PYTHON DAQ – SERIAL PORT

1. READING DATA FROM ARDUINO INTO PYTHON

Python has an external library (PySerial) that can establish and utilize a connection made to your computer's serial (COM) ports. This makes it very simple to communicate with external devices (e.g. getting data from an Arduino's input ports). Here we will quickly walk-through the steps…from a DAQ measurement to a Python plot!

TASK 1.1: SEND ANALOG VALUES FROM THE ARDUINO TO THE SERIAL PORT

You may have already done this, but you might need to pay more attention to a few steps as you go through it again. If you haven't seen this already, then please go through the "Microcontroller Lab" worksheet.

EXERCISE 1.1.1: CONNECT YOUR ARDUINO TO YOUR PC

- Under Tools\Board: Select your board type.
- Under Tools\Port: Select the correct port name.

  Note: This is the connection *name* that Python will need to know!

EXERCISE 1.1.2: TELL ARDUINO TO SEND DATA TO THE SERIAL PORT

- The *Basic* example (*AnalogReadSerial)* does this. The code below (from the Microcontroller Lab) is to remind you what is going on.

```
void setup() {
  // initialize serial communication at 9600 bits per second
  // …note: to go faster you can set 57600 bits per second
  Serial.begin(9600);
}

void loop() {
  int value = 42;
  Serial.println(value);
  delay(1);         // delay (1 ms) in between writes for stability
}
```

- Compile and run either the example or the above code on your Arduino.

TASK 1.2: READ FROM THE SERIAL PORT WITH PYTHON

Python can also access the serial port. Using the library PySerial, which is installed by default by most scientific Python distributions, you can send and receive data to/from any device connected to a "true" (RS-232) serial port (or via a USB-Serial Port converter, as in the case of the Arduino).

EXERCISE 1.2.1: CHECK THAT PYSERIAL IS INSTALLED

- Use the following code to check which Packages are installed with your Python distribution. Make sure the "pyserial" is in this list! If not, install it. (**Anaconda Instructions**: http://docs.continuum.io/anaconda/faq.html#install-packages)

```
import pip
installed_packages = pip.get_installed_distributions()
```

EXERCISE 1.2.2: USE PYSERIAL TO MONITOR THE SERIAL PORT CONNECTED TO THE ARDUINO

- Use the following Python code to continually monitor every "line" of ASCII encoded text arriving at the named serial port.

Advanced Exercise 1.2.2: Can you make the program print "Hit!" if the read value exceeds some value?

```
import serial
ser = serial.Serial('COM3', 9600)    # or 57600 bps
while True:
    print ser.readline()
```

The data arriving is encoded as "text/ASCII" and must be decoded into real values/numbers before it can be processed and analyzed as such. The procedure of converting strings of characters into their denoted values is called "parsing". In this case, where just one number is read at a time versus a line of text with many different numbers and types of data in one long line, a simple command is possible, i.e. int('string'). Other scenarios, such as reading a spreadsheet, will require more sophisticated parsing, e.g. np.genfromtxt().

EXERCISE 1.2.3: ACQURE, PARSE AND DISPLAY A FINITE AMOUNT OF DATA

- Read 1000 samples and plot them in a graph.

```
import serial
import numpy as np
import matplotlib.pyplot as plt

ser = serial.Serial('COM3', 9600)  #  …or 57600 bps
data = np.zeros(1000)              #  Initialize an array with zeros
for i in range(0,1000):
    line = ser.readline()
    value =  int(line)             #  Convert to integer
    data[i] =  value               #  Add new value to the array

plt.plot(data)                     #  Plot
# FIN
```

You may also want to run an experiment/measurement "continuously" and save the data for subsequent analysis. It is possible to "stream" each data sample to a *file* (files are stored on external memory device, e.g. a hard disk drive). In this case, you will first create/open a file "handle", which you can then use to write data to the file when it becomes available. Remember to *close* the file handle when finished!

How do you tell Python when you are finished? There are many ways to do this…in the following example we will execute a *while* loop until the default *KeyboardInterrupt* occurs (CTRL-C).

EXERCISE 1.2.4: SAVE DATA CONTINUOUSLY TO A TEXT FILE (.CSV)

```
import serial

ser = serial.Serial('COM3', 9600)   #  …or 57600 bps

filename = r'C:\somewhere\data.csv' # Specify filename (location on disk)
file_handle = open(filename, "w")   # Open a file "handle" for "w"riting

try: # Attempt the next bit of code...
    while True: # Go "forever"
         line = ser.readline()
         file_handle.write(new_ line) # Write the new text line to the open file

except KeyboardInterrupt: # ...unless the default interrupt occurs (Ctrl-C)
    pass

file_handle.close()       # Close the open file handle
ser.close()
# FIN
```