Dyson School of Design Engineering

Imperial College London

DE2 Electronics 2

Lab Experiment 2: Signal Processing with PyBench & Matlab

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

Objectives

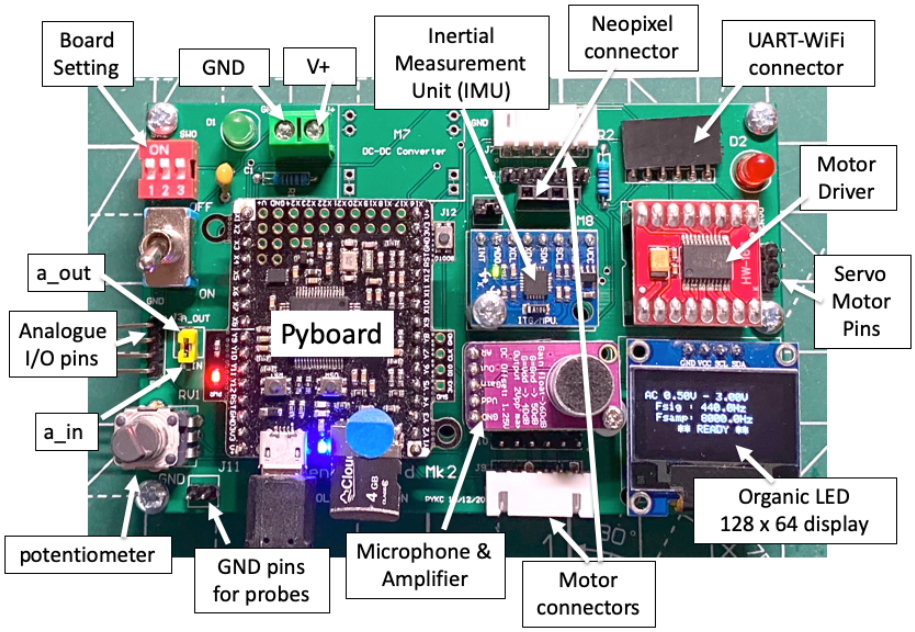
By the end of this experiment, you should have achieved the following:

- Control the PyBench board from your PC or MacBook 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.

The Lab-in-a-Box

All of you should have picked the lab-in-a-box between you and your lab partner. It should contain two items: a PyBench board and a Micro USB cable. You are required to return this kit to return this and other lab equipment to your final week lab oral examiner to be used by next year's students.

The PyBench Board



Unlike Electronics 1, you are **not** required to build electronics circuits on breadboard this year. Instead, you will be using a bespoke board (PCB) that I designed specifically for this module This unit is known as **PyBench Board** – it is my version of an electronic workbench which is programmed using MicroPython (uPy), and controlled via the USB serial port.

The PyBench has many functions and can produce as well as capture signals. The various components on the PyBench board uses a microcontroller known as the PYB v1.1 (full name is Pyboard). Note that PYB is NOT the same as Raspberry Pi. Like the Raspberry Pi, it also uses an ARM processor, but it runs MicroPython out of the box. It is like the ESP32 module you used last year except that it is faster, the DAC output works properly, and it is easier to use.

Task 1: Connecting PyBench to your PC or Macbook running Matlab

Step 1: Connect the PyBench to your computer using the USB to MicroUSB cable provided. You should find a new disk appearing with the label 'PYBENCH3'. This is the MicroSD card plugged into the PYB module. Examine the contents of this device either using Windows 10 Explorer or Mac OSX Finder. This contains files as listed in the table below (mic.py is now called audio.py):

Program	Purpose
boot.py	First to run, the boot file specifies which is the main program.
bulb_test.py	Test and calibration program for the small plug in board for Lab3. Run if SW = 101.
echo.py	Test program for wifi module
main.py	Test the DIP switch setting and execute the corresponding .py file.
mic.py	Microphone class library to acquire signals from microphone module.
motor.py	Driver class library for the motor driver chip TB6612 to drive motors.
mpu6050.py	IMU driver class library – to communicate with the accelerometer and gyroscope.
neopixel.py	Neopixel LED strip driver class library.
oled_938.py	OLED display driver class library.
plotspec.py	Plot spectrum of microphone signal on OLED in dB. Fs = 10kHz.
pybench_main.py	The controlling program for PyBench to interpret commands. Run if SW = 111.
pybench.py	The PyBench class library. Can be used in your own application programs later.
pybench_test.py	Self-test program for the PyBench board to verify the hardware. Run if SW = 110.
font.py	Character fonts used by oled_938.py.

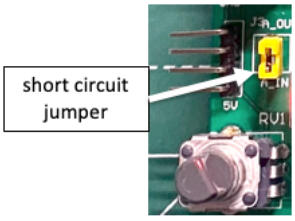
In addition, there are these DFT packages written by Peter Hinch to perform Fourier Transform:

Program	Purpose
dft.py	Fast DFT algorithm written in ARM assembly instructions.
dftclass.py	MicroPython class wrapper for dft.py.
polar.py	Complex coefficient to magnitude/phase translation.
window.py	Application of a window function to input signal.

Step 2: Testing the PyBench system is working properly

With the dual-in-package (DIP) switches set to "110", press the RESET button on the PYB (left button). You should see the OLED display the message "PyBench Self-Test". Now press the USR button on the PYB (right button) for a second and release to execute the microphone test. If you make some sound, the captured microphone signal will be shown on the OLED display.

Depress the USR button again to cycle through the other test in the following sequence:



1st time: Microphone signal test – show capture microphone signal in real-time.

2nd time: Accelerometer test – shown pitch and roll angles of PyBench as you tilt the PyBench board.

3rd time: Motor test – test the motor – **DO NOT DO THIS TEST AT PRESENT!!**

4th time: ADC/DAC test - With the short-circuit jumper installed, you can use the 10k ohm potentiometer (RV1) to change the frequency of the sinewave generated and shown on the display.

Step 3: Check that Matlab on your computer is talking to PyBench hardware

- Set PyBench switch setting to "111" (all up) and press RESET.
- Open Matlab and make sure that you are running version 2019b or later. Under the Command Window, run the command (without >>):

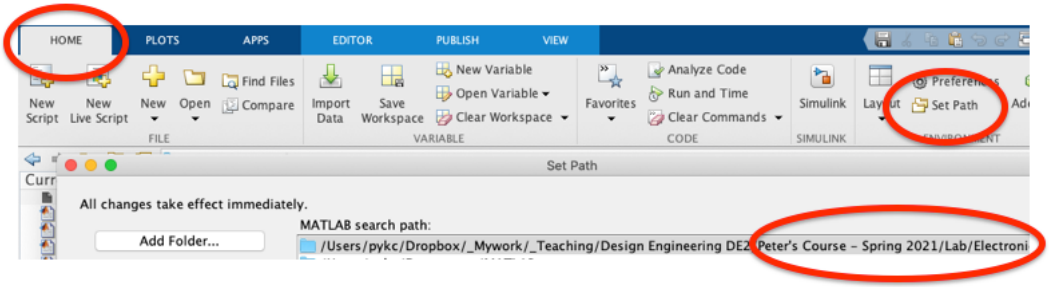
  >> serialportlist

  This shows all the serial ports on your computer. The last one on the list should be the port that communicates with PyBench.

  For PC: "COMx" (where x is a communication port number)

  For Mac: "/dev/tty.usbmodem…" (where … is not listed but is a very long number)

- Download from the course webpage the file "PyBench.m.zip" and unzip this in the "Electronics_2" folder you created in Lab 1 last week.
- Add "Electronics_2" and "Lab2" folders (or whatever folder you put your stuff) to Matlab's search path using Matlab's HOME -> Set Path menu.



- Test that Matlab is now able to use PyBench class library by entering the following commands into Matlab's Command window:

```
>> ports = serialportlist
>> pb = PyBench(ports(end))
```

The first command saves a list of serial ports to the variable "ports". The second command create a PyBench object "**pb**" as defined in the file "**PyBench.m**" which you have downloaded. It also specifies the last (i.e. end) item in "**ports**" to be the serial port corresponding to the USB connection. Matlab should report the properties of **pb** as shown here.

```
pb =

  PyBench with properties:

    BUFFERSIZE: 20000
     sig_freq: 10
         dc_v: 1.6500
        max_v: 3.3000
        min_v: 0
   duty_cycle: 50
    samp_freq: 100
```

You are now ready to proceed with this Lab session.

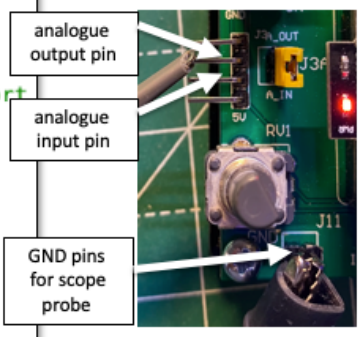
Task 2: Using PyBench to generate signals via Matlab and explore their spectra

PyBench is an electronic instrument designed to be controlled over the USB serial connection. After creating the PyBench object "pb", you can use the following **methods** to control the PyBench system.

Methods	Purpose
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.

- Using the Matlab's editor, create the file **lab2task2.m** Matlab script in the Lab2 folder. Execute the script by entering the name of the script (without .m) in the Command Window.

```
% Lab 2 - Task 2 - Signal generation and capture with PyBench
%
clear all
ports = serialportlist;      % find all serial port
pb = PyBench(ports(end));    % create a PyBench object with last port
% Set the various parameters
f = 440;                     % signal frequency
fs = 8000;                   % sampling frequency
pb = pb.set_sig_freq(f);
pb = pb.set_samp_freq(fs);
pb = pb.set_max_v(3.0);      % set maximum output voltage
pb = pb.set_min_v(0.5);      % set minimum output voltage
pb = pb.set_duty_cycle(50);
% Generate a signal
pb.sine();
```



- Now try changing various parameters such as signal frequency and voltages.
- Generate triangle and square waves using Pybench built-in methods.
- Add the following lines to **lab2task2.m** (**plot_spec.m** is a function created in Lab 1). Make sure that you understand the code.
- Explain the spectrum for the three different signals generated and check that the harmonic components are as you expected.

```
% Capture N samples
N = 1000;
samples = pb.get_block(N);
```

```
% 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);
```

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

Task 3 – Capture and analyse microphone signals

Create the following Matlab script **lab2task3.m** in the Lab 2 directory.

```
% Lab 2 - Task 3 - Capture and analyse microphone sound signal
%
clear all
ports = serialportlist;     % find all serial port
pb = PyBench(ports(end));   % create a PyBench object with last port

% Set sampling frequency
fs = 8000;
pb = pb.set_samp_freq(fs);

% Capture N samples
N = 1000;
samples = pb.get_mic(N);
data = samples - mean(samples);  % remove dc offset

% plot data
figure(1);
clf
plot(data);
xlabel('Sample no');
ylabel('Signal voltage (V)');
title('Microphone signal');

% find and plot spectrum
figure(2);
plot_spec(data, fs)
```

If you run **lab2task3** multiple times while whistling, you should see the effect of your whistle on the spectrum.

Whistling continuously is not only tiresome, it is also impossible to control the frequency precisely. Download one of many free "**tuning fork**" apps for your phone or use one of many online tuning fork to generate a tone at selected frequencies. You should now be able to capture and measure the frequency of the tone being generated using the PyBench board and Matlab without tiring your lips!

Now modify **lab2task3.m** to **lab2task3a.m** by adding the lines shown below so that you capture microphone data and display the spectrum **continuously** (only the spectrum and not the data. 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!

```
 %  repeat capture and plot spectrum
while true
    samples = pb.get_mic(N);
    data = samples - mean(samples);
    figure(2)
    clf;
    plot_spec(data,fs);
end
```

Warning: Running Matlab in an infinite loop may prevent you from re-gaining control over Matlab or even your computer. There are two things you may try if you want to get back control: 1) Type **CTRL+C** in the Command Window to interrupt Matlab; 2) kill the Matlab process and restart it again.

Now change the tuning fork frequency from 3000 Hz to 5000 Hz in steps of, say, 500Hz. Explain what you discover.

Now modify the number of data samples N captured, say from 1000 to 500 and 5000. What is the effect of N on the spectrum of the signal?

Task 4 – Windowing effect on a signal

Download from the course webpage my version of **plot_spec_dB.m**, which is a function (not a script) that plots the magnitude spectrum in decibel (dB) instead of voltage. Modify your previous program in to **lab2task4.m** to use **plot_spec_dB** function in place of **plot_spec**. This will display the amplitude spectrum in dB.

Displaying the magnitude spectrum in a logarithmic scale provides much higher 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 scaled to the peak spectral value. Finally, the magnitude axis is limited to 0dB to -60dB. Make sure you understand the code of **plot_spec_dB.m**.

Now obtain the spectrum for tuning fork sound at 1000Hz and at 1100Hz precisely. Observe the differences in the spectra of these two signals. What you see here is the result of extracting a portion of the signal to analyse. This effectively multiply the signal through **a rectangular window**. This phenomenon is known as the "**windowing**" effect.

Modify your program **lab2task4.m** to **lab2task4a.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 on the same figure for comparison. Explain the signal spectrum with and without Hamming Window.

NOTE: "`data.*window`" means that data vector and the window vector are multiplied **element-by-element** instead of performing inner product computation. The ".*" operator is element-by-element multiplication while "*" operator is matrix multiplication.

```
% 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
```

Task 5 – Music signal segmentation and analysis

Download from the course webpage, the file '**two_drums.wav**'. Create the file **lab2task5.m** to include the following code:

```
% Lab 2 - Task 4 - Analyse Two drum beats
%
clear all
[sig fs] = audioread('two_drums.wav');
sound(sig, fs)
% plot the signal
figure(1);
clf;
plot(sig);
xlabel('Sample no');
ylabel('Signal (v)');
title('Two Drums');
```

From now on, we will be analyzing musical signals stores on your computer as waveform files (.wav) instead of using the microphone signals from the PyBench board. Note that fs is the different 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 this task yourself, the solution is shown in the Appendix.

Task 6 – Analysing complex music

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

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 Task 5

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
% plot spectrum of energy
figure(3)
plot_spec(E - mean(E), 1/T);
```