

DAMP — Data acquisition for music
processing

Bernd Porr

October 29, 2010

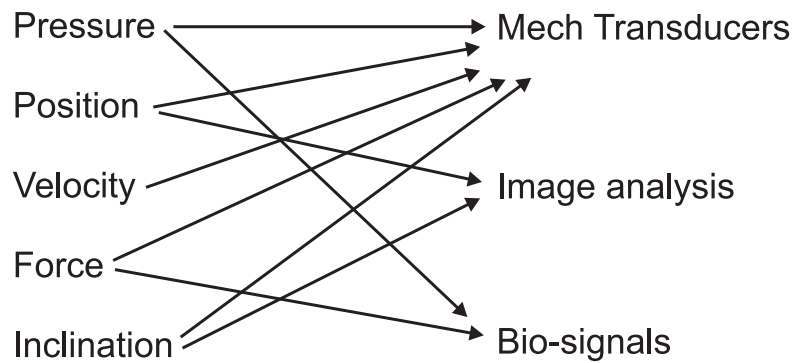


Figure 1: Physical quantities and how to measure them.

1 Introduction

The module is divided into 3 parts. This handout deals with the second part, namely how to measure performance data on your instruments. With performance data we mean anything but sound, be it the angle the bow is hold or how you pluck a string on your guitar. Consequently we are discussing now all different kinds of sensors which could be used for measuring performance data.

Fig. 1 gives an overview over the mapping from physical quantities to sensors. The general idea is to translate a physical quantity into a proportional voltage.

In the following section we are going to present different sensors.

2 How to measure...?

2.1 Pressure/Force

2.1.1 Piezo effect

The piezo effect uses the fact that some materials produce a voltage when pressure is applied. Simple piezo sensors can be found in greeting cards as sounders (yes they also work the other way round) and cost a pound whereas calibrated sensors cost easily hundreds of pounds. The problem is that the

voltage output of a piezo is differential meaning that when pressed it produces a voltage but this decays quickly. In order to measure pressure this voltage needs to be integrated. (Farnell 1192520).

2.1.2 Pressure gauges

These are Wheatstone bridges which need an instrumentation amplifier which transforms the differential signal into a voltage with ground reference. (Farnell 4982277)

2.1.3 Strain gages

These are usually strips glued on a surface and change their resistance when bent.

2.1.4 Resistive pressure sensors

These sensors are relatively new and not that easy to get. However they are superior to Piezo sensors and are very robust. The resistance is inversely proportional to the pressure (www.interlinkelectronics.com).

2.1.5 Muscle activity

Every muscle in the body can generate quite a high voltage up to 10mV and can easily be measured by ECG surface electrodes. Measurement requires a differential amplifier to cancel out noise. (Farnell 1648643).

2.2 Position

2.2.1 Linear voltage dividers

Use simple voltage dividers to measure position wherever possible. For example a string of a guitar has a certain resistance and can be used as a voltage divider.

2.2.2 Inclination

There are sensors around which measure the angle against the ground. Very often they just give a very rough estimate of the angle such as “vertical” or

“horizontal”. These need to be combined with more precise measurements. (Farnell 178338).

2.2.3 Acceleration

Acceleration can be used to calculate position:

$$s(t) = \int \int a(t) dt dt$$

gives us the position. Note that integration in the digital domain is just “adding up” the samples up to the last sample n :

$$s(n) = \sum_{m=0}^n a(m)$$

Accelerometers are usually just in standard ICs. (Farnell 1566165).

2.2.4 Hall sensors

These sensors react on the magnetic field and change their output proportional to the magnetic field. (Farnell 1521708).

2.2.5 Visual tracking

A video camera can be used to track movements of markers or highly salient features in the scene. The most reliable marker is an LED which gives a distinct saturated dot in a video recording. Image processing can be done with the image library openCV (<http://opencv.willowgarage.com>) which is part of most Linux distributions. A simple webcam can do the job of tracking an object. openCV has different functions to track objects. The quickest function tracks points in objects with a technique called Lukas/Kanade tracker. There are also functions to “train” a detector to recognise a certain object with the help of positive and negative examples. The book “Learning openCV” by Bradsky et al is a very good starting point to learn how to use openCV (it’s in the library). However, the best approach is to “hack” one of the example programs which come with openCV. In our case the “lkdemo.c” is useful which demonstrates the functions “good features to track” (which finds you good points to track) and the Lukas Kanada tracker (which tracks features

from one frame to the next). This demo also uses the mouse so that the user can click on points he/she would like to track.

How to compile programs using openCV? Because it's a library you need to do two things:

1. Include the header files:

```
#include <opencv/cv.h>
#include <opencv/highgui.h>
```

in your c program at the top.

2. Create a Makefile which points to the directories where the header files are located and links the libraries into the compiled code:

```
all: myprogram
```

```
myprogram.o: myprogram.c
    g++ -I/usr/include/opencv -c myprogram.c
```

```
myprogram: myprogram.o
    g++ -o myprogram myprogram.o -lxcvcore -lcv -lhighgui -lcvaux -lm
```

Note that the indents are tab stops.

3. Type "make" and the program will be compiled. To start it type "./myprogram".

3 Digital data acquisition with COMEDI

In this section we are describing briefly how the analogue voltages are digitised and how they can be processed under Linux.

3.1 A/D converters

Every A/D converter is characterised by its sampling rate and by its resolution. The A/D converter we are using has a resolution of 12 bits which means that it divides the input range into 4096 equal steps. The input range is conveniently from $-4.096V \dots + 4.096V$. This means that every step is $2mV$.

Numbers can be represented as signed and unsigned numbers. COMEDI represents all numbers as unsigned numbers. This means that the numerical value 0 corresponds to $-4.096V$. The numerical value of 2048 to zero volts and so on. To plot measurements in voltages they need to be converted which is simply a shift and multiplication. The shift is just a subtraction by 2048 and imagine we want to plot our measurements in mV we just need to divide the shifted A/D numbers by 2. If we want to have it in volts we divide the readings by 2000. In general A/D values can be converted by the formula

$$x(n) = \frac{x_{AD} - a}{b}$$

In our case $a = 2048$ and $b = 2000$. However if we have a pre-amplifier in front of the A/D converter we need to take its gain into account. For example if the pre-amplifier has a gain of 100 we have to set $b = 2000 \cdot 100$.

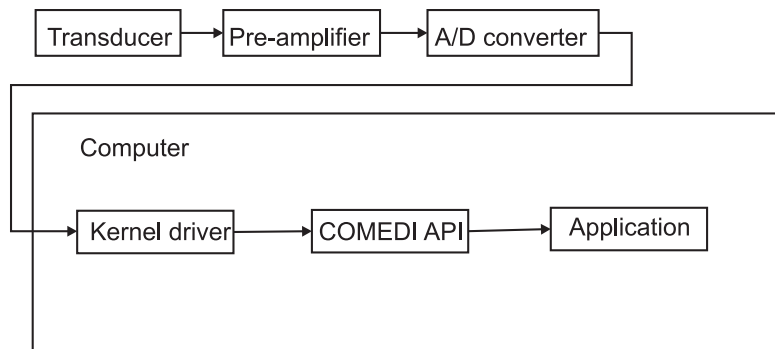


Figure 2: General data flow diagram for analogue/digital processing

Fig. 2 shows a typical flow diagram of how data is processed here. We are going to concentrate on the computer side which uses the so called COMEDI framework which provides a standard API for data acquisition.

3.2 synchronous acquisition

In this mode the application program sends a request through COMEDI to the DAQ card and then is forced to wait until the result is available:

1. request data

2. wait for data
3. get data
4. jump to 1.

In COMEDI synchronous acquisition is done in the following way:

```
comedi_t *device;  
lsampl_t data;  
device = comedi_open('/dev/comedi0');  
comedi_data_read(device,0,0,0,0,&data);  
printf('%d\n',data);
```

The problem is that the time between measurements is not defined and therefore the sample rate is also not defined. Therefore we cannot apply any operation to the samples which imply a constant sample rate. For example integration assumes a constant sample rate and even plotting a curve needs equally spaced samples.

3.3 a-synchronous acquisition

In this mode data is acquired while the main program is performing other tasks. Or in other words the acquisition is running in the background.

1. Initialise the background process (number of channels, etc)
2. Start background/data acquisition process
3. Do something else
4. Get data in chunks
5. go back to 3.

This is the mode you should be working with. COMEDI has a working example called “cmd.c” in comedilib and also comedirecord can be used as a working example.

Here are the essential commands to start an asynchronous acquisition:

```

static unsigned int chanlist[N_CHANS];
comedi_cmd cmd;
comedi_t *dev;
dev = comedi_open(“/dev/comedi0”);
chanlist[0]=CR_PACK(0,0,0);
chanlist[1]=CR_PACK(1,0,0);
comedi_get_cmd_generic_timed(dev,0,&cmd,1E9/frequ);
cmd.chanlist=chanlist;
cmd.chanlist_len=2;
comedi_command_test(dev,cmd);
ret=comedi_command_test(dev,cmd);
if (ret!=0) exit(1);
comedi_command(dev,cmd);

```

the last command starts the acquisition in the background. Reading data is done by:

```

while(1){
    ret = read(comedi_fileno(dev),buf,BUFSZ);
    if(ret < 0){
        /* some error occurred */
        perror("read");
        break;
    }else if(ret == 0){
        /* reached stop condition */
        break;
    }else{
        /* ‘ret’ samples in the buffer */
        /* PROCESS DATA */
    }
}

```

4 How to post-process and display data

Usually you need to analyse and display data. A good program to display data and to minor modifications is “gnuplot”. For more sophisticated data manipulations “octave” is a good choice which is a free MATLAB clone and is a very powerful tool.

Both programs expect the data to be in simple text files where every row contains samples from one moment in time:

```
1 2048 2069 2073
3 2052 2091 2091
5 2061 2117 2109
7 2060 2132 2121
9 2066 2135 2118
11 2065 2129 2112
```

Because it is a textfile it can be edited and viewed with any text editor. It's a very inefficient way of storing data but also the most compatible.

4.1 GNUPLOT

GNUPLOT can plot both analytical functions and numerical data. For example, `plot sin(x)` plots the sine function. The command `plot 'mydata.dat'` plots the data in the file, usually the first two columns. GNUPLOT is command driven which means that if, for example, you want to change the axis then you would write `set xrange [0:10]`. To save a plot as an image file you need to tell GNUPLOT which format you want and the filename which is done with two commands:

```
gnuplot> set term postscript eps
Terminal type set to 'postscript'
Options are 'eps noenhanced defaultplex \
  leveldefault monochrome colortext \
  dashed dashlength 1.0 linewidth 1.0 butt \
  palfuncparam 2000,0.003 \
  "Helvetica" 14 '
gnuplot> set output "myplot.eps"
gnuplot> replot
```

This generates an “encapsulated postscript” file which is used in professional publishing because it is a vector based format (no pixels!). This can be loaded into open Office or can also be modified with inkScape and then imported into a word processor or LaTeX.

4.2 OCTAVE

Octave is a very feature rich program which works with vectors and matrixes in an efficient way. It makes it very easy to manipulate data and plot it. As GNUPLOT it is command line driven and the commands can be stored as a script so that they can be re-used next time.

OCTAVE comes with an excellent online help and there are many example scripts on the Internet.

Here, we explain the special features of the program, especially how it processes vectors and matrixes. Besides that it operates as any other programming language with for-loops, if-conditionals and function definitions. Please consult the online help for exact definitions.

4.2.1 Simple Arithmetic

- `2+3` prints the result
- `a=2+3` stores it in the variable `a`
- `a=2+3;` suppresses the result
- `a` prints the result!

4.2.2 Numbers

- `6E23=` $6 \cdot 10^{23}$
- `2i` or `2j` are complex numbers.

4.2.3 How to Script

You can either type in commands directly in the command line or store them in a file as a list of instructions. Write your commands in *any* text editor and save it as an `.m`-file. Within OCTAVE you can start an editor by writing `edit hello.m`. Under Windows you can use notepad to edit the `m`-files. Under Linux any text-editor, such as `gedit` or `emacs` can be used. Run the script within OCTAVE by typing just the name of the file without the `.m`-suffix, for example `hello`.

4.2.4 Vectors

- `p = [1 9 77]`; or `p = [1,9,77]`; is a vector with the components 1,2 and 77.
- `p = (0 : 0.1 : 1)`; generates 0,0.1,0.2,...,1.
- `p(2)` gives the second element of the vector where the index counts from 1 (!).

What happens if we type?

```
x = (0:0.1:6.2);  
y = sin(x);  
y
```

The variable `y` is again a vector! All elements are processed at the *same* time. Also `y=y+5`; shifts the whole array `y` by 5. A bit tricky is it when we have a product of two vectors because the product is ambiguous. Element-wise multiplication is indicated by a preceding dot: `.*`.

4.2.5 Plotting of functions

Plotting a sine wave:

```
x = (0 : 0.1 : 2 * pi);  
y = sin(x);  
plot(x,y);
```

If you want to create more plot-windows you can use the command `figure`. Access the windows with the commands `figure(1)`, `figure(2)`. How to plot more than one function? `plot (x, sin(x), x, cos(x))`; It is also possible to combine plot windows into one window by tiling them with the command `subplot`. The online help explains how this can be done.

As in GNUPLOT changes to the coordinate system are achieved by text commands:

```
axis([-10,10,-1,1]);  
title('Blah');  
xlabel('time');  
ylabel('sinc');
```

4.2.6 Saving graphics as a file

With the `print` command you can save a figure on the hard drive in various formats or directly send it to the printer. This is postscript by default and can be imported into LaTeX, Inkscape, CorelDraw or Adobe Illustrator. In general use a vector file format (encapsulated postscript, scalable vector graphics, windows meta file, ...) and not a pixel based format (especially JPEG is a nono!). Consult the online help to find out more about the different options.

4.2.7 Matrix

The matrix:

$$a = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \quad (1)$$

is entered in OCTAVE in the following way:

```
a = [1 2 3; 4 5 6; 7 8 9];
```

or

```
a = [1,2,3;4,5,6;7,8,9];
```

Note the use of spaces and “;”.

How to access a single element?

```
a(1,2)
```

gives us the result 2.

and a couple of elements to form a vector:

```
a(1:2,2) access the second column and the first two rows.
```

```
a( : ,2) access all elements of the second column.
```

4.2.8 Importing data

OCTAVE likes data in matrix format where the elements are separated with spaces (same as GNU PLOT). To load such data files just type in: `load 'filename.dat'`. This creates a matrix with the same name. For example, the file:

```
bernd:~/teaching> cat ekg2.dat
0 22 -39 -661
3 44 -6 -664
6 59 32 -668
9 61 76 -672
12 58 120 -677
15 50 141 -681
18 39 130 -685
```

turns into the matrix:

```
octave:1> load 'ekg2.dat'
octave:2> ekg2
ekg2 =
```

```
    0    22   -39  -661
    3    44    -6  -664
    6    59    32  -668
    9    61    76  -672
   12    58   120  -677
   15    50   141  -681
   18    39   130  -685
```

```
octave:3>
```

Thus, if you want to write software which creates OCTAVE readable data then just export it as space separated ASCII files (the program “comedi-record”, for example, does it).

4.2.9 Fourier Transform

It is very easy to manipulate data with the help of the Fourier transform. Imagine you have recorded 1000 samples at a sampling rate of 1kHz. The Fourier transform will create a complex array with 1000 values. The first 500 values correspond then to all frequencies from DC to 500Hz. Important is that the next 500 values represent the mirror of the first 500 values. If you manipulate the spectrum it is important to change always both the spectrum and its mirror. If this is not done we get a complex time domain function when we do the inverse Fourier transform. For example, we want to remove 50Hz from this signal:

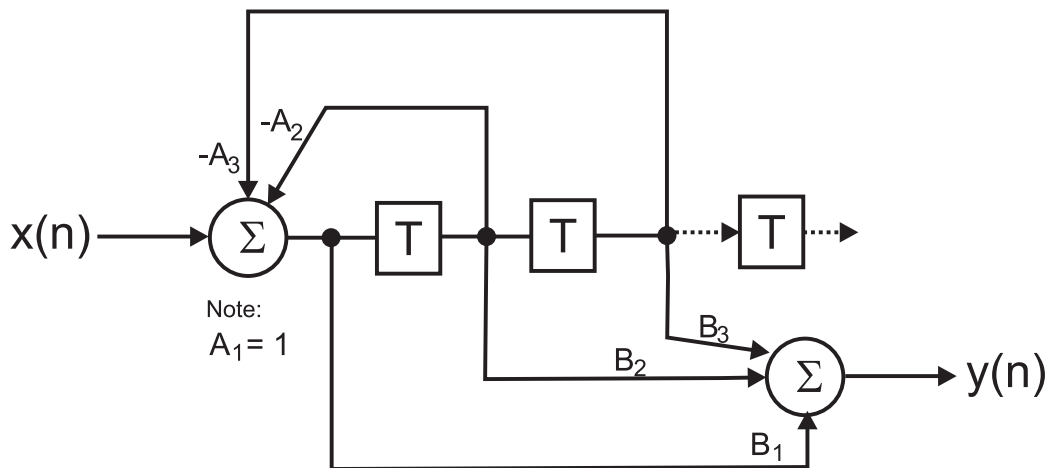


Figure 3: Digital filtering with the `filter` command. The filter takes the input array $x(n)$ and processes it by using the coefficients A and B and generates the output $y(n)$. The coefficients in the array $B = [B_1 B_2 B_3 \dots]$ create a weighted sum of the delayed input signal $x(n)$ in a feed-forward manner. The coefficients $A = [1 A_2 A_3 \dots]$ create feedback loops of the delayed output $y(n)$.

```

octave:9> y=anotherecg2ms(1:1000,2)-2048;
octave:11> y2=fft(y);
octave:12> y2(51)=0;
octave:13> y2(951)=0;
octave:14> y3=ifft(y2);
octave:16> y4=real(y3);
octave:17> plot(y4);
octave:18>

```

4.2.10 Filtering

The reason to use octave is mainly to post process data. Besides scaling and shifting data octave has powerful functions to filter data. Central to this is the function “`filter`” which is able create weighted sums of data (so called FIR filters) and also to generate feedback circuits (so called IIR filters). Here, we concentrate on basic operations which are easy to understand without knowing the theory behind digital signal processing.

The general form of data flow of the function “filter” is shown in Fig. 3. The syntax in octave is

```
y = filter (B, A, X)
```

where the coefficients in B represent a feed forward path and the coefficients in A represent a feedback pathway. Note that the first coefficient in A needs to be one because it just scales the output of the filter but does not create a feedback pathway.

We are now showing a couple of operations which are useful for data processing.

- **Averaging** The simplest form of signal processing is averaging where we take a couple of samples and sum them up.

```
load 'mydata.dat'  
y=mydata(:,2);  
average_coeff=[1 1 1 1 1 1 1 1 1 1];  
average_coeff=average_coeff/10;  
y2=filter(average_coeff,1,y);  
plot(y2);
```

This, for example loads data from the file “mydata.dat”, extracts the second column, sums up 10 samples, stores them in the array y2 and plots them.

- **Differentiation** A differentiation in the digital domain is just a subtraction of two consecutive samples from each other, so the coefficients are:

```
diff_coeff=[-1 1];  
y2=filter(diff_coeff,1,y);
```

- **Integration** The integration is a bit more tricky because it needs to add *all* samples from the past. This could be done with a recursive setup.

```
int_coeff=[1 -1];  
y3=filter(1,int_coeff,y);
```

Note that now integration is done by a recursive loop where the last sample is added to the new sample. In this way the filter accumulates the data.

This setup might be unstable because it's on the verge of self amplification. One could set up an "leaky" integrator which slightly "forgets" previous input values by setting the filter coefficients to `int_coeff=[1 -0.999]`; instead of `int_coeff=[1 -1]`;

Besides these simple examples, one can design any filter especially low-, high-, band-passfilters by calculating the coefficients for A and B.