Dyson School of Design Engineering

Imperial College London

DE2.3 Electronics 2

Lab Experiment 2: Signal Processing with PyBench & Matlab

(webpage: http://www.ee.ic.ac.uk/pcheung/teaching/DE2_EE/)

Objectives

By the end of this experiment, you should have achieved the following:

- Control the PyBench board from your PC running Matlab.
- Generate and capture real electrical signals using PyBench.
- Analyse spectrum of sound signals captured from the microphone/amplifier.
- Examine the effect of aliasing effect due to under sampling.
- Frequency resolution on the spectrum computed using FFT.
- Understand the impact of using different types of windows on signals.
- Analyse sound signal stored as a .WAV file on a computer.
- Perform signal segmentation using energy of signal.
- Analyse the spectrum of actual musical sound.

Before you start

Download from the course webpage the file "PyBench.m", and put this in your Matlab work directory in its search path. This is a Matlab program I have written so that you can control the Pybench board using Matlab commands/instructions.

The PyBench board has many functions and can produce signals as well as capturing signals. The various modules on the PyBench board is interfaced to the Pyboard itself (the board that you used last year for DE1.3 Electronics 1 module). Here is the usage of the pins on the PyBoard:

PIN	FUNCTION
X1	Motor PWM_A/Servo 1
X2	Motor PWM_B/Servo 2
X3	Motor control AIN1/Servo 3
X4	Motor control AIN2/Servo 4
X5	Analogue OUTPUT
X6	SW0
X7	Motor control BIN1
X8	Motor control BIN2
X9	IMU-I2C SCL
X10	IMU-I2C SDA
X11	POT5K
X12	Analogue INPUT

PIN	FUNCTION4›
Y1	BLE-UART Tx
Y2	BLE-UAR Rx
Y3	SW14›
Y4	Motor sensor A_A
Y5	Motor sensor A_B
Y6	Motor sensor B_A
Y7	Motor sensor B_B
Y8	OLED-i2C RST
Y9	OLED-I2C SCL
Y10	OLED-I2C SDA
Y11	Microphone amplifier
Y12	Unused

This pin allocation table is useful for you to understand how the system works when you examine the Python code that drives the PyBench board. It is here for future reference.

Exercise 0: Connecting PyBench board to your PC running Matlab

The PyBench board has a 2-way DIP switch. This determines what the board does when you first apply power to it when running code stored on the MicroSD card. The card is preloaded with a number of programs written in Python:

Program	Purpose
boot.py	Boot file specifying which is the main program.
main.py	Test the DIP switch setting and execute the corresponding .py file.
pybench_main.py	The controlling program for pybench to interprete commands. Run if SW = 00.
pybench.py	The pybench class library. Can be used in your own application program later.
pybech_test.py	Self-test program for the pybench board to verify the hardware. Run if SW = 11.
oled_938.py	OLED display driver class library.
font.py	Character fonts used by oled_938.py.
mpu6050.py	IMU driver class library – to communicate with the accelerometer and gyroscope.
drive.py	Drive class for the motor driver chip TB6612.

With the DIP switch set to "00", connect your PC to the PyBench board using the micro USB cable provided. You should see a message on the OLED display indicating that pybench.py program is running. The BLUE LED should be ON.

Now enter in Matlab command window the following command: **pb = PyBench(device_name)**

where for Mac:	device_name = '/dev/tty.usbmodem14x2'	where x = 1 or 2
for PC:	device_name = 'COMx'	where x = 1, 2, 3, or 4

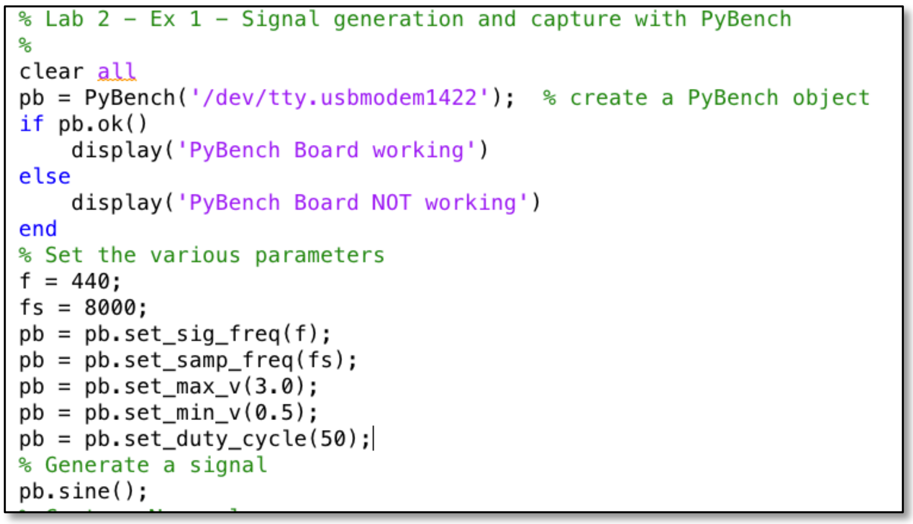
This command creates an object "**pb**" which, in Matlab, is the PyBench board with its associated methods (in object oriented speak). Here are the available methods you can use to control the PyBench board. Analogue output signal is produced on pin X5 of the Pyboard, the top BNC connector and the top test pin. Analogue input is captured from pin X12, the bottom BNC connector and the lower test pin.

Methods	Purpose
pb.ok ( )	Return 1 (true) if PyBench board is working.
pb.set_sig_freq (f)	Set signal frequency to f. 0.1 Hz ≤ f ≤ 3000 Hz
pb.set_samp_freq (f)	Set sampling frequency to f. 1 Hz ≤ f ≤ 30,000 Hz
pb.set_max_v (v)	Set maximum amplitude to v. 0 ≤ v ≤ 3.3
pb.set_min_v (v)	Set minimum amplitude to v. 0 ≤ v ≤ 3.3
pb.set_duty_cycle (d)	Set duty cycle of a square signal to d. 0 ≤ d ≤ 100
pb.dc (v)	Output a dc voltage v. 0 ≤ v ≤ 3.3
pb.sine ( )	Output a sinusoidal signal at set signal frequency between max_v and min_v.
pb.triangle ( )	Output a triangular signal at set signal frequency between max_v and min_v.
pb.square ( )	Output a square signal at set signal frequency between max_v and min_v, with the set duty cycle.
v = pb.get_one ()	Capture one sample v from analogue input. 0 ≤ v ≤ 3.3
data = pb.get_block (n)	Capture n samples from analogue input. 0 ≤ data ≤ 3.3
data = pb.get_mic (n)	Capture n samples from microphone. 0 ≤ data ≤ 3.3
[p, r] = pb.get_accel ( )	Get pitch angle p and roll angle r from the IMU. -90 ≤ p, r ≤ +90
[dx, dy, dz] = pb.get_gyro ( )	Get accelerations (dx, dy, dz) in three axes from the IMU in degrees/sec.

If you now type: **pb.ok( )** in Matlab, it should return '1' or true to indicate that all is well.

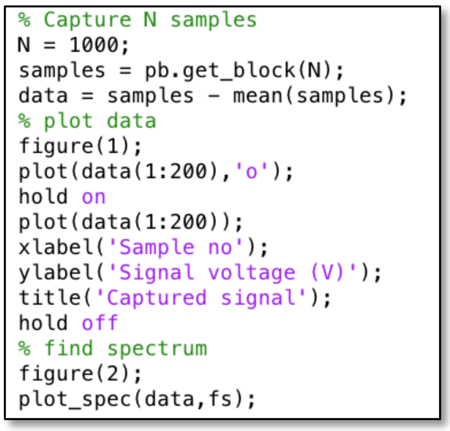
Exercise 1: Generate and capture signals using PyBench board

- Connect the top BNC connector on the PyBench board to Channel 1 of the oscilloscope using the BNC cable provided. Switch on the scope.
- Make sure that the analogue output is looped back (connected) to the analogue input via a shorting link on the left 4-way header pins (shorting the two middle pins, pin2 and pin3).
- With Matlab's editor, create the file **ex1.m** containing the following script, and run it by enter **ex1** in the command window.



```
% Lab 2 - Ex 1 - Signal generation and capture with PyBench
%
clear all
pb = PyBench('/dev/tty.usbmodem1422');  % create a PyBench object
if pb.ok()
    display('PyBench Board working')
else
    display('PyBench Board NOT working')
end
% Set the various parameters
f = 440;
fs = 8000;
pb = pb.set_sig_freq(f);
pb = pb.set_samp_freq(fs);
pb = pb.set_max_v(3.0);
pb = pb.set_min_v(0.5);
pb = pb.set_duty_cycle(50);
% Generate a signal
pb.sine();
```

- Use the scope to check that a 440Hz sinewave is generated by the PyBench board.
- Now try changing various parameters such as signal frequency and voltages.
- Generate triangle and square waves, and observe them on the scope.
- Add the following lines to **ex1.m** (**plot_spec.m** is from Lab 1). Make sure that you under the code.



```
% Capture N samples
N = 1000;
samples = pb.get_block(N);
data = samples - mean(samples);
% plot data
figure(1);
plot(data(1:200),'o');
hold on
plot(data(1:200));
xlabel('Sample no');
ylabel('Signal voltage (V)');
title('Captured signal');
hold off
% find spectrum
figure(2);
plot_spec(data,fs);
```

- Explain the spectrum for the three different signals generated and check that the harmonic components are as you expected.
- Now modify plot_spec.m to plot_spec_full.m, and plot the spectrum over the entire positive and negative frequency range.

Don't forget to record what you have done in your electronic logbook. You can use screen capture for almost everything, and type your thoughts and observations into your eLogbook. During the lab oral (in the DRAW week), you will be asked questions that will require you to refer to your logbook for answers.

Exercise 2 – Capture and analyse microphone signal

Create the Matlab script **ex2.m** in the Lab 2 directory with the following Matlab commands:

```
% Lab 2 - Ex 2 - Capture and analyse microphone sound signal
%
clear all
pb = PyBench('/dev/tty.usbmodem1422');  % create a PyBench object
% Set the various parameters
fs = 8000;
pb = pb.set_samp_freq(fs);
% Capture N samples
N = 1000;
samples = pb.get_mic(N);
data = samples - mean(samples);
% plot data
figure(1);
clf
plot(data);
xlabel('Sample no');
ylabel('Signal voltage (V)');
title('Microphone signal');
% find spectrum
figure(2);
plot_spec(data, fs)
```

If you run **ex2.m** multiple times while whistling, you should see the effect of your whistle on the spectrum.

Whistling continuously is not only tiring; it is also impossible to control the frequency precisely. Download one of many free "**tuning fork**" apps for your phone and generate a tone of selected frequency using the app. You should now be able to capture and measure the frequency of the tone being generated without tiring your lips!

Now modify **ex2.m** so that you capture microphone data and display the spectrum **continuously**. (You need to use a while loop, and the command "clf" to clear the current figure window.) Congratulations, you have built yourself a spectrum analyzer!

Now change the tuning fork frequency from 3000 Hz to 4500 Hz in steps of, say, 100Hz. Explain what you discover.

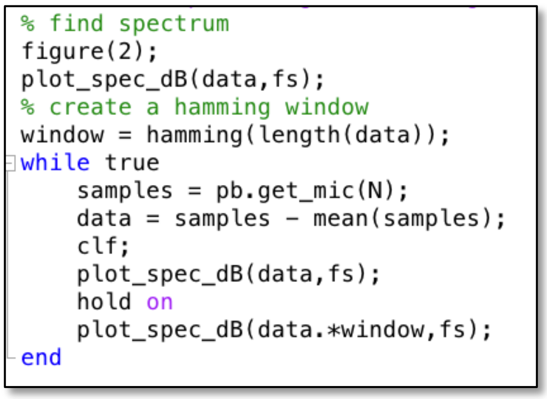
Now modify the number of data samples N captured each time for analysis. What is the effect of N on the the spectrum of the signal?

Exercise 3 –Windowing effect on signal

Download from the course webpage my version of **plot_spec_dB.m**, which plots the magnitude spectrum in decibel (dB). Modify your previous program (**ex2.m**) to use **plot_spec_dB.m** so that the spectrum of the signal captured from the microphone is displayed (and updated) continuously. Displaying the magnitude spectrum in a logarithmic scale provides much high sensitivity than in a linear scale. The plot is also normalized in a way that the maximum frequency component is at 0dB, i.e. all spectral components are relatively scale to the peak spectral value. Finally, the magnitude axis is limited to 0dB to -60dB.

Now obtain the spectrum for tuning fork sound at 1000Hz and at 1100Hz. Observe the difference in the spectrum produced. What you see here is the impact of exacting a portion of the signal to analyse. This effective "views" the signal through a rectangular window.

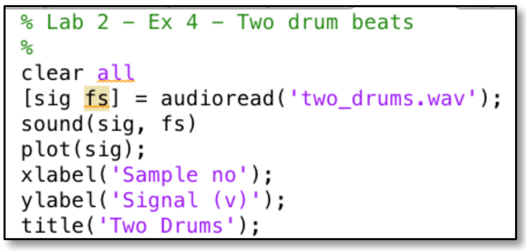
Modify your program **ex3.m** to **ex3a.m** with the following Matlab code for finding the spectrum. Here we apply a window (known as **Hamming Window**) to the signal before calculating the spectrum. We also plot the original spectrum and that using the Hamming Window. I will be explaining why you see the difference on Thursday's lecture.



```
% find spectrum
figure(2);
plot_spec_dB(data,fs);
% create a hamming window
window = hamming(length(data));
while true
    samples = pb.get_mic(N);
    data = samples - mean(samples);
    clf;
    plot_spec_dB(data,fs);
    hold on
    plot_spec_dB(data.*window,fs);
end
```

Exercise 4 – Music signal segmentation and analysis

Download from the course webpage, the file '**two_drums.wav**'. Create the file ex4.m to include the following code:



```
% Lab 2 - Ex 4 - Two drum beats
%
clear all
[sig fs] = audioread('two_drums.wav');
sound(sig, fs)
plot(sig);
xlabel('Sample no');
ylabel('Signal (v)');
title('Two Drums');
```

From now on, we will be analyzing musical signals stores on your computer in the form of waveform files instead of using the microphone on the PyBench board. This allows you to work at home on your own PC without the need of any hardware. Note that fs is the sampling frequency. For these computer stored waveforms, fs = 44100.

The tasks in this exercise are to:

1. Divide the signal into 20 msec segments, and computer the energy of the signal in that segment. The energy is defined as:
   $$\sum_{i=1}^{N} x^2(t)$$ where N is the number of samples in 20ms.
2. Plot the energy graph to identify where the peaks are manually.
3. Look up the Matlab function: '**findpeaks**', and put a label (e.g. 'x' or 'o') on the energy graph.
4. Discover the dominant frequencies of the two drums.

While you are encourage writing the rest of the code for Exercise 4 yourself, the solution for tasks 1 to 3 is provided in the Appendix.

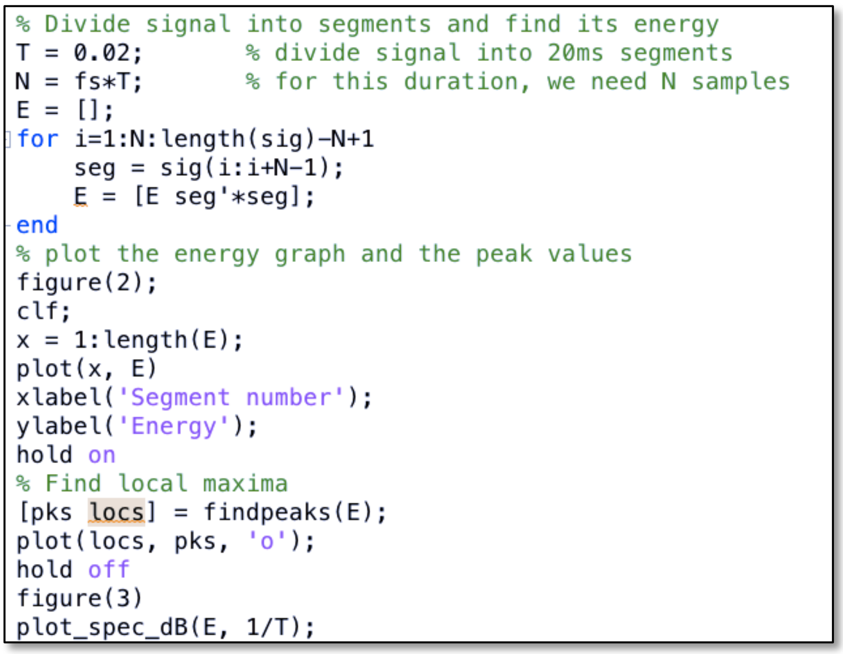
Exercise 5 – Analysing complex music

This exercise is open-ended and you may want to do this outside the normal laboratory time.

Download two further music files: **quitar.wav** and **bass.wav** from the course webpage. Play with these and extract various features from them (such as beat and spectral information). You may also add the two signals together and analyse the combined signal.

Appendix - Solution to Exercise 4

Additional code …….



```
% Divide signal into segments and find its energy
T = 0.02;        % divide signal into 20ms segments
N = fs*T;        % for this duration, we need N samples
E = [];
for i=1:N:length(sig)-N+1
    seg = sig(i:i+N-1);
    E = [E seg'*seg];
end
% plot the energy graph and the peak values
figure(2);
clf;
x = 1:length(E);
plot(x, E)
xlabel('Segment number');
ylabel('Energy');
hold on
% Find local maxima
[pks locs] = findpeaks(E);
plot(locs, pks, 'o');
hold off
figure(3)
plot_spec_dB(E, 1/T);
```