Circuits vs. Software

January 15, 2009 by Scott | Circuitry, DIY ECG, General, Python | Leave a comment

> ***UPDATE:*** *An improved ECG design was posted in August, 2016.*
> *Check out: https://www.swharden.com/wp/2016-08-08-diy-ecg-with-1-op-amp/*

Would I rather design circuits or software? I'm a software guy (likely due to my lack of working knowledge of circuits) so I'd rather record noisy signals and write software to eliminate the noise, rather than assembling circuits to eliminate the noise for me. In the case of my DIY ECG machine, I'd say I've done a great job of eliminating noise via the software route. Most DIY ECG circuits on the net use multiple op-amps and diodes to do this, and have a hardware-based band-pass filter to eliminate frequencies around 60 Hz. Instead of all that fancy stuff, I made a super-crude circuit (a single op-amp and two resisters) to record my ECG. It was INCREDIBLY noisy! So, how did I clean it up with software? I'll tell you.

The first step in removing electrical noise is classifying it. Most of the noise in my signal were overlapping sine waves caused by my electrodes picking up signals not from my body. This was determined by simply close-up observation of the original trace. Since this sine-type interference is consistant, power-spectral analysis could be applied to determine the frequencies of the noise so I could block them out. I used the fast Fourier transform algorithm (FFT) on the values from my wave file to generate a plot of noise level with respect to frequency (noise was seen as sharp peaks). I manually blocked-out certain regions of the FFT trace that I thought were noise-related (colored bands, all the values of

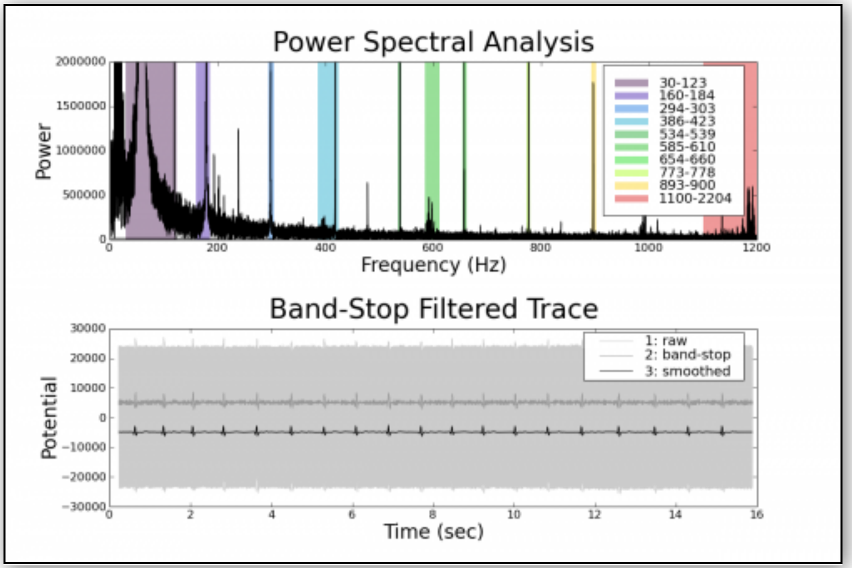
Subscribe via Email

Enter your email address to subscribe to this blog and receive notifications of new posts by email.

Email Address

Subscribe

which were set to zero) – a process known as band-pass filtering (something which is possible to do electronically with a more complicated circuit) – then performed an inverse FFT algorithm on the trace. The result was a trace with greatly reduced noise and after a moving window smoothing algorithm was applied the signal was better than ever. Below are some figures. Note that I recorded the raw WAV file with "sound recorder" (not GoldWave) and did all of the processing (including band-pass filtering) within Python.



What an awesome chart! On top we have the power spectral analysis with band-stop filters applied at the colored regions. Below is the trace of the original WAV file (light gray), the inverse-FFT-filtered trace (dark gray), and the smoothed inverse-FFT-filtered trace (black) – the final ECG signal I intend to use.

This is a magnified view of a few heartbeats after the inverse FFT filtering (multi-band-blocking) process was applied. Not bad! Not bad at all...

And, of course, the updated code:

Categories

C/C++ Circuitry
DIY ECG Electronics
General GitHub
HAB (high altitude balloon)
Linux
Microcontrollers
Molecular Biology My
Website PHP Prime
Numbers Programming
Python QRSS / MEPT

```
import wave, struct, numpy, pylab, scipy

fname='./success3.wav'

def readwave(wavfilename):
    """load raw data directly from a WAV file."""
    global rate
    w=wave.open(wavfilename,'rb')
    (nchannel, width, rate, length, comptype, compname) = w.getparams()
    print "[%s] %d HZ (%0.2fsec)" %(wavfilename, rate, length/float(rate))
    frames = w.readframes(length)
    return numpy.array(struct.unpack("%sh" %length*nchannel,frames))

def shrink(data,deg=100):
    """condense a linear data array by a multiple of [deg]."""
    global rate
    small=[]
    print "starting with", len(data)
    for i in range(len(data)/deg):
        small.append(numpy.average(data[i*deg:(i+1)*deg]))
    print "ending with", len(small)
    rate = rate/deg
    #return small[40000:50000]
    return small

def normalize(data):
    """make all data fit between -.5 and +.5"""
    data=data-numpy.average(data)
    big=float(max(data))
    sml=float(min(data))
    data=data/abs(big-sml)
    data=data+float(abs(min(data)))-.47
    return data

def smooth(data,deg=20,expand=False):
    """moving window average (deg = window size)."""
    for i in range(len(data)-deg):
        if i==0: cur,smooth=sum(data[0:deg]),[]
        smooth.append(cur/deg)
        cur=cur-data[i]+data[i+deg]
    if expand:
        for i in range(deg):
            smooth.append(smooth[-1])
    return smooth

def smoothListGaussian(list,degree=10,expand=False):
    window=degree*2-1
    weight=numpy.array([1.0]*window)
    weightGauss=[]
    for i in range(window):
        i=i-degree+1
        frac=i/float(window)
        gauss=1/(numpy.exp((4*(frac))**2))
        weightGauss.append(gauss)
    weight=numpy.array(weightGauss)*weight
    smoothed=[0.0]*(len(list)-window)
    for i in range(len(smoothed)):
```

(manned experimental propagation transmistter)
RF (Radio Frequency)
Thermoregulation UCF Lab
Uncategorized

Categories

C/C++ (46)
Circuitry (103)
DIY ECG (16)
Electronics (15)
General (196)
GitHub (16)
HAB (high altitude balloon) (4)
Linux (15)
Microcontrollers (77)
Molecular Biology (3)
My Website (10)
PHP (4)
Prime Numbers (16)
Programming (4)
Python (69)
QRSS / MEPT (manned experimental propagation transmistter) (4)
RF (Radio Frequency) (84)
Thermoregulation (3)
UCF Lab (4)
Uncategorized (2)

Archives

```
        smoothed[i]=sum(numpy.array(list[i:i+window])*weight)/sum(weight)
    if expand:
        for i in range((degree*2)-1):
            smoothed.append(smoothed[-1])
    return smoothed

def goodSmooth(data):
    #data=smooth(fix,20,True)
    data=smooth(fix,100,True)
    #data=smooth(fix,20,True)
    return data

def makeabs(data):
    """center linear data to its average value."""
    for i in range(len(data)): data[i]=abs(data[i])
    return data

def invert(data):
    """obviously."""
    for i in range(len(data)): data[i]=-data[i]
    return data

def loadwav(fname):
    """a do-everything function to get usable, smoothed data from a WAV."""
    wav=readwave(fname)
    wav=shrink(wav)
    wav=invert(wav)
    wav=smooth(wav)
    wav=smooth(wav,10)
    wav=normalize(wav)
    Xs=getXs(wav)
    return Xs,wav

def getXs(datalen):
    """calculate time positions based on WAV frequency resolution."""
    Xs=[]
    for i in range(len(datalen)):
        Xs.append(i*(1/float(rate)))
    print len(datalen), len(Xs)
    return Xs

def integrate(data):
    """integrate the function with respect to its order."""
    inte=[]
    for i in range(len(data)-1):
        inte.append(abs(data[i]-data[i+1]))
    inte.append(inte[-1])
    return inte

def getPoints(Xs,data,res=10):
    """return X,Y coordinates of R peaks and calculate R-R based heartrate."""
    pXs,pYs,pHRs=[],[],[]
    for i in range(res,len(data)-res):
        if data[i]>data[i-res]+.1 and data[i]>data[i+res]+.1:
            if data[i]>data[i-1] and data[i]>data[i+1]:
                pXs.append(Xs[i])
                pYs.append(data[i])
```

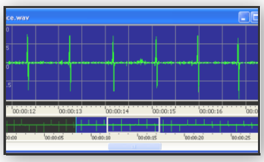

December 2017 (1)
September 2017 (2)
August 2017 (3)
June 2017 (1)
April 2017 (1)
February 2017 (2)
October 2016 (1)
September 2016 (4)
August 2016 (4)
July 2016 (6)
December 2015 (1)
April 2014 (1)
February 2014 (1)
June 2013 (5)
May 2013 (4)
April 2013 (3)
January 2013 (1)
December 2012 (2)
August 2012 (1)
June 2012 (3)
August 2011 (5)
July 2011 (7)
June 2011 (3)
March 2011 (4)
February 2011 (6)
January 2011 (4)
December 2010 (1)
November 2010 (5)
September 2010 (5)
August 2010 (6)
July 2010 (3)
June 2010 (19)
May 2010 (14)
April 2010 (2)
March 2010 (13)
February 2010 (2)

```
                if len(pXs)>1:
                    pHRs.append((1.0/(pXs[-1]-pXs[-2]))*60.0)
    pHRs.append(pHRs[-1])
    return pXs,pYs,pHRs

def bandStop(fft,fftx,low,high,show=True):
    lbl="%d-%d"%(low,high)
    print "band-stopping:",lbl
    if show:
        col=pylab.cm.spectral(low/1200.)
        pylab.axvspan(low,high,alpha=.4,ec='none',label=lbl,fc=col)
        #pylab.axvspan(-low,-high,fc='r',alpha=.3)
    for i in range(len(fft)):
        if abs(fftx[i])>low and abs(fftx[i])<high :
            fft[i]=0
    return fft

def getXs(data):
    xs=numpy.array(range(len(data)))
    xs=xs*(1.0/rate)
    return xs

def clip(x,deg=1000):
    return numpy.array(x[deg:-deg])

pylab.figure(figsize=(12,8))
raw = invert(shrink(readwave(fname),10))
xs = getXs(raw)
fftr = numpy.fft.fft(raw)
fft = fftr[:]
fftx= numpy.fft.fftfreq(len(raw), d=(1.0/(rate)))

pylab.subplot(2,1,1)
pylab.plot(fftx,abs(fftr),'k')

fft=bandStop(fft,fftx,30,123)
fft=bandStop(fft,fftx,160,184)
fft=bandStop(fft,fftx,294,303)
fft=bandStop(fft,fftx,386,423)
fft=bandStop(fft,fftx,534,539)
fft=bandStop(fft,fftx,585,610)
fft=bandStop(fft,fftx,654,660)
fft=bandStop(fft,fftx,773,778)
fft=bandStop(fft,fftx,893,900)
fft=bandStop(fft,fftx,1100,max(fftx))
pylab.axis([0,1200,0,2*10**6])
pylab.legend()
pylab.title("Power Spectral Analysis",fontsize=28)
pylab.ylabel("Power",fontsize=20)
pylab.xlabel("Frequency (Hz)",fontsize=20)

pylab.subplot(2,1,2)
pylab.title("Original Trace",fontsize=28)
pylab.ylabel("Potential",fontsize=20)
pylab.xlabel("Time (sec)",fontsize=20)
pylab.plot(clip(xs),clip(raw),color='.8',label='1: raw')
```

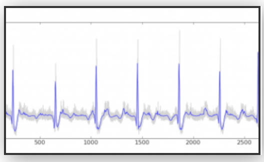
January 2010 (2)
December 2009 (2)
September 2009 (1)
August 2009 (7)
July 2009 (4)
June 2009 (18)
May 2009 (11)
April 2009 (10)
March 2009 (2)
February 2009 (4)
January 2009 (18)
November 2008 (5)

```
fix = scipy.ifft(fft)
pylab.plot(clip(xs),clip(fix)+5000,color='.6',label='2: band-stop')
pylab.plot(clip(xs),clip(goodSmooth(fix))-5000,'k',label='3: smoothed')
pylab.legend()
pylab.title("Band-Stop Filtered Trace",fontsize=28)
pylab.ylabel("Potential",fontsize=20)
pylab.xlabel("Time (sec)",fontsize=20)

pylab.subplots_adjust(hspace=.5)
pylab.savefig('out.png',dpi=100)
pylab.show()
print "COMPLETE"
```

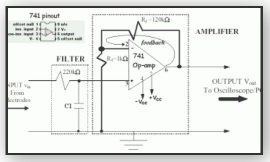
Share this:



ECG Success!
January 14, 2009
In "Circuitry"



DIY ECG Improvements
January 20, 2009
In "Circuitry"



DIY ECG Attempt 1: Failure
January 14, 2009
In "Circuitry"

Circuitry, DIY ECG, General, Python | Leave a comment

‹ DIY ECG Trial 1
› DIY ECG Progress

You must log in to post a comment.

About Scott

Scott Harden lives in Gainesville, Florida and works at the University of Florida as a biological research scientist studying cellular

Meta

Log in
Entries RSS
Comments RSS
WordPress.org

Contact

SWHarden@gmail.com

neurophysiology. Scott has lifelong passion for computer programming and electrical engineering, and in his spare time enjoys building small electrical devices and writing cross-platform open-source software. more →

Subscribe via Email

Enter your email address to subscribe to this website and receive notifications of new posts by email.

Email Address

Subscribe