# Commande d'un Arduino en Python

## 1. Introduction

Ce document montre comment commander un Arduino depuis un script Python tournant sur un ordinateur. La communication entre l'Arduino et l'ordinateur se fait par la liaison série (câble USB). Pour la partie Python, le module pyserial est nécessaire (ce module est inclus dans pythonxy).

On considère dans un premier temps un exemple simple de communication, où des commandes standard de l'Arduino sont envoyées par le script Python.

On verra ensuite comment échanger des tableaux de données entre l'Arduino et Python.

## 2. Exécution de commandes Arduino depuis Python

### 2.a. Principe

Le protocole Firmata permet de commander un Arduino depuis un programme tournant sur un ordinateur (en langage Python, Java, etc). Le programme tournant sur l'arduino reçoit des ordres correspondant aux commandes standard (par exemple `digitalWrite`) de la part du programme de l'ordinateur.

Ce type d'interface est intéressant pour effectuer des tâches simples ne nécessitant pas de réaction en temps réel de la part de l'Arduino. Il peut aussi servir pour vérifier le fonctionnement d'un circuit électronique relié à l'Arduino, avant de mettre en place une solution en temps réel complète.

On propose ici une mise en œuvre de ce principe, avec un protocole élémentaire, qui permet de comprendre comment se fait la communication série. Le programme Arduino exécute sur demande l'une des commandes suivantes :

- ▷ pinMode
- ▷ digitalWrite
- ▷ digitalRead
- ▷ analogWrite
- ▷ analogRead

Les programmes présentés pourront être facilement complétés pour accomplir des tâches plus spécifiques (par exemple des lectures ou écritures séquentielles).

### 2.b. Programme Arduino

Voyons tout d'abord le programme tournant sur l'Arduino, écrit en C.

On commence par définir les codes des différentes commandes, qui doivent être des caractères, c'est-à-dire des entiers 8 bits.

commandesPython.ino

```
#define PIN_MODE 100
#define DIGITAL_WRITE 101
#define DIGITAL_READ 102
#define ANALOG_WRITE 103
```

```
#define ANALOG_READ 104
```

Dans la fonction `setup`, on initialise la liaison série et on envoit la série de caractères (0,255,0) pour informer le correspondant que l'Arduino est prêt.

```
void setup() {
  char c;
  Serial.begin(500000);
  Serial.flush();
  c = 0;
  Serial.write(c);
  c = 255;
  Serial.write(c);
  c = 0;
  Serial.write(c);
}
```

Dans la fonction `loop`, on teste la présence d'une commande envoyée par l'ordinateur (sous forme d'un caractère). Si un caractère de commande est présent, on appelle la fonction de traitement correspondant à la commande. La fonction `loop` pourra bien sûr être complétée par d'autres actions à faire, soit en tâche de fond, soit en réaction à une commande de l'ordinateur.

```
void loop() {
  char commande;
  if (Serial.available()>0) {
     commande = Serial.read();
     if (commande==PIN_MODE) commande_pin_mode();
     else if (commande==DIGITAL_WRITE) commande_digital_write();
     else if (commande==DIGITAL_READ) commande_digital_read();
     else if (commande==ANALOG_WRITE) commande_analog_write();
     else if (commande==ANALOG_READ) commande_analog_read();
  }
  // autres actions à placer ici
}
```

On remarque que la commande `Serial.read()` n'est pas bloquante, c'est-à-dire qu'elle n'attend pas qu'un caractère soit présent sur le port série en lecture pour retourner. C'est pour cette raison qu'il faut utiliser la fonction `Serial.available()`. Ce mode de fonctionnement permet de tester la présence d'un caractère de commande sans bloquer le fonctionnement de l'Arduino.

Voici la fonction de traitement de la commande pinMode. La boucle `while` permet d'attendre que le numéro de la borne et le mode soient envoyés par l'ordinateur, tous deux sous forme de caractère (entier 8 bits).

```
void commande_pin_mode() {
```

```
    char pin,mode;
    while (Serial.available()<2);
    pin = Serial.read(); // pin number
    mode = Serial.read(); // 0 = INPUT, 1 = OUTPUT
    pinMode(pin,mode);
}
```

Voici la fonction de traitement de la commande digitalWrite :

```
void commande_digital_write() {
   char pin,output;
   while (Serial.available()<2);
   pin = Serial.read(); // pin number
   output = Serial.read(); // 0 = LOW, 1 = HIGH
   digitalWrite(pin,output);
}
```

Pour la commande digitalRead, il faut renvoyer à l'ordinateur la valeur lue, sous forme d'un caractère :

```
void commande_digital_read() {
   char pin,input;
   while (Serial.available()<1);
   pin = Serial.read(); // pin number
   input = digitalRead(pin);
   Serial.write(input);
}
```

Voici le traitement de la fonction analogWrite (qui envoit un signal PWM) :

```
void commande_analog_write() {
   char pin,output;
   while (Serial.available()<2);
   pin = Serial.read(); // pin number
   output = Serial.read(); // PWM value between 0 and 255
   analogWrite(pin,output);
}
```

Pour la fonction analogRead, qui utilise le convertisseur analogique-numérique, il faut envoyer à l'ordinateur la valeur lue, sous forme d'un entier 10 bits. En envoit en fait un entier 16 bits sous forme de deux caractères, le premier correspondant aux 8 premiers bits (poids fort), le seconde aux 8 derniers bits.

```
void commande_analog_read() {
   char pin;
```

```
    int value;
    while (Serial.available()<1);
    pin = Serial.read(); // pin number
    value = analogRead(pin);
    Serial.write((value>>8)&0xFF); // 8 bits de poids fort
    Serial.write(value & 0xFF); // 8 bits de poids faible
}
```

### 2.c. Programme Python

Voici le programme Python, qui utilise le module pyserial. Toutes les fonctions sont incluses dans une classe `Arduino`.

commandesPython.py

```
# -*- coding: utf-8 -*-
import serial

class Arduino():
    def __init__(self,port):
        self.ser = serial.Serial(port,baudrate=500000)
        c_recu = self.ser.read(1)
        while ord(c_recu)!=0:
            c_recu = self.ser.read(1)
        c_recu = self.ser.read(1)
        while ord(c_recu)!=255:
            c_recu = self.ser.read(1)
        c_recu = self.ser.read(1)
        while ord(c_recu)!=0:
            c_recu = self.ser.read(1)
        self.PIN_MODE = 100
        self.DIGITAL_WRITE = 101
        self.DIGITAL_READ = 102
        self.ANALOG_WRITE = 103
        self.ANALOG_READ = 104
        self.INPUT = 0
        self.OUTPUT = 1
        self.LOW = 0
        self.HIGH = 1

    def close(self):
        self.ser.close()

    def pinMode(self,pin,mode):
        self.ser.write(chr(self.PIN_MODE))
        self.ser.write(chr(pin))
        self.ser.write(chr(mode))

    def digitalWrite(self,pin,output):
```

```
        self.ser.write(chr(self.DIGITAL_WRITE))
        self.ser.write(chr(pin))
        self.ser.write(chr(output))

    def digitalRead(self,pin):
        self.ser.write(chr(self.DIGITAL_READ))
        self.ser.write(chr(pin))
        x = self.ser.read(1)
        return ord(x)

    def analogWrite(self,pin,output):
        self.ser.write(chr(self.ANALOG_WRITE))
        self.ser.write(chr(pin))
        self.ser.write(chr(output))

    def analogRead(self,pin):
        self.ser.write(chr(self.ANALOG_READ))
        self.ser.write(chr(pin))
        c1 = ord(self.ser.read(1))
        c2 = ord(self.ser.read(1))
        return c1*0x100+c2
```

On remarque que la fonction `Serial.read` est bloquante : elle ne retourne que lorsque le nombre de caractères demandé est présent sur la liaison série. Dans le cas présent, la réponse de l'arduino est immédiate et ce blocage ne pose pas de problème. Si l'on souhaite mettre en place un traitement asynchrone (pour des commandes plus longues), il faudra le faire au moyen de threads. Le traitement asynchrone est nécessaire dans le cas d'un programme avec interface graphique, pour des commandes dont la durée d'exécution excède la seconde. Pour un programme en mode console, cela n'est pas nécessaire.

Le constructeur de la classe nécessite le numéro du port série sur lequel se trouve relié l'Arduino. Celui-ci est facilement accessible dans l'interface de programmation de l'arduino. Si le port est COM5, il faudra fournir le numéro de port 4.

Voici un exemple d'utilisation. Le fichier `arduino.py` doit se trouver dans le même répertoire que le script.

```
import time
from commandesPython import Arduino

port = 4
ard = Arduino(port)

ard.pinMode(2,ard.OUTPUT)
ard.pinMode(3,ard.INPUT)
ard.pinMode(4,ard.OUTPUT)
for i in range(10):
    print(ard.analogRead(0))
    time.sleep(1)
```

```
ard.analogWrite(5,100) # utiliser une sortie PWM pour cela
for i in range(10):
    ard.digitalWrite(2,ard.HIGH)
    time.sleep(1)
    ard.digitalWrite(2,ard.LOW)
    time.sleep(1)
    print(ard.digitalRead(3))
ard.close()
```

## 3. Échanges de tableaux entre Arduino et Python

### 3.a. Programme Arduino

Il peut être nécessaire d'échanger des données sous forme de tableaux d'entiers 8 bits. On propose ci-dessous un prototype pour effectuer ce type d'échanges, avec un tableau tampon de 256 octets. Le même principe peut être utilisé pour transmettre des entiers 16 bits : il suffira de convertir chaque paire d'entiers de 8 bits en un entier 16 bits.

Le programme suivant répond à deux commandes :

- ▷ `SEND_BUFFER` : envoi d'un tableau contenant les entiers 0,1,2, etc.
- ▷ `READ_SEND_BUFFER` : lecture d'un tableau puis renvoi de ce tableau.

transmissionTableaux.ino

```
#define SEND_BUFFER 100
#define READ_SEND_BUFFER 101
#define N 256
uint8_t buffer[N];

void setup() {
  char c;
  Serial.begin(500000);
  Serial.flush();
  Serial.setTimeout(0);
  c = 0;
  Serial.write(c);
  c = 255;
  Serial.write(c);
  c = 0;
  Serial.write(c);
}

void loop() {
  char c1;
  int n;
  int count;
  for (n=0; n<N; n++) buffer[n] = n;
```

```
  if (Serial.available()>0) {
      c1 = Serial.read();
      if (c1==SEND_BUFFER) {
        Serial.write(buffer,N);
     }
     else if (c1==READ_SEND_BUFFER) {
       count = 0;
       while (count < N) {
          n = Serial.readBytes(buffer+count, N-count);
          count = count + n;
        }
        Serial.write(buffer,N);
     }
  }
}
```

La fonction `Serial.setTimeout(0)` permet de rendre la fonction `Serial.readBytes` non bloquante. Celle-ci est appelée tant que le tableau de 256 éléments n'est pas complètement lu.

### 3.b. Programme Python

Voici le programme Python :
transmissionTableaux.py

```
# -*- coding: utf-8 -*-
import serial

class Arduino():
    def __init__(self,port):
        self.ser = serial.Serial(port,baudrate=500000)
        c_recu = self.ser.read(1)
        while ord(c_recu)!=0:
            c_recu = self.ser.read(1)
        c_recu = self.ser.read(1)
        while ord(c_recu)!=255:
            c_recu = self.ser.read(1)
        c_recu = self.ser.read(1)
        while ord(c_recu)!=0:
            c_recu = self.ser.read(1)
        self.SEND_BUFFER = 100
        self.READ_SEND_BUFFER = 101
        self.N = 256 # taille des tableaux

    def close(self):
        self.ser.close()
```

La fonction suivante effectue `n` fois l'appel de la commande `SEND_BUFFER` et calcule le débit de transfert des données.

```
def test_debit_lecture(self,n):
    t = time.clock()
    liste_recus = []
    N = self.N
    for k in range(n):
        self.ser.write(chr(self.SEND_BUFFER))
        liste_recus = self.ser.read(N)
    t = time.clock()-t
    baud = int(n*8*N/t)
    print ("%d bits/s"%baud)
    print ("%d octets/s"%(int(n*N/t)))
    liste = []
    for k in range(len(liste_recus)):
        liste.append(ord(liste_recus[k]))
    print(liste)
```

La fonction suivante effectue `n` fois l'appel de la commande `READ_SEND_BUFFER` et calcule le débit de transfert des données.

```
def test_debit_ecriture_lecture(self,n):
    N = self.N
    liste_emis = []
    for n in range(N):
        liste_emis.append(chr(n))
    liste_recus = []
    t = time.clock()
    for k in range(n):
        self.ser.write(chr(self.READ_SEND_BUFFER))
        self.ser.write(liste_emis)
        liste_recus = self.ser.read(N)
    t = time.clock()-t
    baud = int(n*8*N*2/t)
    print ("%d bits/s"%baud)
    print ("%d octets/s"%(int(n*N*2/t)))
    liste = []
    for k in range(len(liste_recus)):
        liste.append(ord(liste_recus[k]))
    print(liste)
```

Voici un exemple, pour un Arduino connecté sur le port COM5 :

```
from transmissionTableaux import Arduino
port = 4
```

```
ard = Arduino(port)
ard.test_debit_lecture(1000)
ard.close()
```

Pour un arduino Duemilanove (ATmega328), le débit obtenu est d'environ 97000 bits par seconde, soit 12000 octets par seconde. Pour un arduino Mega (ATmega2560), le débit est d'environ 254000 bits par seconde, soit 32000 octets par seconde.

Pour un Arduino Due (SAM3X8E) , avec le port USB natif, il faut utiliser la fonction `SerialUSB`. Le débit obtenu est d'environ 540000 bits par seconde, soit 680000 octets par seconde.

Lorsque la taille du tampon (ici 256) est réduite, le débit est réduit. Pour échanger une grande quantité de données avec un débit maximal, il faut donc utiliser un tampon de manière à échanger les données par bloc. Une taille de tampon de 256 donne de bons résultats.