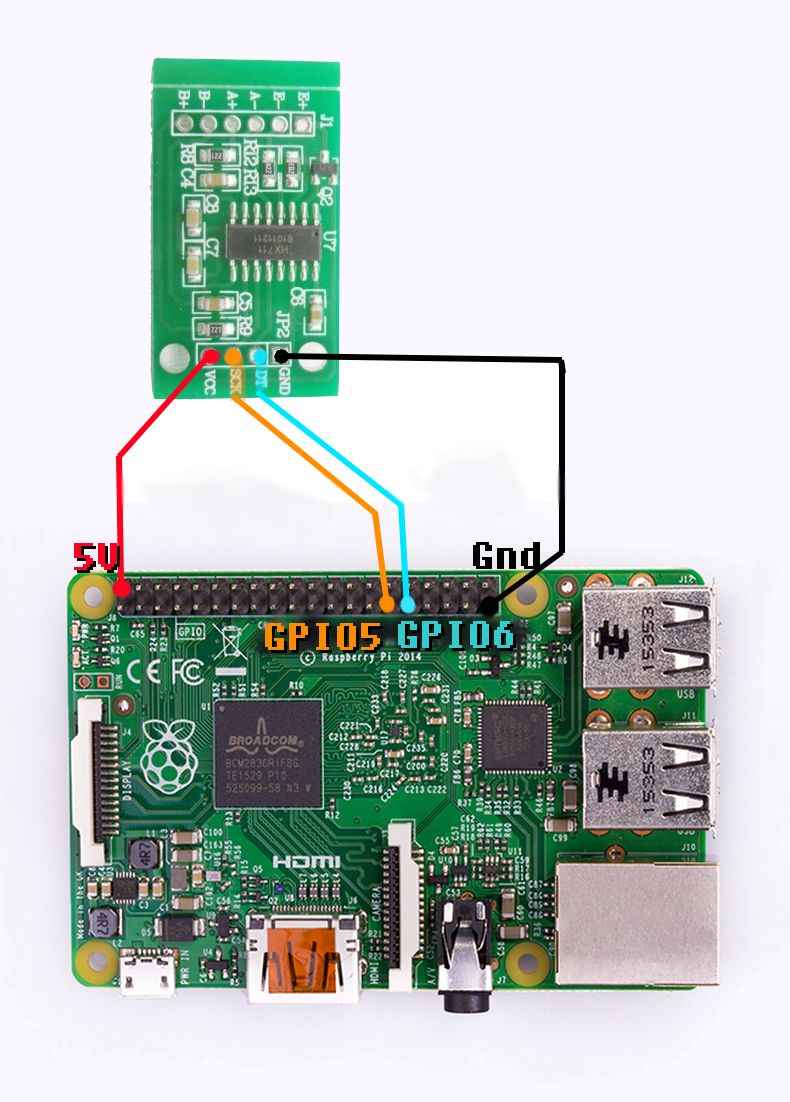
Le module Hx711:

Il s’agit d’une cellule de pesage de 0 à 3kg ainsi que d’une carte hx711 : c’est un amplificateur convertisseur digital 24 bits.

Ils peuvent être vendus sous la forme suivante, où le plateau supérieur est celui sur lequel la masse est mesurée :



Le tutoriel se base sur ce montage (le plateau supérieur peut être remplacé, selon l’utilisation du module dans votre projet).

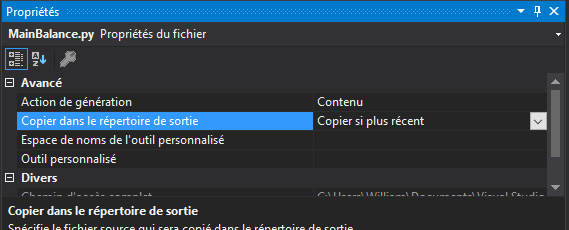


Nous utilisons une Raspberry pi 2 modèle B+, les branchements changent en fonction du modèle. Prenez soin de vérifier l’emplacement des broches de votre modèle Raspberry.

Voici le code du package installé sur la sentinelle. Il se base sur un code python récupéré sur ce lien :

<https://github.com/ggurov/hx711> faisant fonctionner la balance depuis la Raspberry en python.

Sur Visual Studio, créez un nouveau projet « constellation package python ». Dans le dossier Script, Renommez le fichier « Demo.py » selon vos préférences, puis copiez-y le code suivant :

(NB :Dans les propriétés du package, il faut choisir « copier si plus récent » ou « toujours copier » dans l’option « copier dans le répertoire de sortie », voir comment créer un package python : <https://developer.myconstellation.io/getting-started/creez-votre-premier-package-constellation-en-python/> )

import Constellation

import time

import numpy

import RPi.GPIO as GPIO

import sys

#Nous avons une Raspberry B+ donc DT sur le pin 29(soit le GPIO 5), SCK sur le pin 31 (soit le GPIO 6).

#(5V sur pin 1 et gnd sur 39 par exemple)

GPIO.setwarnings(False)

class HX711:

def \_\_init\_\_(self, dout, pd\_sck, gain=128):

self.PD\_SCK = pd\_sck

self.DOUT = dout

GPIO.setmode(GPIO.BCM)

GPIO.setup(self.PD\_SCK, GPIO.OUT)

GPIO.setup(self.DOUT, GPIO.IN)

self.GAIN = 0

self.REFERENCE\_UNIT = 1 # The value returned by the hx711 that corresponds to your reference unit AFTER dividing by the SCALE.

self.OFFSET = 1

self.lastVal = long(0)

self.LSByte = [2, -1, -1]

self.MSByte = [0, 3, 1]

self.MSBit = [0, 8, 1]

self.LSBit = [7, -1, -1]

self.byte\_range\_values = self.LSByte

self.bit\_range\_values = self.MSBit

self.set\_gain(gain)

time.sleep(1)

def is\_ready(self):

return GPIO.input(self.DOUT) == 0

def set\_gain(self, gain):

if gain is 128:

self.GAIN = 1

elif gain is 64:

self.GAIN = 3

elif gain is 32:

self.GAIN = 2

GPIO.output(self.PD\_SCK, False)

self.read()

def createBoolList(self, size=8):

ret = []

for i in range(8):

ret.append(False)

return ret

def read(self):

while not self.is\_ready():

pass

dataBits = [self.createBoolList(), self.createBoolList(), self.createBoolList()]

dataBytes = [0x0] \* 4

for j in range(self.byte\_range\_values[0], self.byte\_range\_values[1], self.byte\_range\_values[2]):

for i in range(self.bit\_range\_values[0], self.bit\_range\_values[1], self.bit\_range\_values[2]):

GPIO.output(self.PD\_SCK, True)

dataBits[j][i] = GPIO.input(self.DOUT)

GPIO.output(self.PD\_SCK, False)

dataBytes[j] = numpy.packbits(numpy.uint8(dataBits[j]))

#set channel and gain factor for next reading

for i in range(self.GAIN):

GPIO.output(self.PD\_SCK, True)

GPIO.output(self.PD\_SCK, False)

#check for all 1

#if all(item is True for item in dataBits[0]):

# return long(self.lastVal)

dataBytes[2] ^= 0x80

return dataBytes

def get\_binary\_string(self):

binary\_format = "{0:b}"

np\_arr8 = self.read\_np\_arr8()

binary\_string = ""

for i in range(4):

# binary\_segment = binary\_format.format(np\_arr8[i])

binary\_segment = format(np\_arr8[i], '#010b')

binary\_string += binary\_segment + " "

return binary\_string

def get\_np\_arr8\_string(self):

np\_arr8 = self.read\_np\_arr8()

np\_arr8\_string = "[";

comma = ", "

for i in range(4):

if i is 3:

comma = ""

np\_arr8\_string += str(np\_arr8[i]) + comma

np\_arr8\_string += "]";

return np\_arr8\_string

def read\_np\_arr8(self):

dataBytes = self.read()

np\_arr8 = numpy.uint8(dataBytes)

return np\_arr8

def read\_long(self):

np\_arr8 = self.read\_np\_arr8()

np\_arr32 = np\_arr8.view('uint32')

self.lastVal = np\_arr32

return long(self.lastVal)

def read\_average(self, times=3):

values = long(0)

for i in range(times):

values += self.read\_long()

return values / times

def get\_value(self, times=3):

return self.read\_average(times) - self.OFFSET

def get\_weight(self, times=3):

value = self.get\_value(times)

value = value / self.REFERENCE\_UNIT

return value

def tare(self, times=15):

# Backup REFERENCE\_UNIT value

reference\_unit = self.REFERENCE\_UNIT

self.set\_reference\_unit(1)

value = self.read\_average(times)

self.set\_offset(value)

self.set\_reference\_unit(reference\_unit)

def set\_reading\_format(self, byte\_format="LSB", bit\_format="MSB"):

if byte\_format == "LSB":

self.byte\_range\_values = self.LSByte

elif byte\_format == "MSB":

self.byte\_range\_values = self.MSByte

if bit\_format == "LSB":

self.bit\_range\_values = self.LSBit

elif bit\_format == "MSB":

self.bit\_range\_values = self.MSBit

def set\_offset(self, offset):

self.OFFSET = offset

def set\_reference\_unit(self, reference\_unit):

self.REFERENCE\_UNIT = reference\_unit

# HX711 datasheet states that setting the pin on high for a more than 60 microseconds would power off the chip.

# I used 100 microseconds, just in case.

# I've found it is good practice to reset the hx711 if it wasn't used for more than a few seconds.

def power\_down(self):

GPIO.output(self.PD\_SCK, False)

GPIO.output(self.PD\_SCK, True)

time.sleep(0.0001)

def power\_up(self):

GPIO.output(self.PD\_SCK, False)

time.sleep(0.0001)

def reset(self):

self.power\_down()

self.power\_up()

Constellation.StartAsync() #fonction qui démarre le code dans la Constellation

hx = HX711(5,6) #choix du GPIO 5 pour la broche DT et du GPIO 6 pour la broche SCK

hx.set\_reading\_format("LSB", "MSB")

hx.set\_reference\_unit(113)

hx.set\_reference\_unit(92)

hx.reset() #fonctions d'initialisation du code de la balance.

hx.tare() #On met la balance à 0

intervalle = int(Constellation.GetSetting("intervalle")) #récupère la valeur de l'intervalle de lecture, choisie dans les setting au déploiement du package

intervalle=0.001\*intervalle #on converti l'entier en millisecondes

while Constellation.IsRunning: #début de la boucle

volt = hx.get\_weight(5) #lit la valeur de la balance sur le GPIO 5(c'est une tension)

mass = 0.124\*volt #transforme l'information reçue en grammes (le coefficient change selon la balance, pour le trouver, faire des tests avec une balance déjà étalonnée ou utiliser des objets dont la masse est connue, puis trouver le coefficient. Attention! ce n'est pas toujours de forme affine!)

hx.power\_down()

hx.power\_up() #reinitialise les broches GPIO

time.sleep(intervalle)

Constellation.PushStateObject("Balance", mass, "g", lifetime=10\*intervalle)

Constellation.StartAsync()

Après cette fonction, la constellation « lit les lignes de codes ».

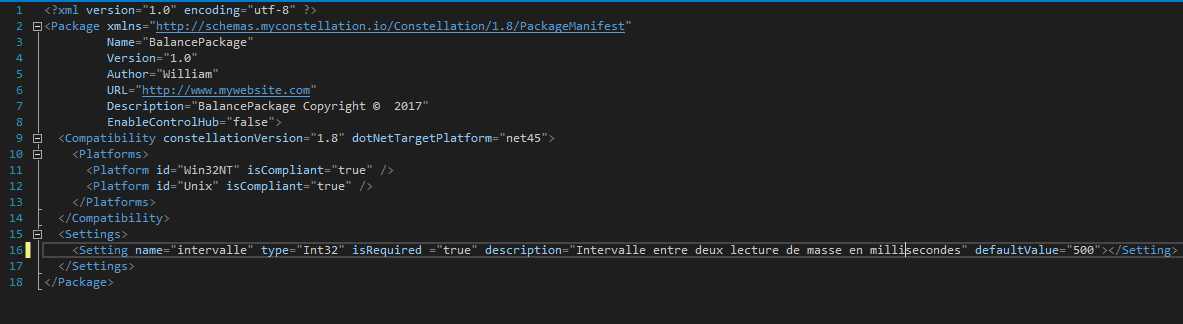
while Constellation.IsRunning:

Permet de lancer une boucle, tant qu’on est connecté à la constellation.

Constellation.PushStateObject("Balance", mass, "g", lifetime=10\*intervalle)

Créé un state object. Le state object est rafraichi à chaque fois que cette fonction est lancée.

Notez que la variable intervalle, qui détermine l’intervalle de temps entre deux mesures, est définie en “Setting”, c’est à dire qu’elle sera choisie au déploiement du package sur la constellation. Cela nécessite d’ajouter une ligne de code dans le fichier PackageInfo.xml de votre solution :



<Setting name="intervalle" type="Int32" isRequired ="true" description="Intervalle entre deux lecture de masse en millisecondes" defaultValue="500"></Setting>

Cette ligne est à ajouter dans la balise : <Settings> … </Settings>

Vous pouvez ajouter des informations complémentaires :

name=le nom de l’information qui vous sera demandé au déploiement du package.

type =informe sur le type de la variable.

isRequired = true : cette information indique que la variable doit nécessairement se voir attribuée une valeur, le package ne peut pas être installé dans le cas contraire (cela affichera un petit message d’erreur en rouge)

description = affiche de informations supplémentaires

defaultValue = La valeur par défaut, elle s’affichera en grisée au moment de choisir la valeur de l’intervalle.