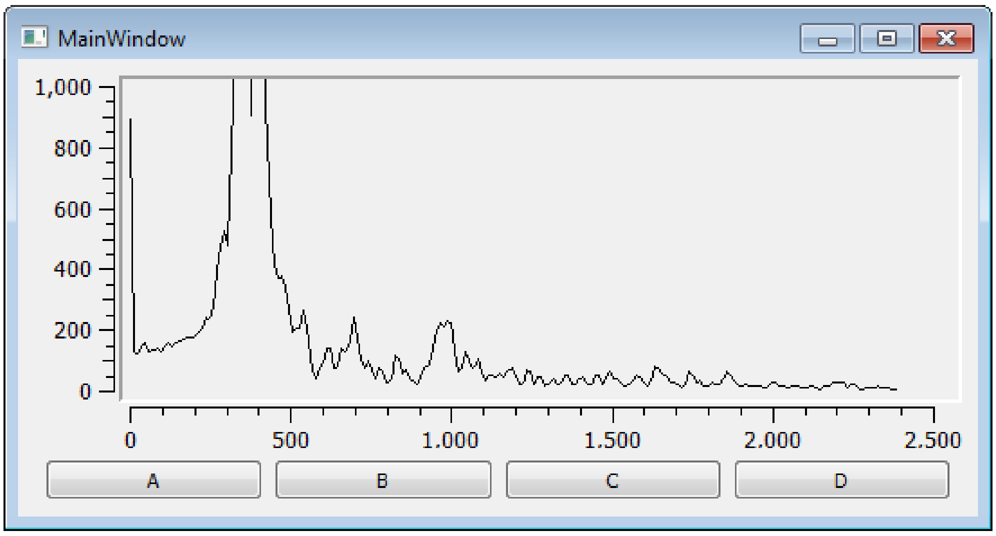
Realtime FFT Audio Visualization with Python

Categories: General, Python, RF (Radio Frequency)

12 comments

May 9, 2013

I'm no stranger to visualizing linear data in the frequency-domain. Between the high definition spectrograph suite I wrote in my first year of dental school (QRSS-VD, which differentiates tones to sub-Hz resolution), to the various scripts over the years (which go into FFT imaginary number theory, linear data signal filtering with python, and real time audio graphing with wckgraph), I've tried dozens of combinations of techniques to capture data, analyze it, and display it with Python. Because I'm now branching into making microcontroller devices which measure and transfer analog data to a computer, I need a way to rapidly visualize data obtained in Python. Since my microcontroller device isn't up and running yet, linear data from a PC microphone will have to do. Here's a quick and dirty start-to-finish project anyone can tease apart to figure out how to do some of these not-so-intuitive processes in Python. To my knowledge, this is a cross-platform solution too. For the sound card interaction, it relies on the cross-platform sound card interface library PyAudio. My python distro is 2.7 (python xy), but pythonxy doesn't [yet] supply PyAudio.



The code behind it is a little jumbled, but it works. For recording, I wrote a class "SwhRecorder" which uses threading to continuously record audio and save it as a numpy array. When the class is loaded and started, your GUI can wait until it sees *newAudio* become *True*, then it can grab *audio* directly, or use fft() to pull the spectral component (which is what I do in the video). Note that my fft() relies on numpy.fft.fft(). The return is a nearly-symmetrical mirror image of the frequency components, which (get ready to cringe mathematicians) I simply split into two arrays, reverse one of them, and add together. To turn this absolute value into dB, I'd take the log10(fft) and multiply it by 20. You know, if you're into that kind of thing, you should really check out a post I made about FFT theory and analyzing audio data in python.

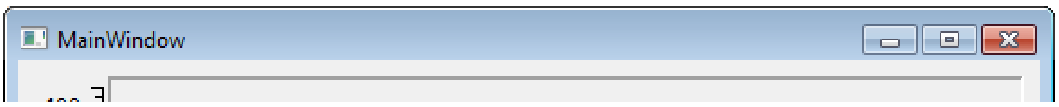
Here's the meat of the code. To run it, you should really grab the zip file at the bottom of the page. I'll start with the recorder class:

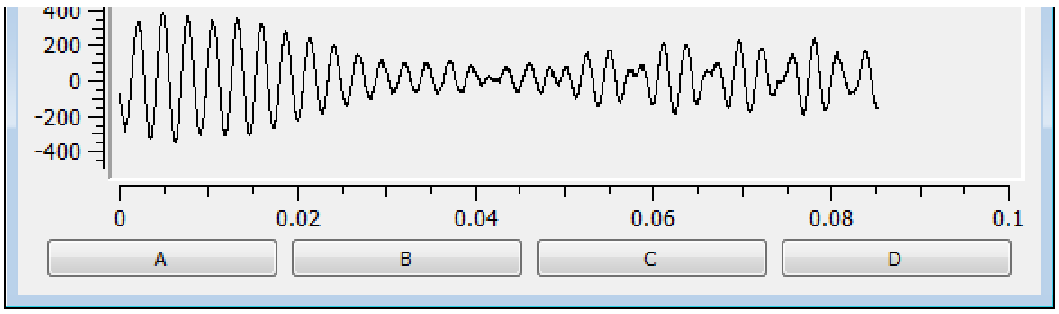
```
import matplotlib
matplotlib.use('TkAgg') # THIS MAKES IT FAST!
import numpy
import scipy
import struct
import pyaudio
import threading
import pylab
import struct

class SwhRecorder:
    """Simple, cross-platform class to record from the microphone."""

    def __init__(self):
        """minimal garb is executed when class is loaded."""
        self.RATE=48100
        self.BUFFERSIZE=2**12 #1024 is a good buffer size
        self.secToRecord=.1
        self.threadsDieNow=False
        self.newAudio=False

    def setup(self):
        """initialize sound card."""
        #TODO - windows detection vs. alsa or something for linux
        #TODO - try/except for sound card selection/initiation

        self.buffersToRecord=int(self.RATE*self.secToRecord/self.BUFFERSIZE)
        if self.buffersToRecord==0: self.buffersToRecord=1
        self.samplesToRecord=int(self.BUFFERSIZE*self.buffersToRecord)
        self.chunksToRecord=int(self.samplesToRecord/self.BUFFERSIZE)
        self.secPerPoint=1.0/self.RATE

        self.p = pyaudio.PyAudio()
        self.inStream = self.p.open(format=pyaudio.paInt16,channels=1,
            rate=self.RATE,input=True,frames_per_buffer=self.BUFFERSIZE)
        self.xsBuffer=numpy.arange(self.BUFFERSIZE)*self.secPerPoint
        self.xs=numpy.arange(self.chunksToRecord*self.BUFFERSIZE)*self.secPerPoint
        self.audio=numpy.empty((self.chunksToRecord*self.BUFFERSIZE),dtype=numpy.int16)

    def close(self):
        """cleanly back out and release sound card."""
        self.p.close(self.inStream)

    ### RECORDING AUDIO ###

    def getAudio(self):
        """get a single buffer size worth of audio."""
        audioString=self.inStream.read(self.BUFFERSIZE)
        return numpy.fromstring(audioString,dtype=numpy.int16)

    def record(self,forever=True):
        """record secToRecord seconds of audio."""
        while True:
            if self.threadsDieNow: break
            for i in range(self.chunksToRecord):
                self.audio[i*self.BUFFERSIZE:(i+1)*self.BUFFERSIZE]=self.getAudio()
            self.newAudio=True
            if forever==False: break

    def continuousStart(self):
        """CALL THIS to start running forever."""
        self.t = threading.Thread(target=self.record)
        self.t.start()

    def continuousEnd(self):
        """shut down continuous recording."""
        self.threadsDieNow=True

    ### MATH ###

    def downsample(self,data,mult):
        """Given 1D data, return the binned average."""
        overhang=len(data)%mult
        if overhang: data=data[:-overhang]
        data=numpy.reshape(data,(len(data)/mult,mult))
        data=numpy.average(data,1)
        return data
    def fft(self,data=None,trimBy=10,logScale=False,divBy=100):
        if data==None:
            data=self.audio.flatten()
        left,right=numpy.split(numpy.abs(numpy.fft.fft(data)),2)
        ys=numpy.add(left,right[::-1])
        if logScale:
            ys=numpy.multiply(20,numpy.log10(ys))
        xs=numpy.arange(self.BUFFERSIZE/2,dtype=float)
        if trimBy:
            i=int((self.BUFFERSIZE/2)/trimBy)
            ys=ys[:i]
            xs=xs[:i]*self.RATE/self.BUFFERSIZE
        if divBy:
            ys=ys/float(divBy)
        return xs,ys

    ### VISUALIZATION ###

    def plotAudio(self):
        """open a matplotlib popup window showing audio data."""
        pylab.plot(self.audio.flatten())
        pylab.show()
```

And now here’s the GUI launcher:

```
import ui_plot
import sys
import numpy
from PyQt4 import QtCore, QtGui
import PyQt4.Qwt5 as Qwt
from recorder import *

def plotSomething():
    if SR.newAudio==False:
        return
    xs,ys=SR.fft()
    c.setData(xs,ys)
    uiplot.qwtPlot.replot()
    SR.newAudio=False

if __name__ == "__main__":
    app = QtGui.QApplication(sys.argv)

    win_plot = ui_plot.QtGui.QMainWindow()
    uiplot = ui_plot.Ui_win_plot()
    uiplot.setupUi(win_plot)
    uiplot.btnA.clicked.connect(plotSomething)
    #uiplot.btnB.clicked.connect(lambda: uiplot.timer.setInterval(100.0))
    #uiplot.btnC.clicked.connect(lambda: uiplot.timer.setInterval(10.0))
    #uiplot.btnD.clicked.connect(lambda: uiplot.timer.setInterval(1.0))
    c=Qwt.QwtPlotCurve()
    c.attach(uiplot.qwtPlot)

    uiplot.qwtPlot.setAxisScale(uiplot.qwtPlot.yLeft, 0, 1000)

    uiplot.timer = QtCore.QTimer()
    uiplot.timer.start(1.0)

    win_plot.connect(uiplot.timer, QtCore.SIGNAL('timeout()'), plotSomething)

    SR=SwhRecorder()
    SR.setup()
    SR.continuousStart()

    ### DISPLAY WINDOWS
    win_plot.show()
    code=app.exec_()
    SR.close()
    sys.exit(code)
```

Note that by commenting-out the FFT line and using “c.setData(SR.xs,SR.audio)” you can plot linear PCM data to visualize sound waves like this:





Finally, here’s the zip file. It contains everything you need to run the program on your own computer (including the UI scripts which are not written on this page)

DOWNLOAD: **SWHRecorder.zip**

If you make a cool project based on this one, I’d love to hear about it. Good luck!