De Arduino ? Python - mesures de distance (HC-SR04)

April 4, 2019

1 Communiquer avec une carte Arduino :

1.1 1 - Récupérer les données issues du microcontrôleur :

Les données issues de la carte Arduino peuvent être récupérées et traitées avec Python.

Il faut au préalable installer le module **Serial** (https://riptutorial.com/fr/python/topic/5744/communicatio serie-python–pyserial-) qui permet de gérer les ports série : pour cela, il faut retourner dans **Anaconda-Navigator**, puis dans **Environments** (en haut à gauche), choisir "not installed" et rechercher **pyserial**.

Une fois l'installation terminée, il faut importer le module **Serial** et le module **time** :

```
In [1]: import serial as sr
        import time
```

Il faut ensuite préciser à Python : - le **port** utilisé et le **taux de transfert** : le port est indiqué en bas à droite de la fenêtre **Arduino** ; - le **taux de transfert** choisi et indiqué dans le programme Arduino IDE.

2 TRÈS IMPORTANT :

Le moniteur série d'Arduino doit toujours être fermé avant d'activer le port dans Python : la carte Arduino ne peut pas communiquer simultanément avec le sketch Arduino et avec Python.

De même le port série doit être désactivé par la commande *port_serie.close()* dans Python si on veut visualiser le moniteur série !

```
In [2]: port_serie = sr.Serial(port = "/dev/cu.usbmodem1D13301", baudrate = "115200")
```

On va ensuite lire les informations qui sont envoyées par la carte sous la forme d'une chaine de caractères.

On va utiliser le montage de mesure de distances avec le télémètre à ultrason (capteur HC-SR04) (programme ci-dessous) :

```
const byte TRIGGER_PIN = 2; // entrée numérique 2 affectée au trig
const byte ECHO_PIN = 3;    // entrée numérique 3 affectées à l'écho

const unsigned long MEASURE_TIMEOUT = 25000UL; // délais avant annulation de mesure : 25ms corre
```

```
const float SOUND_SPEED = 340.0 / 1000; // Vitesse du son dans l'air en mm/us

int compt = 0; // on crée un entier qui servira de compteur initialisé à 0

void setup() {

  Serial.begin(115200); //Initialisation du port série et définition du taux de transfert

  /* Initialise les broches */
  pinMode(TRIGGER_PIN, OUTPUT);   // Activation de la broche de trig
  digitalWrite(TRIGGER_PIN, LOW); // Initialisation de la broche de trig
  pinMode(ECHO_PIN, INPUT);       // Activation de la broche de d'écho
  }

void loop() {
  compt += 1; // incrémente le compteur d'une unité

  digitalWrite(TRIGGER_PIN, HIGH); // envoi d'une impulsion dans le trig
  delayMicroseconds(10);           // d'une durée de 10 µs
  digitalWrite(TRIGGER_PIN, LOW);  // initialisation de la broche d'écho

  long measure = pulseIn(ECHO_PIN, HIGH, MEASURE_TIMEOUT); // mesure de la durée entre le trig e

  float distance_mm = measure / 2.0 * SOUND_SPEED;  // calcul de la distance en mm

  Serial.print(compt);            // affiche le numéro de la mesure
  Serial.print("\t");             // affiche une tabulation
  Serial.println(distance_mm,0);  // affiche la distance en mm sans chiffre après la virgule
  delay(100);                     // génrère un délais de 100 ms

  if (compt >9999){
    exit(0) ;                     // impose l'arrêt des mesures après 10000 mesures
}
}
```

On affiche ici l'information brute récupérée de la carte Arduino jusqu'au premier retour à la ligne :

```
In [3]: print(port_serie.readline())

        # b : mot binaire
        # '' : chaine de caractère
        # \r = retour chariot
        # \n = retour à la ligne
```

```
b'0\t988\r\n'
```

On va ensuite récupérer les données envoyées par l'Arduino et les stocker dans une liste à l'aide d'une boucle itérative. On le fait ici sur seulement 5 valeurs pour l'exemple.

```
In [4]: distance = []                          # on génère une liste nommée distance
        for i in range(5):                     # pour i de 0 à 4 (5 premières lignes en partant d
            d = port_serie.readline()      # on crée la variable "mesure" qui prend la valeur de
            distance.append(d)                 # on ajoute la valeur "d" à la liste distance grâc
            print(d.split())                   # la fonction split() permet de découper la chaine
        port_serie.close()                     # on ferme le port série
```

```
[b'211', b'1001']
[b'212', b'1002']
[b'213', b'1001']
[b'214', b'993']
[b'215', b'990']
```

On peut maintenant automatiser l'ensemble pour récupérer à chaque instant t la mesure de la distance d :

NB : Tant que les crochets à gauche de ce cadre contiennent une *, c'est que l'aquisition n'est pas terminée.

```
In [34]: port_serie = sr.Serial(port = "/dev/cu.usbmodem1D13301", baudrate = "115200")

         port_serie.setDTR(False)                  # Procédure non indispensable
         time.sleep(0.1)                           # de
         port_serie.setDTR(True)                   # réinitialisation
         port_serie.flushInput()                   # du port

         distance = []                             # génère une liste nommée distance

         N = int(input('Nombre de mesures souhaité :'))
         print('Patientez...')

         for i in range(N):                        # prends les N premières valeurs
             val = port_serie.readline().split() # place dans la variable "val" les lignes décon
             try:                                  # permet de sauter à la valeur suivante en cas
                 if float(val[1]) == 0.:
                     pass
                 else:
                     d = float(val[1])
                     distance.append(d)                # on ajoute d à la liste des distances
             except:
                 pass                              # sauf en cas d'erreur
         print('Acquisition terminée')
         port_serie.close()                        # on ferme le port série
```

```
Nombre de mesures souhaité : 1000


Patientez...
Acquisition terminée
```

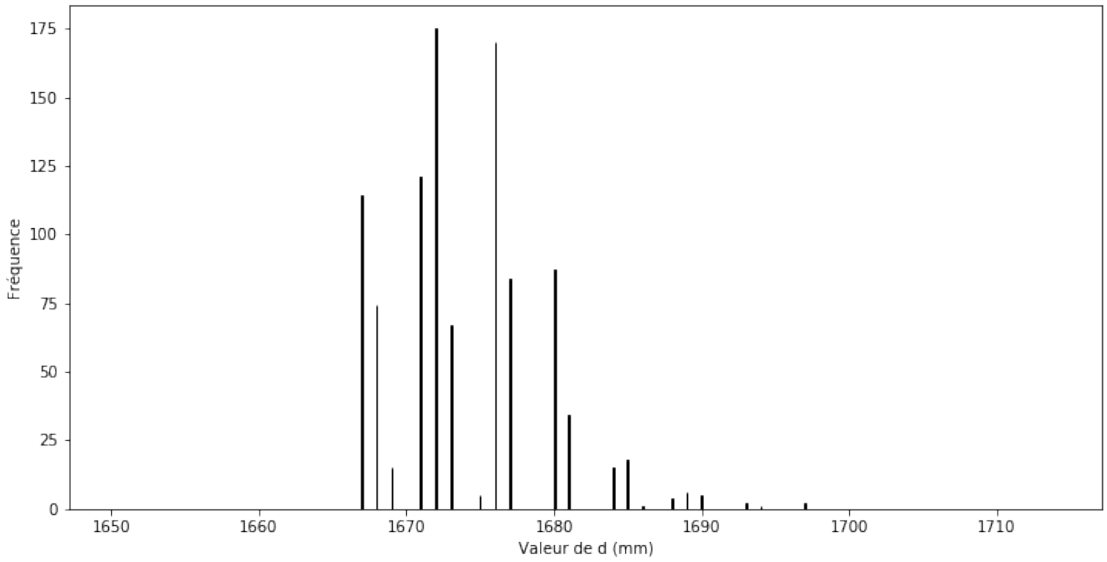
2.1 2 - Effectuer le traitement statistique des données :

On peut analyser la variabilité des mesures de la distance en traçant un histogramme d'abord avec autant de classes statistiques que de mesures pour vérifier que les mesures sont cohérentes :

```
In [35]: import matplotlib.pyplot as plt

         N_classes=int(len(distance))
         plt.rcParams['figure.figsize'] = [12,6]

         plt.hist(distance,range=[min(distance)*0.99,max(distance)*1.01],bins=N_classes, edgecol
         plt.xlabel("Valeur de d (mm)")
         plt.ylabel ("Fréquence")
         plt.show()
```



On peut à présent limiter l'étendue de l'histogramme aux valeurs mesurées, calculer la moyenne et l'écart-type, puis comparer à une loi normale.

Il faut pour cela chargée les fonctions **mean** et **std_m** du module skipy.

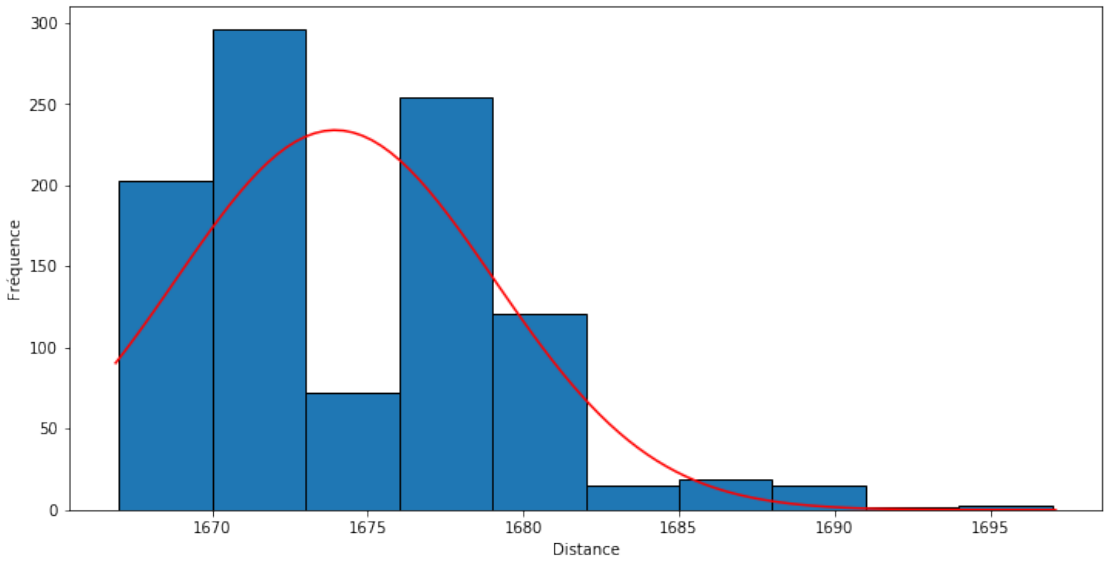
```
In [38]: from scipy import std , mean
         moyenne = mean(distance)
         std_m = std (distance , ddof=1)
         print ( "Moyenne numérique = " , round(moyenne/1000,3) , "m\nÉcart type numérique = " ,
```

```
Moyenne numérique =  1.674 m
Écart type numérique =  0.005 m
```

```
In [42]: import numpy as np
```

```
def gauss(x, A, moyenne, ecarttype):
    return A/(ecarttype*np.sqrt(2*np.pi))*np.exp(-(x-moyenne)**2/(2*ecarttype**2))
                                                    # on définit d'abord la f
N_classes = int(input('Entrer le nombre de clases'))          # on demande le nombre de
A = N * (max(distance)-min(distance))/N_classes                 # on calcule l'air sous
x = np.linspace(int(min(distance)*1000-100)/1000, int((max(distance))*1000+100)/1000,10
y = gauss(x, A, moyenne, std_m)
plt.rcParams['figure.figsize'] = [12,6]                         # définit la taille du gr
plt.hist(distance,range=[min(distance),max(distance)],bins = N_classes, edgecolor='k')
plt.xlabel ("Distance")
plt.ylabel ("Fréquence")
plt.plot(x,y,'r')
plt.show()
```

```
Entrer le nombre de clases 10
```



In [ ]: