9

Data Acquisition with Python and Tkinter

9.1 Basics

The open() method: This function is a most commonly used method that is available in Python, and it is used to manipulate files.

To open a file command:

>>> f = open('test.txt', 'w')

This command creates a test.txt file in the same folder in which the Python interpreter is saved or the location from where the code is being executed.

The modes that can be used with the open() function are discussed in Table 9.1.

The write() method: Once the file is open in the writing mode, the write() method is used to start writing to the file object. The write() method takes input argument only in the format of a string.

>>>F.write("Hello World!\n")

Here "Hello World" is a string, and \n means a new line character.

To write a sequence of strings, the writelines() method is used:

>>> sq = ["Python programming for Arduino\n", "Bye\n"]

>>>W.writelines(sq)

The close() method: The close() method is used to close the file, and the file object can't be used again. The command is:

>>>W.close()

The read() method: This read() method reads the data of a file. To use this method, open the file with any of the reading compatible modes like: w+, r, r+, or a+:

>>>D = open('test.txt', 'r')

>>> D.read()

TABLE 9.1

Description of Modes

Mode	Description
W	This mode opens a file only to write. It overwrites an existing file.
w+	This mode opens a file to write and read both. It overwrites an existing file.
R	This mode opens a file only to read.
r+	This mode opens a file to write and read both.
A	This mode opens a file for appending, starting from end of the document.
a+	This mode opens a file for appending and reading, starting from end of the document.

```
'Hello World!\nPython programming for Arduino\nBye\n'
>>>D.close()
```

With this method, the entire contents of the file is stored into memory. To read the content line to line, use the readlines() method:

```
>>>D= open('test.txt', 'r')
>>>X =D.readlines()
>>> print X
['Hello World!\n', 'Python programming for Arduino\n', 'Bye\n']
>>>D.close()
```

9.2 CSV File

CSV files are used to store the data in Python. A CSV writer is used to write data on a CSV file and the reader to read it with simple commands:

CSV writer

```
import csv
data = [[1, 2, 3], ['a', 'b', 'c'], ['Python', 'Arduino', 'Programming']]
with open('example.csv', 'w') as f:
    w = csv.writer(f)
    for row in data:
        w.writerow(row)
```

CSV reader

```
import csv
with open('example.csv', 'r') as file:
    r = csv.reader(file)
    for row in r:
        print row
```

9.3 Storing Arduino Data with CSV File

The CSV files can be used to store the sensory data from Arduino. To understand the concept, a system is discussed. The system is comprised of Raspberry Pi, Arduino, a pyroelectric infrared (PIR) sensor (digital sensor), a potentiometer (POT) (as an analog sensor), and a power supply. The objective is to store the sensory data with CSV files and Arduino interfacing with Python. Figure 9.1 shows the circuit diagram of the system.

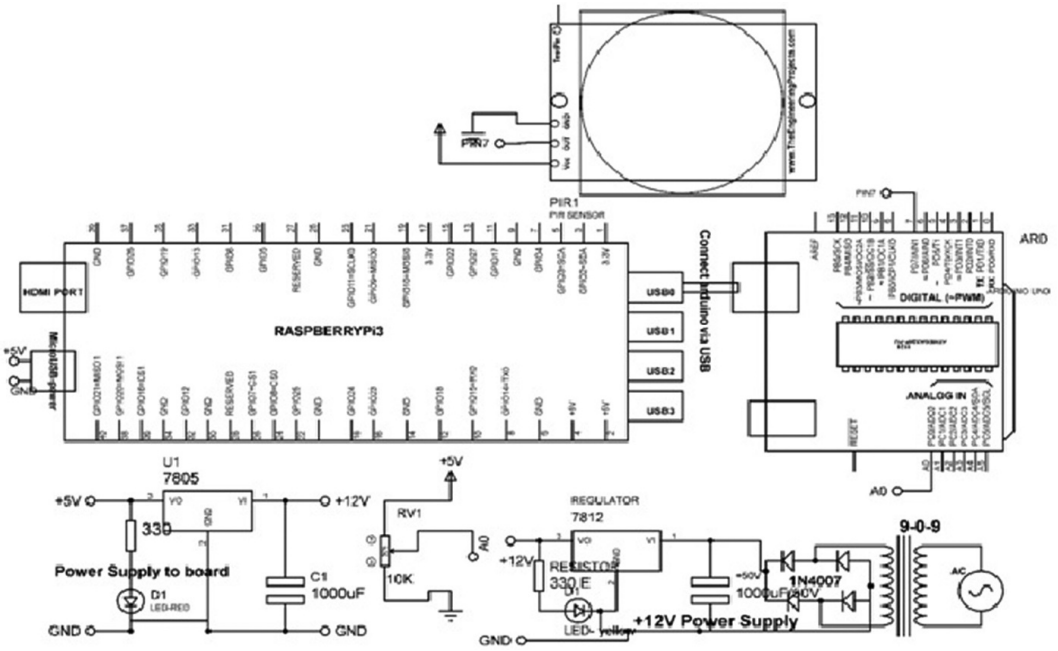

FIGURE 9.1
Circuit diagram to interface the PIR sensor and the POT with Arduino UNO and Pi.

Connections:

- Connect one terminal of POT to +5 V, the other terminal to ground, and wiper to pin (A0) of Arduino Uno.
- Connect pins (Vcc and ground) of the PIR sensor to +5 VDC and ground, respectively.
- Connect pin (OUT) of the PIR sensor to pin(7) of Arduino Uno.
- Connect Arduino Uni to Raspberry Pi through a USB.

9.3.1 Recipe

```
import csv # import CSV library
  import pyfirmata # import pyfirmata library
  import time as wait # import time library
  board = pyfirmata.Arduino('/dev/ttyUSB0')
  it = pyfirmata.util.Iterator(board)
  it.start() # start iterator
  PIR_pin = board.get_pin('d:7:i') # connect PIR sensor to pin 7 as input
  POT_pin = board.get_pin('a:0:i') # connect POT to pin 0 as input
  with open('SensorDataStore.csv', 'w') as f:
    w = csv.writer(f)
    w. writerow(["Number", "Potentiometer", "Motion sensor"])
    i = 0
    PIR_Data = PIR_pin.read() # read PIR sensor
    POT_Data = POT_pin.read() # read POT
    while i < 25:
      sleep(1)
      if PIR_Data is not None:
        i += 1
        row = [i, POT_Data, PIR_Data]
        w. writerow(row)
    print "process complete. CSV file is ready!"
  board.exit()
```

9.4 Plotting Random Numbers Using Matplotlib

The installation of matplotlib is an easy process on Ubuntu by a simple command:

$ sudo apt-get install python-matplotlib

Click on "yes" when prompted to install dependencies. The matplotlib library provides the plot() method to create line charts. The plot() method takes a list or an array data structure made up of integer or floating point numbers as input. Plot() uses values for the *x*-axis and *y*-axis, if two arrays are given as inputs. If only one list or array is provided as input, plot() assumes the values for the *y*-axis and autogenerates the incremental values for the *x*-axis:

pyplot.plot(x, y)

To change the style of line and makers with different colors, the plot() method can be used, e.g., for the solid line style command:

pyplot.plot(x, y, '-')

Figure 9.2 shows the plotting of the random number.

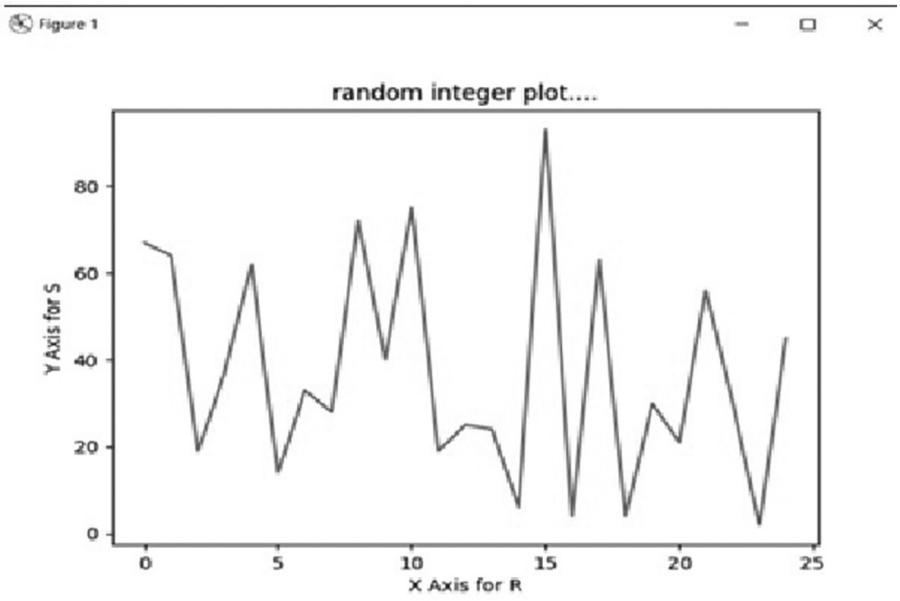

FIGURE 9.2
Plotting the random number.

9.4.1 Recipe

For generating and plotting random numbers:

```
import random
R = range(0,25)
S = [random.randint(0,100) for r in range(0,25)]
FIGURE1 = pyplot.figure()
pyplot.plot(R, S, '-')
pyplot.title('random integer plot....')
pyplot.xlabel('X Axis for R')
pyplot.ylabel('Y Axis for S')
pyplot.show()
```

9.5 Plotting Real-Time from Arduino

The real-time data plotting from Arduino is an important task where sensory data is critical. To understand the concept, real-time POT values are discussed in this section. The system is comprised of Raspberry Pi, Arduino, a POT, and a power supply. Arduino is connected to Raspberry Pi through a USB. One terminal of the POT is connected to +5 V, the other to the ground, and the wiper is connected to pin (A0) of Arduino (Figure 9.3). To plot the real-time data, move the knob of the POT and check the results.

The real-time plotting can be achieved by using a combination of the pyplot functions ion(), draw(), set_xdata(), and set_data(). The ion() method is used to initialize the interactive mode of pyplot. This helps to dynamically change the *x* and *y* values of the plots in the figure:

```
pyplot.ion()
```

Once the interactive mode is set, the plot will be drawn by calling the draw() method. Now, initialize the plot with a set of blank data, 0, in this case:

```
pData = [0] * 25
```

In this array for *y* values, *pData*, is used to append values from the sensor in the while loop to keep appending the newest values to this data array and redraws the plot with these updated arrays for the *x* and *y* values.

```
pData.append(float(a0.read()))
del pData[0]
```

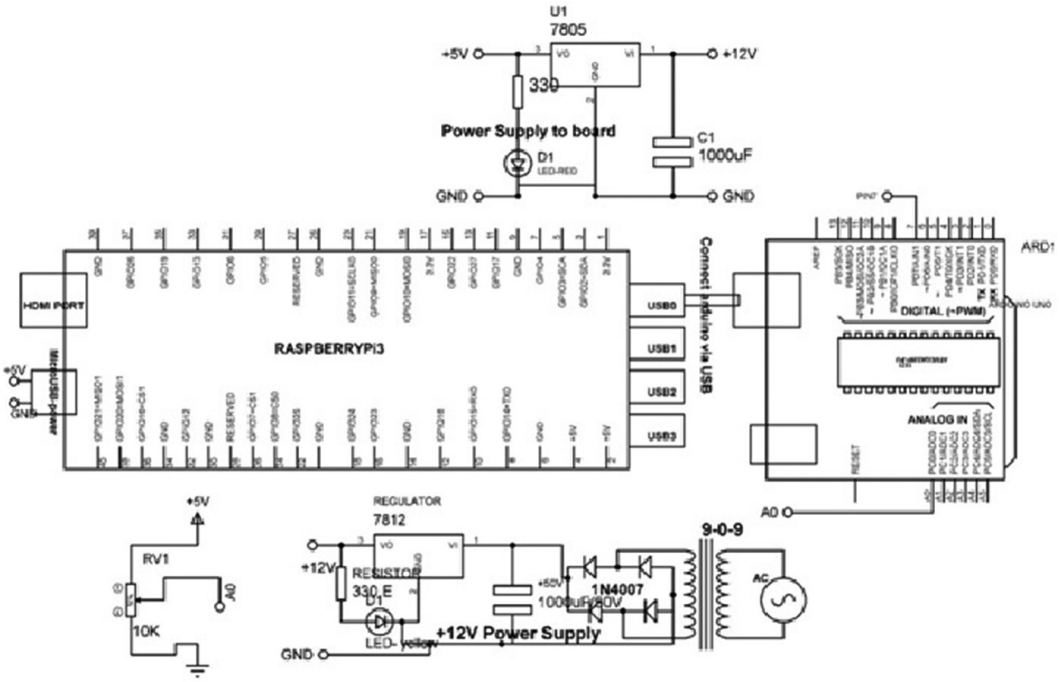

FIGURE 9.3
Circuit diagram for the interfacing of POT with Arduino.

The set_xdata() and set_ydata() methods are used to update the *x* and *y* axes data.

```
l1.set_xdata([i for i in xrange(25)])
l1.set_ydata(pData) # update the data
pyplot.draw() # update the plot
```

The code snippet, [i for i in xrange(25)], is to generate a list of 25 integer numbers that will start incrementally at 0 and end at 24.

9.5.1 Recipe

```
//from matplotlib import pyplot
  import pyfirmata # import pyfirmata library
  import time as wait
  board = pyfirmata.Arduino('/dev/ttyUSB0')
  wait.sleep(5) # wait for 5 Sec
  it = pyfirmata.util.Iterator(board)
  it.start() # start iterator
  POT_pin = board.get_pin('a:0:i') # connect POT to A0 pin as input
```

```
pyplot.ion()
pData = [0.0] * 25
fig = pyplot.figure()
pyplot.title('Real time data plot from POT')
ax1 = pyplot.axes()
l1, = pyplot.plot(pData)
pyplot.ylim([0, 1])

while True:
  try:
wait.sleep(1)
      pData.append(float(POT_pin.read()))
      pyplot.ylim([0, 1])
      del pData[0]
      l1.set_xdata([i for i in xrange(25)])
      l1.set_ydata(pData) # update the data
      pyplot.draw() # update the plot
  except KeyboardInterrupt:
      board.exit()
      break
```

9.6 Integrating the Plots in the Tkinter Window

Section 9.5 describes how to draw a plot for continuous sensory data from Arduino with the help of a POT. Python has a powerful integration capability with the matplotlib library and Tkinter graphical interface. For the same circuit as Figure 9.3, this integration is discussed. The program uses the interfacing of Tkinter with matplotlib.

9.6.1 Recipe

```
import sys # import sys library
from matplotlib import pyplot # import library
import pyfirmata # import library
import time as wait # import time library
import Tkinter # import library
```

```
    def start_button_press():
      while True:
          if FLAG.get():
  wait.sleep(1) # wait for 1 Sec
            pData.append(float(POT_pin.read()))
            pyplot.ylim([0, 1])
            del pData[0]
            l1.set_xdata([i for i in xrange(25)])
            l1.set_ydata(pData) # update the data
            pyplot.draw() # update the plot
            TOP.update()
          else:
            FLAG.set(True)
            break
def pause_button_press():
  FLAG.set(False)
def exit_button_press():
  print "out from data recording...."
pause_button_press()
   board.exit()
   pyplot.close(fig)
   TOP.quit()
   TOP.destroy()
   print "completed....."
   sys.exit()
board = pyfirmata.Arduino('/dev/ttyUSB0')
# Using iterator thread to avoid buffer overflow
it = pyfirmata.util.Iterator(board)
it.start() # start iterator
# Assign a role and variable to analog pin 0
POT_pin = board.get_pin('a:0:i') # read POT
# Tkinter canvas
TOP = Tkinter.Tk()
TOP.title("Tkinter + matplotlib")
# Create flag to work with indefinite while loop
FLAG = Tkinter.BooleanVar(TOP)
```

```
FLAG.set(True)
pyplot.ion()
pData = [0.0] * 25
figure = pyplot.figure()
pyplot.title('Potentiometer')
ax1 = pyplot.axes()
l1, = pyplot.plot(pData)
pyplot.ylim([0, 1])
# Create Start button and associate with start button press method
start_Button = Tkinter.Button(TOP, text="Start", command=start_
    button_press)
start_Button.grid(column=1, row=2)
# Create Stop button and associate with pause button press method
pause_Button = Tkinter.Button(TOP, text="Pause", command=pause_
    button_press)
# Create Exit button to exit from window
exit_Button=Tkinter.Button(TOP,text="Exit",command=exit_button_press)
exit_Button.grid(column=3, row=2)
TOP.mainloop()
```

Execute the program, and a window will appear on the screen (Figure 9.4). The plot can be controlled by using "Start", "Pause" and "Exit" buttons. Click on start button and rotate the knob of POT and see the changes in plot. The process can be paused or close the program with "Exit" button.

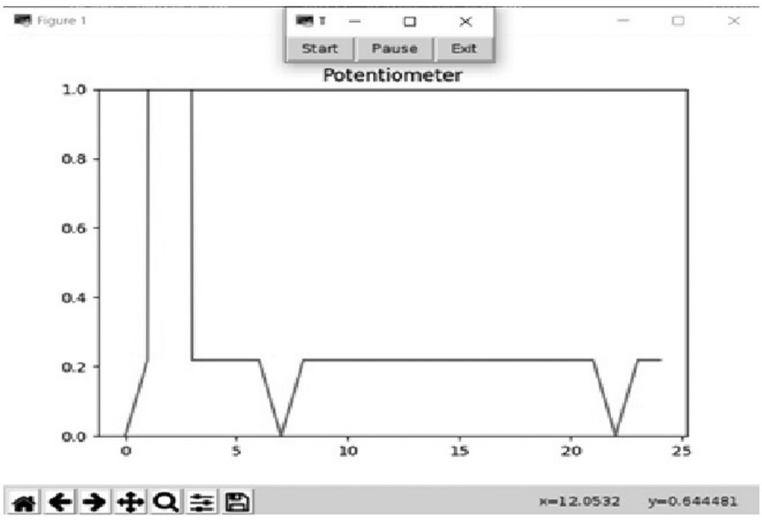

FIGURE 9.4
Plot for real-time data from Arduino.