CHARLES CARNUS
LYCÉE TECHNIQUE
ENSEIGNEMENT SUPÉRIEUR

Arduino ⟷ Python

Acquisition et traitement de données

Prof : **Kamal Boudjelaba**

5 janvier 2023

B T S S N

LTP Charles Carnus
Avenue de Saint Pierre - Bourran - 12000 RODEZ, France

IR–EC

Table des matières

1. Communication Arduino ⟷ Python

1.1 Demander à la carte Arduino de générer des nombres entiers aléatoires entre 1 et 100

Coder le programme suivant sous Arduino. Ce code envoie une valeur entière aléatoire chaque seconde.

Programme 1 avec Arduino — BTS SN

```
void setup()
  {
    Serial.begin(9600);
    randomSeed(analogRead(0)) ;
  }
void loop()
  {
    Serial.println(random(1, 100));
    delay(1000);
  }
```

1.2 Récupérer les valeurs générées par la carte Arduino dans Python

Code Python : On doit indiquer dans le programme le port série sélectionné (à vérifier dans le menu Arduino Outils ⟶ Port série).

Important

Fermer le moniteur série coté Arduino, pour pouvoir établir une liaison avec Python.

Programme 1.1 avec Python — BTS SN

```
import serial # module gestion du port série
serial_port = serial.Serial(port = "COM1", baudrate = 9600)
serial_port.readline()
#ou essayer la ligne ci-dessous et constater la différence
# serial_port.readline().decode('ascii')
```

Remarque :

Si on exécute le programme Python, on devrait avoir une seule valeur alors que Arduino génère des nombres aléatoires toutes les secondes et on est obligé d'exécuter plusieurs fois la commande `serial_port.readline()`. Si on veut récupérer les nombres aléatoires en continu, on doit modifier le programme Python de cette manière (utiliser une boucle).

Programme 1.2 avec Python — BTS SN

```
import serial
serial_port = serial.Serial(port = "COM1", baudrate = 9600)
for i in range(10):
    print(serial_port.readline())
serial_port.close()
```

1.3 Version améliorée des codes

Si on veut avoir une bonne synchronisation, c'est à dire récupérer les premières valeurs transmises par Arduino à partir de la réinitialisation du microcontrôleur et ce quelque soit la vitesse d'acquisition et de transmission, on peut utiliser le code Python suivant :

Programme 2 avec Arduino — SN Carnus : KB

```
// Version amelioree du programme precedent
int i;
void setup()
{
  Serial.begin(9600);
  randomSeed ( analogRead ( 0 ) ) ;
i = 0;
}
void loop()
{
  Serial.print( i );
  Serial . print ( "\t" ) ;
  Serial.println(random(1 ,100));
  i = i + 1;
  delay(1000);
}
```

Programme 2 avec Python — BTS SN

```
import serial
import time # module de gestion du temps
serial_port=serial.Serial(port = "COM1", baudrate = 9600)
# réinitialisation
serial_port.setDTR(False)
time.sleep(0.1)
serial_port.setDTR(True)
# on vide le buffer
serial_port.flushInput( )
# lecture des données
for i in range(10):
    print(serial_port.readline())
serial_port.close()
```

Une des broches matérielles de contrôle du flux (DTR) du circuit intégré ATmega est connecté à la ligne de réinitialisation de l'ATmega 328 via un condensateur de 100 nF. Lorsque cette broche est mise au niveau LOW, la broche de réinitialisation s'abaisse suffisamment longtemps pour réinitialiser le microcontrôleur. On force la réinitialisation juste avant la lecture des données envoyées par l'Arduino.

Questions :

Maintenant on veut traiter les données reçues avec Python :

1. Récupérer les données utiles
2. Tracer ces données
3. Traiter ces données (moyenne, maximum, minimum, ...)
4. Enregistrer ces données dans un fichier .csv

2. Exercice 1 : Mesure de fréquence

On utilise un stroboscope (application smartphone).

— Régler la fréquence sur 1Hz

— Placer le flash de votre téléphone au dessus de la photorésistance ou de la photodiode

On lit les valeurs envoyées par le capteur (photorésistance ou photodiode) sur une des entrées analogiques de la carte Arduino.

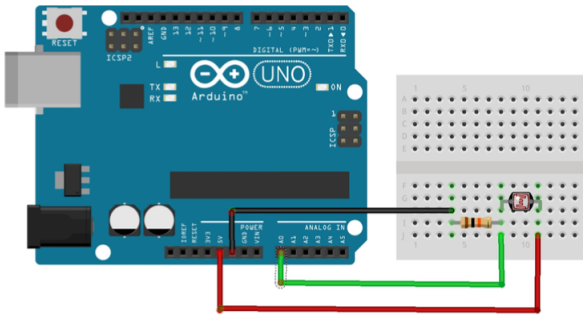


Figure 1. Mesure de fréquence

— Comment faudrait-il modifier le code pour que le nom de variable "valeur" référence bien une tension (attention au type de la variable).

Programme Arduino : BTS SN

```
// Variables à déclarer
void setup(){
    Serial.begin(19200);
}
void loop(){
    // A completer              // Valeur numérique lue sur la broche A0
    Serial.print(valeur);       // Envoi la mesure au PC par la liaison série (port USB)
    Serial.print("\t");         // Ajout d'une tabulation
    Serial.println(millis());   // Envoi de la valeur temps puis retour à la ligne
                                // Une éventuelle temporisation
}
```

2.1 Traitement des données dans Python

On utilise Python pour automatiser la gestion des données envoyées par le capteur.

Programme Python : BTS SN

```
import serial
import time
import matplotlib.pyplot as plt
%matplotlib inline

# connexion Mac (Linux) au port série
serial_port = serial.Serial( port = "/dev/ttyACM1", baudrate =19200)
serial_port.setDTR(False)
time.sleep (0.1)
serial_port.setDTR(True)
serial_port.flushInput()
# les mesures
mesure = []
temps = []
serial_por.flushInput()
for i in range(1000):
  val = serial_port.readline().split()
  try :
    t = float(val[1])
    m= float(val[0])
    temps.append(t)
    mesure.append (m)
  except :
    pass

# fermeture du port série
serial_port.close()

# On évite les effets de bord en éliminant
#les valeurs de début et de fin de transmission
plt.plot(temps[100:900], mesure[100:900])
plt.xlabel("Temps (s)")
plt.ylabel("Intensité")
plt.grid()
plt.show()
```

On peut améliorer la lecture du flux de données afin d'assouplir l'utilisation du code Python. Pour cela on va écrire deux fonctions Python dont on détaillera l'utilisation.

Programme Python : BTS SN

```
def acquisition(n, serial_port ):
  """
  Cette fonction permet de faire l'acqusition des données en fonction du temps reçues par le port USB.
  Elle renvoie deux listes  : temps et mesures (du capteur)
  n <int>                   : nombre total de valeurs à lire
  serial_port <serial>      : le port série ouvert à la communication
  """
  i = 0
  temps, mesures = [], []
  while i < n:
    val = serial_port.readline().split()
    try:
      t = float(val[1])
      m = float(val[0])
      temps.append(t)
      mesure.append(m)
      i=i+1
    except:
      pass
  return temps, mesures
```

Pour lancer une acquisition avec 1 000 points :

Programme Python : BTS SN

```
# connexion Mac (Linux) au port série
serial_port = serial.Serial( port = "/dev/ttyACM1", baudrate =19200)
serial_port .setDTR(False)
time.sleep(0.1) # attention le module time est nécessaire
serial_por.setDTR(True)
serial_port.flushInput()
temps, mesure = acquisition(1000, serial_port)

# fermeture du port série
serial_por.close()
```

2.2 Fixer la durée d'acquisition

Il est souvent plus utile de pouvoir contrôler le temps d'acquisition. Le code Arduino ne change pas et le code Python ne va subir qu'une toute petite modification au niveau de la boucle. Au lieu de compter un nombre de points, nous allons définir un temps d'acquisition. Rappelons que le code Arduino transmet à chaque itération de la fonction loop une ligne contenant une valeur et une date d'acquisition. Pour contrôler le temps d'acquisition il suffit donc de surveiller la différence entre la date en cours d'acquisition et la date du début d'acquisition. Comme les dates d'acquisition sont dans une liste temps, nous allons surveiller `temps[-1]-temps[0]` avec :

`temps[-1]` : le dernier élément de la liste temps et `temps[0]` : le premier élément de la liste.

Programme Python : BTS SN

```
# ouverture du port série
serial_port = serial.Serial( port = "/dev/ttyACM0", baudrate =19200)
serial_port.setDTR(False)
time.sleep (0.1)
serial_port.setDTR(True)
serial_port.flushInput()
# les mesures
mesure = []
temps = []
duree = 10000
end = False
while end == False or temps[-1] - temps[0] <= duree:
    val = serial_port.readline().split ()
    try :
        t = float(val[1])
        m= float(val[0])
        temps.append(t)
        mesure.append (m)
        end = True
    except :
        pass
# fermeture du port série
serial_port.close()
```

– Écrire une fonction `acquisition_temps(duree, serial_port)` qui prend en paramètres la durée d'acquisition et la connexion au port série. Cette fonction renvoie dans l'ordre la liste des dates et les mesures de l'acquisition.
– Afficher les données sous la forme d'un graphique

Programme Python : BTS SN

```
# attention les deux listes doivent contenir le même nombre de valeurs.
plt.plot(temps, mesure)
plt.title("Fréquence d'un stroboscope")
plt.ylabel('Intensité')
plt.xlabel('Temps (ms)')
plt.grid()
plt.show()
```

Solution

Programme Python : BTS SN

```
import serial
import time
import matplotlib.pyplot as plt
%matplotlib inline

def acquisition(n, serial_port ):
  """
  Cette fonction permet de faire l'acquisition des données en fonction du temps reçues par le port USB.
  Elle renvoie deux listes  : temps et mesures (du capteur)
  n <int>                   : nombre total de valeurs à lire
  serial_port <serial>      : le port série ouvert à la communication
  """
  i = 0
  temps, mesures = [], []
  while i < n:
    val = serial_port.readline().split()
    try :
      t = float(val[1])
      m= float(val[0])
      temps.append(t)
      mesure.append(m)
      i = i+1
    except:
      pass
  return temps, mesures
```

Programme Python : BTS SN

```
# connexion Linux au port série
serial_port = serial.Serial( port = "/dev/ttyACM1", baudrate =19200)
serial_port.setDTR(False)
time.sleep(0.1) # attention le module time est nécessaire
serial_port.setDTR(True)
serial_port.flushInput()
temps, mesure = acquisition(1000, serial_port)
# fermeture du port série
serial_port.close()
```

Fixer la durée d'acquisition

Programme Python : BTS SN

```
# ouverture du port série
serial_port = serial.Serial( port = "/dev/ttyACM0", baudrate =19200)
serial_port.setDTR(False)
time.sleep(0.1)
serial_port.setDTR(True)
serial_port.flushInput()
# les mesures
mesure = []
temps = []
duree = 10000
end = False
while end == False or temps[-1] - temps[0] <= duree:
  val = serial_port.readline().split()
  try :
    t = float(val[1])
    m= float(val[0])
    temps.append(t)
    mesure.append(m)
    end = True
  except :
    pass
# fermeture du port série
serial_port.close()

# attention les deux listes doivent contenir le même nombre de valeurs.
plt.plot(temps, mesure)
plt.title("Fréquence d'un stroboscope")
plt.ylabel("Intensité")
plt.xlabel("Temps(ms)")
plt.grid()
plt.show()
```

2.3 Utilisation d'un bouton poussoir pour déclencher l'acquisition

L'objectif est d'ajouter à l'expérience du stroboscope, un bouton poussoir pour déclencher l'acquisition coté Arduino afin que Python puisse enregistrer les données transférées (Exemple, très inspiré d'une activité de Jean-Luc Charles).

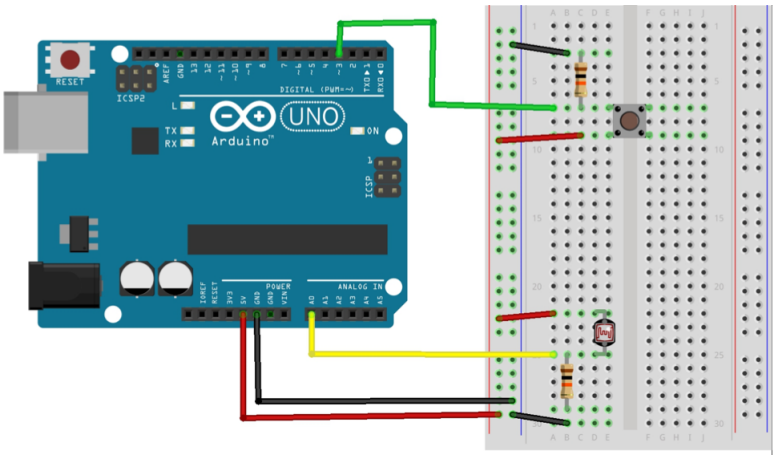


Figure 2. Mesure de fréquence

La broche numérique 3 de la carte Arduino est utilisée comme une entrée numérique qui reste à LOW tant que le bouton n'est pas enfoncé. Le bouton se comporte comme un intérrupteur qui ne laisse pas passer le courant tant qu'il est en position haute.
Dans cet exemple la broche 3 est en mode INPUT : pinMode(3, INPUT) , pour indiquer que la broche est en mode

lecture. Elle ne va donc pas piloter du courant, mais être à l'écoute du courant qui lui arrive.

Programme Arduino : BTS SN

```
// Etat en cours
int etat;
// Etat à mémoriser
int oldEtat;
//Les états possibles
const int WAIT = 2;
const int START = 1;
const int STOP = 0;
// Les broches utilisées
// capteur
const int broche = A0;
//bouton poussoir
const int BP = 3;
void setup ( ) {
    //initialisation des variables
    oldEtat = LOW;
    etat = WAIT;
    //configuration E/S
    pinMode (BP, INPUT) ;
    // liaison série
    Serial.begin(19200);
}
```

Nous avons défini les différents états et initialisé une variable oldEtat qui nous permettra de garder en mémoire l'état du bouton avant un nouvel appui. On remarquera également que l'état WAIT, nous attendons le démarrage de l'acquisition.

Programme Arduino : BTS SN

```
void loop() {
    int etatBP = digitalRead(BP); // Lecture du bouton
    if(oldEtat == LOW && etatBP == HIGH){ // gestion des états
        if (etat == WAIT)
        {
            etat = START;
        }
        else if (etat == STOP)
        {
            etat = START;
        }
        else if (etat == START)
        {
            etat = STOP;
        }
    }
    // Traitement des états
    if( etat == START){
        int valeur = analogRead(broche);
        Serial.print(valeur);
        Serial.print("\t") ;
        Serial.println(millis());
    }
    oldEtat = etatBP;
    delay(10);
}
```

— Lancer l'acquisition en appuyant une première fois sur le bouton
— Stopper l'acquisition en appuyant une deuxième fois sur le bouton.

Modifier le programme pour que lorsque l'acquisition s'arrête, c'est à dire que l'on appuie sur le bouton pour la deuxième fois, la chaîne -1\t -1 s'affiche dans le moniteur série. Attention dans le moniteur série le \t sera remplacé

par une tabulation.

Le code Python ci-dessous fonctionnera correctement uniquement si vous avez répondu à la question précédente.

Programme Python : BTS SN

```
# Les modules à importer
import serial
import matplotlib.pyplot as plt
%matplotlib inline

# Ouverture du port série et synchronisation des données entre Arduino et Python.
serial_port = serial.Serial( port = "COM0", baudrate =19200, timeout = None)
serial_port.flushInput()
# Les mesures
mesure = []
temps = []
end = False

while end == False :
    val = serial_port.readline().split()
    if val[0] == b'-1':  # Bouton poussoir à l'état STOP
        end = True       # Fin de la boucle
    else :
        try :
            t = float(val[1])
            m= float(val[0])
            temps.append(t)
            mesure.append(m)
        except :
            pass

# Fermeture du port série
serial_port.close()
```

Pour tester l'ensemble, assurez-vous que vous avez bien effectué les étapes de la section précédente coté Arduino :

— Normalement sur la gauche de la deuxième cellule vous observez une petite étoile : In[*]

— Positionner votre stroboscope au dessus de la photorésistance

— Lancer l'acquisition des valeurs en appuyant une première fois sur le bouton.

— Terminer l'acquisition en appuyant une deuxième fois sur le bouton, si tout c'est bien passé l'étoile de votre In[*] disparait pour laisser place à un nombre.

— Afficher vos résultats dans un graphique.

Á chaque fois que l'on termine une acquisition il faut revalider la cellule du notebook contenant le code ci-dessus pour mettre en attente le code Python d'une nouvelle acquisition. L'instruction `serial_port.close()` réinitialise le code Arduino et met donc, coté Arduino, dans l'état WAIT. Il n'y a plus qu'à appuyer sur le bouton.

Solution

Bouton poussoir : Arduino BTS SN

```
//Etat en cours
int etat;
//Etat à mémoriser
int oldEtat;
//Les états possibles
const int WAIT = 2;
const int START = 1;
const int STOP = 0;
//Les broches utilisées
//capteur
const int broche = A0;
//bouton poussoir
const int BP = 3;
void setup () {
  //initialisation des variables
  oldEtat = LOW;
  etat = WAIT;
  //config E/S
  pinMode (BP, INPUT);
  //liaison série
  Serial.begin(19200);
}
void loop() {
  //Lecture du bouton
  int etatBP = digitalRead(BP);
  //gestion des états
  if(oldEtat == LOW && etatBP == HIGH)
    {
    if (etat == WAIT)
      {
        etat = START;
      }
    else if (etat == STOP)
      {
        etat = START;
      }
    else if (etat == START)
      {
        etat = STOP;
      }
    }
  //Traitement des états
  if( etat == START)
    {
      int valeur = analogRead(broche);
      Serial.print(valeur );
      Serial.print ("\t");
      Serial.println(millis ());
    }
  else if (etat == STOP)
    Serial.println("-1\t -1");
  oldEtat = etatBP;
  delay(10);
}
```

3. Exercice 2 : Mesurer la vitesse du son

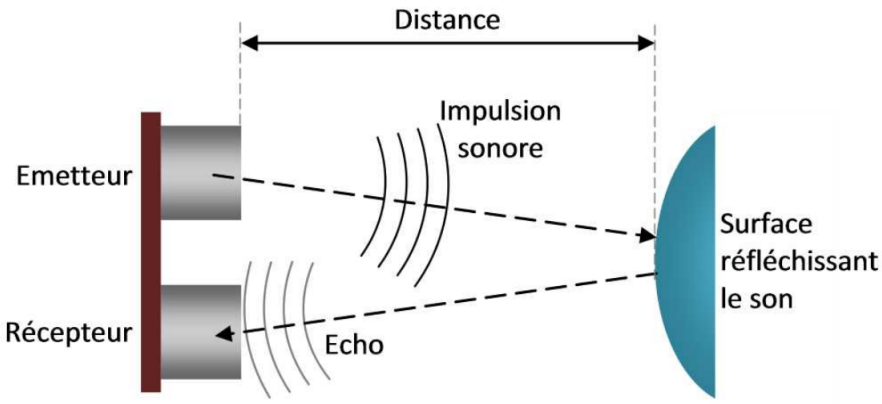


Figure 3. Mesure de la vitesse du son

Les capteurs de distance à ultrasons utilisent le principe de l'écho pour déterminer la distance à laquelle se trouve un objet :

— Un court signal sonore (ultrason à ≈ 40 kHz) est envoyé

— Le son est réfléchi par une surface et repart en direction du capteur

— Le capteur le détecte, une fois revenu à son point de départ.

Les capteurs ultrasonores type HC-SR04 permettent de générer une impulsion ultrasonore puis de mesurer le temps que met le son à revenir d'un obstacle.
Il est donc possible d'utiliser ce capteur pour mesurer la vitesse du son ou des distances.
Ce capteur fonctionne, en lui envoyant une impulsion de 5 V pendant 10 μs sur la broche TRIG.
Une fois qu'il a reçu cette information, il envoie une impulsion ultrasonore et mesure le temps que met cette impulsion à revenir. Ce résultat est retourné par la borne ECHO en micro-secondes.

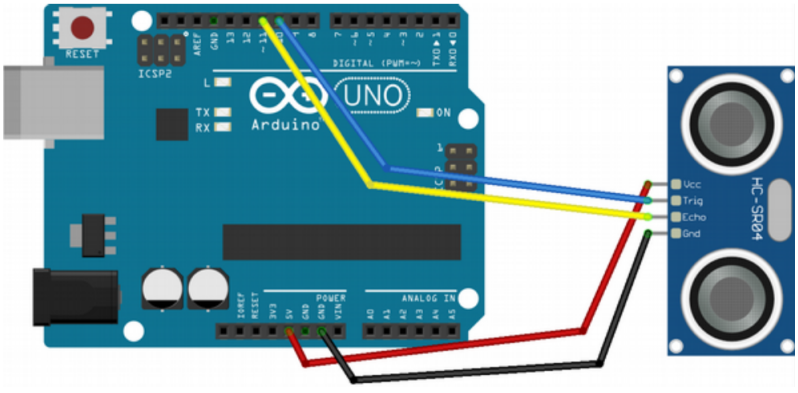


Figure 4. Circuit de mesure de la vitesse du son

3.1 Mesurer la vitesse du son avec Arduino et Python

Le contrôle de l'acquisition est effectué avec une carte Arduino UNO, le traitement sera réalisé avec Python. Les étapes à réaliser sont :

- Arduino
  - Compléter le programme Arduino ci-dessous pour réaliser l'acquisition de la distance et du temps.
- Python

— Récupérer les mesures de temps pour des distances comprises entre 10 cm et 50 cm, avec un pas de 5 cm.
— Tracer le graphique de la distance en fonction du temps.
— Calculer la vitesse moyenne.
— Sauvegarder les données(temps, distance) dans un fichier CSV.

Programme Arduino/Python : à compléter BTS SN

```
// Déclaration des variables globales : broches
//Broche TRIGGER
//Broche ECHO

void setup () {
  pinMode(trig, OUTPUT);          //Configuration des broches
  digitalWrite(trig, LOW);        //La broche TRIGGER doit être à LOW au repos
  pinMode(echo, INPUT);           //La broche ECHO en entrée

  //À compléter                   //Démarrage de la liaison série
}

void loop() {
  digitalWrite(trig, HIGH);       //Lance une mesure de distance en envoyant
  delayMicroseconds(10);          //une impulsion HIGH de 10 microsecondes
  digitalWrite(trig, LOW);

  temps = pulseIn(echo, HIGH);    //Mesure le temps en microseconde entre
                                  //l'envoi de l'ultrason et sa réception

  //À compléter                   //Les résultats sur le port série
                                  //Pause
}
```

Une partie du programme Python pour tester la fonction input et donner des idées sur la manière de gérer le flux de données envoyé par la carte Arduino.

Programme Python : BTS SN

```
mesure = float(input("Entrez votre mesure (distance) : "))
while mesure != -1:
  serial_port.flushInput()
  print(serial_port.readline())
  mesure = float(input("Entrez votre mesure (distance) : "))
```

Solution

Programme Arduino/Python : solution — BTS SN

```
// Déclaration des variables globales : broches
int trig 10;
int echo 11;

void setup () {
  pinMode(trig , OUTPUT);             //Configuration des broches
  digitalWrite(trig , LOW);           //La broche TRIGGER doit être à LOW au repos
  pinMode(echo , INPUT);              //La broche ECHO en entrée
  Serial.begin(9600);                 //Démarrage de la liaison série
}

void loop() {
  digitalWrite(trig , HIGH);          //Lance une mesure de distance en envoyant
  delayMicroseconds(10);              //une impulsion HIGH de 10 microsecondes
  digitalWrite(trig , LOW);           //Fin d'émission
  int temps = pulseIn(echo , HIGH);   //Mesure le temps d'émission-réception

  Serial.print(temps);
  Serial.print("\t");                 //On ajoute une tabulation et
  Serial.println("-1");               //la valeur -1
  delay(500);
}
```

Dans cette solution seule la valeur temps nous intéresse. Mais pour uniformiser le code, on ajoute une tabulation et la valeur 1. Cela permet de réutiliser les codes Python précédents. En effet coté Python, on attend toujours deux informations (deux grandeurs physiques). Il suffit d'ignorer la lecture de la valeur -1 coté Python. En informatique il est courant d'utiliser la valeur -1 pour indiquer soit une erreur soit une valeur à ignorer.

Penser à écrire une fonction pour calculer le temps moyen sur un nombre entier n de valeurs renvoyées par le capteur ultrason.

Programme Python : solution — BTS SN

```
def temps_moyen(n, serial_port ):
  """
  Cette fonction calcule une moyenne des temps ultrasons reçus sur n valeurs
  n           -> <int>    : Nombre de valeurs à lire
  serial_port -> <serial> : Port série
  """
  i = 0
  t_somme = 0
  serial_port.flushInput()
  while i < n:
    val = serial_port.readline().split()
    try:
      t = float(val[0])    # lecture temps ultrason
      t_somme += t         # la somme des temps ultrasons
      i = i+1              # ajoute 1 à la variable comptant les mesures
    except :
      pass
  return t_somme/n
```

Programme Python : solution BTS SN

```python
import serial
import matplotlib.pyplot as plt
%matplotlib inline

# ouverture du port série et synchronisation des données entre Arduino et Python.
serial_port = serial.Serial(port="/dev/ttyACM0", baudrate=9600)

temps = []       # une liste pour les mesures de temps
distances = []   # une liste pour les mesures de distance
dist = 0         # juste pour entrer dans la boucle
nb_t = 10        # le nombre de mesures temps

while dist != -1:
  dist = float(input("Entrez votre mesure (distance) : "))
  if dist != -1:
    distances.append(dist)
    tm = temps_moyen(nb_t, serial_port)
    temps.append(tm)

# fermeture du port série
serial_port.close()

plt.figure(figsize=(12 ,6))
plt.plot(mesures, distances,    ro   )
plt.xlabel("Temps (s)")
plt.ylabel("Distance (m)")
plt.title("Distance de l'obstacle en fonction du temps de réponse du capteur")
plt.grid ()
plt.show()
```

— Téléverser le programme Arduino dans la carte.

— Ouvrir un moniteur série et vérifier que les mesures de temps s'affichent. Faire varier la distance entre l'objet et le capteur HC-SR04 pour observer que les valeurs de temps augmentent si la distance augmente.

— Placer correctement le capteur HC-SR04 à une distance déterminée (voir l'énoncé).

— Exécuter les cellules contenant le code Python concernant cet exercice.

— Normalement, une cellule s'ouvre pour saisir la distance.

4. Exercice 3 : Mesure de température

Afin d'assurer une meilleure surveillance de la température d'un atelier, on utilise un capteur de température (LMT85) et une LED reliés à la carte Arduino.
Réaliser le montage permettant l'acquisition de la température et l'allumage de la LED en cas où la température est élevée ($\geq 24°C$).

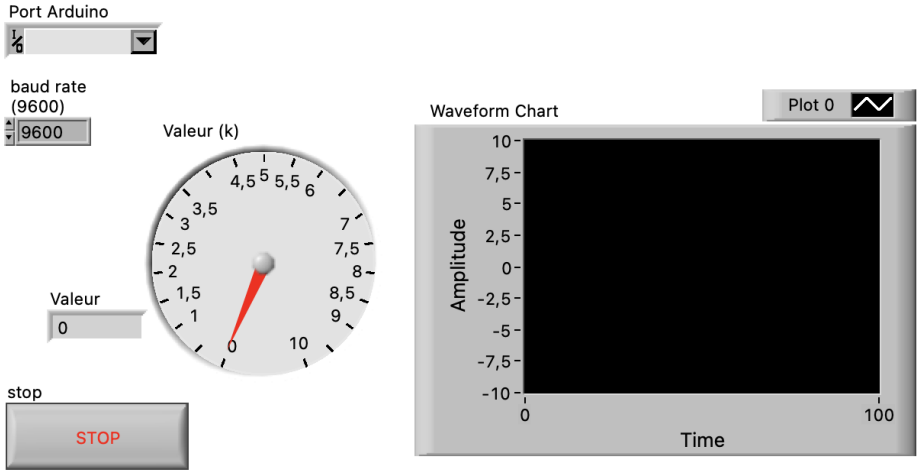
Á partir de l'interface de programmation Arduino, on écrit un programme pour demander à la carte de réaliser les tâches. Ce programme doit être téléverser sur la carte par le port série. Puis on récupère les données via le port série afin de les analyser avec Python.

1. Récupérer les données utiles (Temps et température)
2. Tracer ces données
3. Traiter ces données (moyenne, maximum, minimum, ...)
4. Enregistrer ces données dans un fichier `.csv`

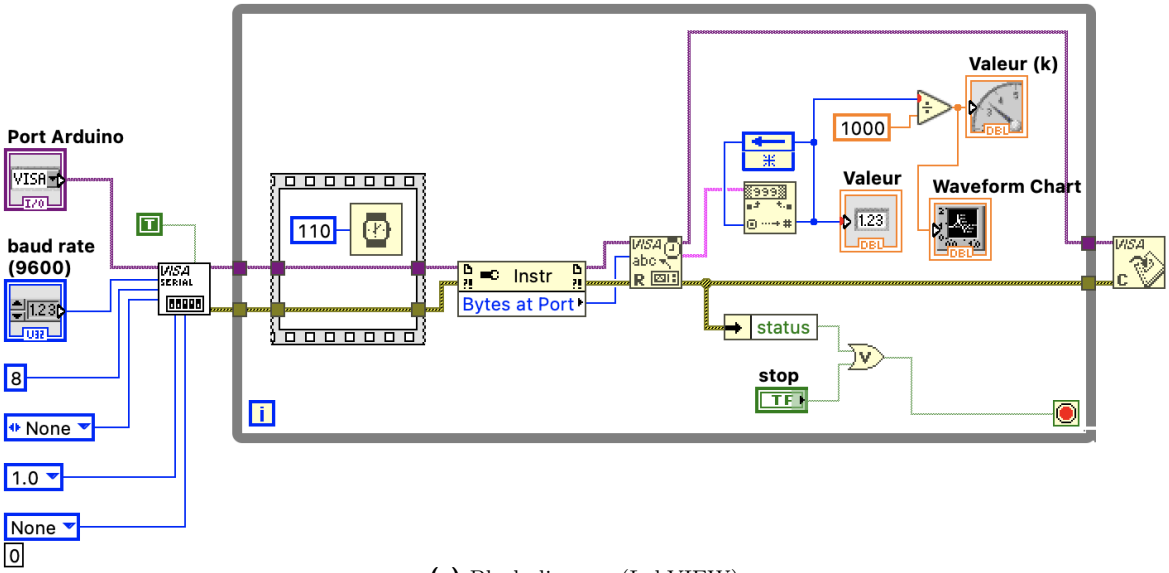
5. Arduino ⟷ LabVIEW

```arduino
// delay(Arduino) < attente(LabVIEW)
int timer = 100;
int y;
void setup() {
    Serial.begin(9600);
    }
void loop() {
    for (int x=0; x<=30; x++) {
    delay(timer);
    Serial.println(y);
    y=y+1000;
    }
  y=0;
}
```

(a) Programme Arduino.



(b) Front panel (LabVIEW).



(c) Block diagram (LabVIEW).

Figure 5. VI LabVIEW.