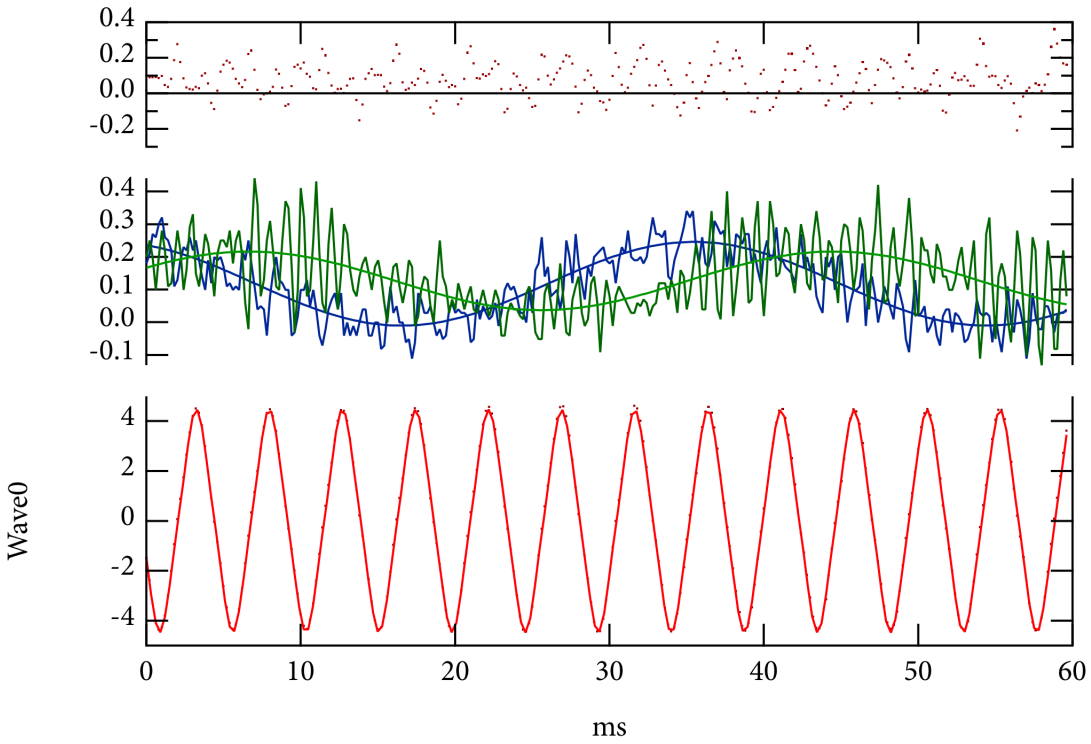
Attempt to Use LabJack U3 to Record Building Vibrations

Thursday, May 26, 2011, 21:01

I set up a python program to attempt to make sense of the data coming from two acceleration sensors placed on adjacent edges of the top plane of a 3-story building of height (very) roughly 1 m. The drive motor was mounted on the first floor, which shouldn't shake as much as the top, in hopes of reducing the back-action on the motor.

The python program, **noise.py**, reproduced below, uses the stream interface to sample four channels continuously at 5000 Hz. The data are written to a text file, which is then repeatedly loaded by Igor. Crude, but reasonably effective. I'll have to see about setting up a pipe so it can happen more smoothly (?).

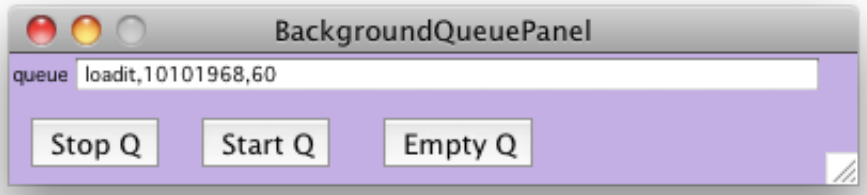
Using a background task, the data were loaded and analyzed a slightly slower than 1 Hz. An example data set with analysis is shown in the figure below.

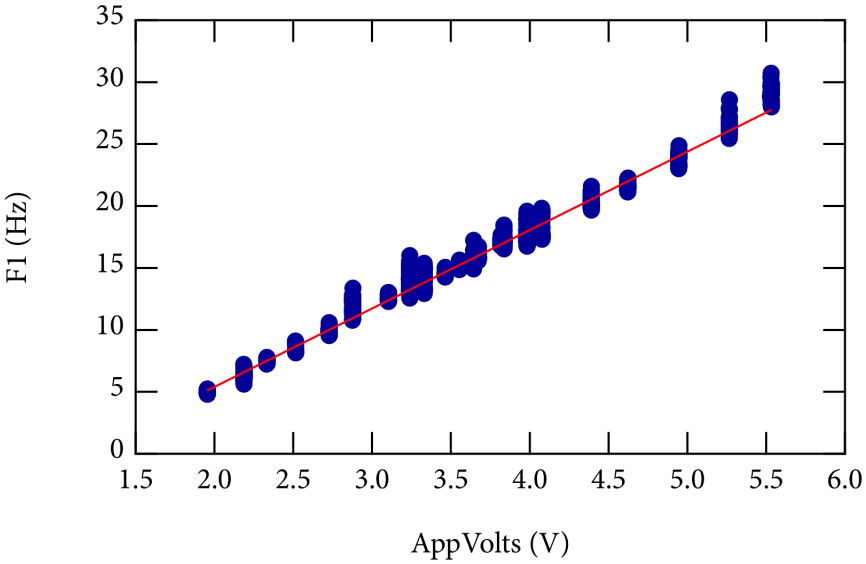


The bottom panel shows the output from the motor's tachometer, which the manual explains gives a signal whose fundamental frequency is 8 times the actual rotation frequency of the shaft. I found that I could fit it reasonably well with a combination of fundamental and third harmonic; the red dots at the top are the residuals.

The middle panel shows the output from the two accelerometers, fitted to a sine wave whose period was constrained to be 8 times the one determined for the tachometer.

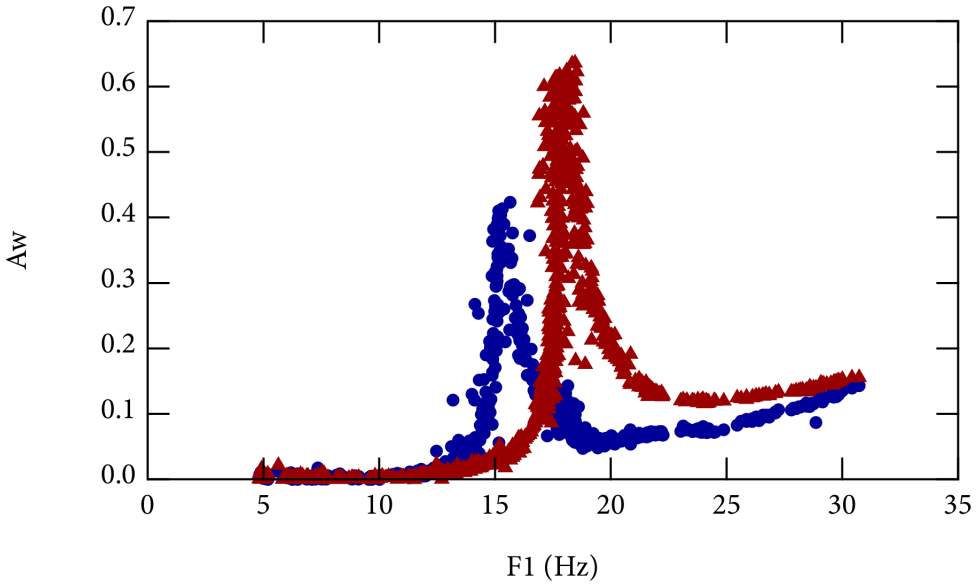
The final analog input channel was given to measuring the voltage applied to the motor, to allow me to correlate drive voltage and rotation frequency. For a given applied voltage setting, we get a range of values for the actually rotation frequency, due to coupling between the motor and the building vibration. I would typically change the voltage, let things settle down for 10–20 seconds, and then begin taking data by clicking the **Start Background** button on the BackgroundQueuePanel.





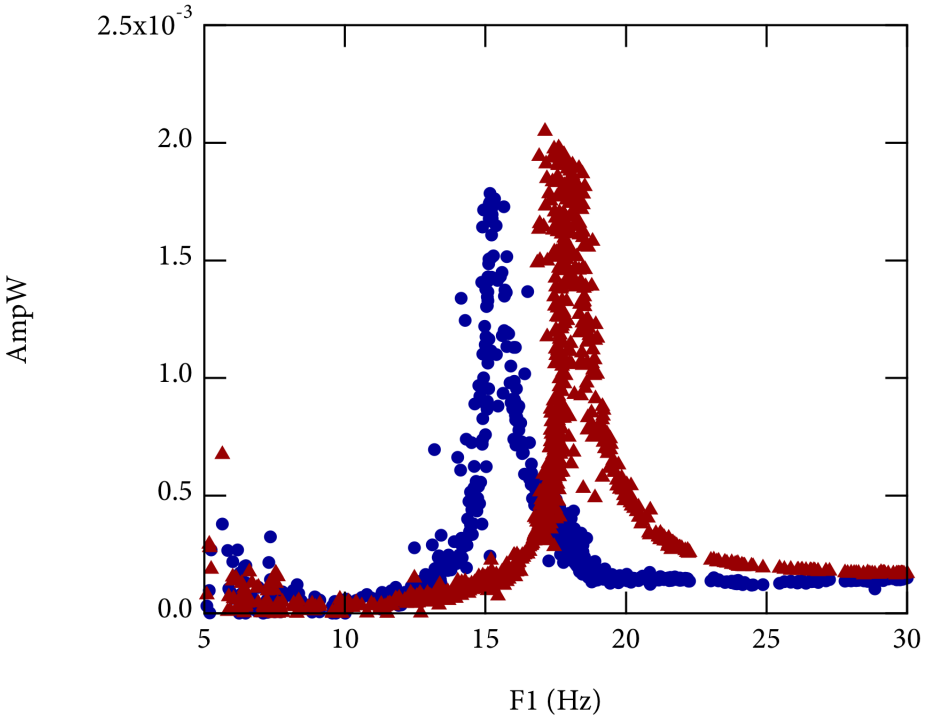
Function: line
Coefficient values ± one standard deviation
a =-7.2428 ± 0.114
b =6.3298 ± 0.0295

Once the analysis routine seemed to be up and running, I had it automatically log data to four global waves of very stupid names: f1, aw, ah, and appVolts. Note that f1 has the recorded tachometer frequency divided by 8. Each point consisted of frequency, the fitted sinusoidal amplitude to each of the accelerometers, and the applied voltage. The accelerations look as follows:



I was expecting the building to be stiffer when shaking parallel to the long side of the rectangular floors, but I'm pretty sure it goes the other way. We'll have to think about that more carefully.

I was bothered by the rise in the data at the right, since things should die after resonance. I think it is caused by the fact that the accelerometer output should be proportional to both the amplitude and the square of the frequency. So, I'll prepare a reduced graph by dividing out the frequency squared.



Better, if not perfect. We still seem to asymptote to a nonzero value at high frequency. I don't know yet whether that has to do with higher-order modes, electrical noise, or ...?

Python Code

```
import u3
from time import sleep

def noiselevel( a ):
   mean = sum(a) / len(a)
   sdiff = 0
   for e in a:
      sdiff += (e-mean)**2
   return ((sdiff/(len(a)-1))**0.5)

d = u3.U3()
d.configU3()
d.configIO( FIOAnalog = 1 )
d.streamConfig( NumChannels = 4, PChannels = [ 0, 1, 2, 3 ], NChannels = [ 31, 31, 31, 31 ],
Resolution = 3, SampleFrequency = 5000 )

def measure():
   try:
      d.streamStart()

      for r in d.streamData():
         if r is not None:
            if r['errors'] or r['numPackets'] != d.packetsPerRequest or r['missed']:
               print "error"
            else:
               res = noiselevel( r['AIN0'] )
            break
   finally:
      d.streamStop()
   return r

def writeData( r ):
   f = open( 'noise.txt', 'w' )
   ch0 = r['AIN0']
   ch1 = r['AIN1']
   ch2 = r['AIN2']
   ch3 = r['AIN3']
   for i in range(0, len(ch0)-1):
      f.write( '{0:.6f}\t{1:.6f}\t{2:.6f}\t{3:.6f}\n'.format(ch0[i],ch1[i],ch2[i],ch3[i]) )
   f.close()

for i in range(1,1000):
   writeData( measure() )
   sleep(1)


Igor Code
#pragma rtGlobals=1           // Use modern global access method.
#include "Background"

// Read data saved by the Python program noise.py into the file noise.txt
// and analyze it. To attempt to head off collisions between the Python program
// and Igor, quickly duplicate the file. Perhaps I should do this with a call to the OS?
Function LoadIt()
   String fname = "Macintosh HD:Users:saeta:Documents:Courses:cl57:Data Acquisition
Stuff:noise.txt"
   Variable refnum
   String everything

   Open /R refnum as fname

   // We need to prepare a string to hold the entire file. Igor uses the length
   // of this string to figure out how many bytes to read. We can get that number
   // from a call to FStatus, which sets V_logEOF
   FStatus refnum
   everything = PadString( "", V_logEOF, 0)     //    pad the string with nulls
   FBinRead refnum, everything                        //    read everything, then
   Close refNum                                             //    close the file

   // It is much more efficient to let Igor's built-in data loader make sense of
```

```
   // the data than to parse the string ourselves. So, write the string out to a
   // temporary file, close the file, then load it.
   fname += ".dat"
   Open refnum as fname
   FBinWrite refnum, everything
   Close refnum

   if ( strlen(everything) < 100 )
      return 0
   endif

   // Look up help for LoadWave to see what the flags do.
   LoadWave/N/Q/G/K=1  fname
   WAVE wave0, wave1, wave2, wave3
   // At present, I have no way of communicating the sample rate between
   // the programs, so I am hard-wiring it in here. This is *ugly* !!
   SetScale/P x 0,2e-4,"s",wave0,wave1, wave2, wave3

   Variable freq, amp
   WAVE f1, ah, aw, appVolts                          //    the "big" waves holding the
results

   Analyze( wave1, wave0, freq, amp )

   // Store away the results by appending a point to each of the "big" waves
   apnt( f1, freq )
   apnt( appVolts, mean(wave3))
   apnt( ah, abs(amp) )
   Analyze( wave1, wave2, freq, amp )
   apnt( aw, abs(amp) )

   return 0                                           //    returning 0 means we're good
                                                      //    to be called again; keep us in
the
                                                      //    background queue.
End

Function Analyze( tach, sensor, freq, amp )
   WAVE tach, sensor
   Variable &freq, &amp

   Variable V_FitOptions = 4        //    suppress fitting window
   WaveStats/Q tach
   tach -= V_avg
   // Let's look for the first two zero crossings
   Variable z0, z1
   FindLevel /Q tach, 0
   if ( V_flag )
      return -1
   endif
   z0 = V_LevelX
   FindLevel /Q /R=(z0+DimDelta(tach,0)*3,inf) tach, 0
   if ( V_flag )
      return -2
   endif
   z1 = V_LevelX
   freq = 0.5 / (z1 - z0)
   Variable t0 = ( V_rising ? z1 : z0 )
   WaveStats/Q tach
   amp = V_max
   Make/D/N=4/O acoeffs
   acoeffs[0] = freq
   acoeffs[1] = t0
   acoeffs[2] = amp
   acoeffs[3] = amp / 25

   FuncFit/Q/NTHR=0 SineFAnd3F acoeffs tach /D/R

   Make/D/N=4/O scoeffs
   WaveStats/Q sensor
```

```
   scoeffs[0] = V_avg
   scoeffs[2] = 2 * pi * acoeffs[0] / 8
   scoeffs[1] = (V_max - V_min) / 2
   scoeffs[3] = 2
   CurveFit /Q /H="0010" sin, kwCWave=scoeffs sensor /D           // k0 + k1 * sin(K2*x+K3)

   WAVE results
   freq = acoeffs[0] / 8
   amp = scoeffs[1]
End

// I want to use a fitting function with the peaks of the fundamental
// and third harmonic in phase
Function SineFAnd3F(w,t) : FitFunc
   Wave w
   Variable t
   Variable x = 2 * pi * w[0] * (t-w[1]) - pi/2
   return w[2] * cos(x) + w[3] * cos(3*x)
End
```