Dyson School of Design Engineering

Imperial College London

DE2.3 Electronics 2

Lab Experiment 5: Team Project Session 1

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

Introduction

You have finished all formal laboratory sessions for DE2.3. From now, you should focus only on the Team Project during the lab sessions. This document is intended to offer you some guidance and hints as to how to go about the complex task of achieving the goals of Milestones 1 and 2. Before that, let us do one more exercise on filters.

Exercise 1: Moving Average filter

This exercise will be conducted using Matlab alone. The goal is for you to explore the low pass filtering effect of the moving average filter.

Download the music file: **bgs.wav** from the course webpage. This is a short segment of music from "Staying Alive".

Create the following Matlab script as **lab5_ex1.m**.

```
% Lab 5 - Ex 5a: 4-taps Moving average filter
%   inefficient way
clear all
[sig fs] = audioread('bgs.wav');
% Add noise to music
x = sig + 0.2*rand(size(sig));
% Plot the signal
figure(1);
clf;
plot(x);
xlabel('Sample no');
ylabel('Signal (v)');
title('Stay Alive Music');
% Filter music with moving average filter
N = size(x);
for i=4:N
    y(i) = (x(i)+x(i-1)+x(i-2)+x(i-3))/4;
end
y(1)=x(1)/4;
y(2)=(x(2)+x(1))/4;
y(3)=(x(3)+x(2)+x(1))/4;
% Play the original & then the filtered sound
sound(x, fs)
disp('Playing the original - press return when finished')
pause;
sound(y, fs)
disp('Playing the filter music')
```

When you compare the noise-corrupted music with the filtered version, you should notice a slight reduction in the noise.

Modify **lab5_ex1a.m** to **lab5_ex1b.m** so that you can use a variable number of taps. Change this to 10 and 20, and see how the filtered music sound. (Solution in Appendix A.)

Milestone 1

By the end of today, you should have put together your Segway with the stabilizer installed. You can design your own if you wish, but I also have some laser cut stabilizers for you to use.

You also need to write the controlling program in MicroPython to drive the Segway using the AdaFruit mobile App via Bluetooth.

Sampling microphone signal using interrupt

Although you are not expected to complete Milestone 2 until the end of February, you should read through the instruction below concerning how to sample and buffer N microphone signals under the control of a timer interrupt. The timer is programmed to issue an interrupt at an 8kHz sampling rate. That is, an interrupt is generated EVERY 125 μsec.

The code to do this is shown below and is stored in the file 'buffer.py', which you can download from the course webpage.

This program will store away N samples in a pre-allocated array '**s_buf[.]**' in the interrupt service routine '**isr_sampling**(.)'. Once set up, this will happen automatically in the background without further instruction from the main program loop in the foreground.

We also need a way to tell the main program loop that the buffer is full, i.e. N samples have been taken. To do this, we use a status flag (also known as a '**semaphore'**) to indicate that N samples have been taken.

The main program loop simply wait until the buffer is filled, and then it displays the buffer content on the OLED display the function '**plot_signal(.)**'.

Finally, we have created an output pin object on 'X5', which is connected to the top BNC connector (used as analogue output in earlier experiments). We then toggle this pin in the main program loop to indicate the time it takes for the main loop to go around the loop. The time it takes for the semaphore to go high is the half period (i.e. it toggles every N x 125 μsec).

If you now measure the signal on X5, you will see that it is much slower than expected. Why? Because driving the OLED display is slow! If you now comment out the OLED display statements, you will see that X5 signal is as expected.

You should find this buffer program useful when you write the code to synchronise the movement of the Segway to the beat of the music.

Milestone 2 (leading to 3)

For milestone 2, you should program the Segway to dance to music WITHOUT balancing (because the stabilizer will keep the Segway upright). Before you attempt Milestone 2, pick a piece of music that has a strong beat. Your Segway will dance to this music in real-time. Last year, all groups were told to use "Staying Alive" by the Bee Gees. This piece has very clear beat and is useful to test your algorithm. However, you are encouraged to choose your own.

Milestone 2 is a relatively large task, which can be broken down into the following sub-tasks:

1) Process the music signal offline using Matlab to derive the beat period (i.e. how long between beats and therefore the duration of each dancing step);
2) Determine the colour, mood, shape of the music through the spectrum of around a few second of the signal (in Matlab);

3) Map the spectrum to dancing steps (automatically if possible) and code this as ASCII text file to be transferred to the microSD card;
4) Write the MicroPython program on PyBench to synchronize the Segway's movement to the beat of music. There are a few possible approaches to this, some easy, and some much harder. This is discussed below. Once a beat is detected, flash one or more of the four LEDs. You can therefore test your music analysis and the dance routine WITHOUT anything moving!

For the synchronization task, you may consider the following methods:

a) Simply wait until you detect the first beat. Since you have worked out the exact beat period, and if 'Staying Alive' has a steady beat, you can use

```
tic = pyb.millis()
…..
elapsed_time = pyb.millis() – tic
```

to determine exactly when the next beat is. The disadvantage is that timing error will accumulated. However, you will probably be proximately right, particularly at the beginning.

b) You can find the ratio of instantaneous energy over a 20 msec window to the local average energy over 1 or 2 second moving window, and detect the start of the beat when this ratio exceeds some threshold.
c) To reduce the chance of false beat detection, you can look for the beat only after a certain period of time has elapsed since the last detect beat.

Appendix B: Solution to Excise 1b

```
% Lab 5 - Ex 5b: N-taps Moving average filter
%
clear all
[sig fs] = audioread('bgs.wav');
% Add noise to music
x = sig + 0.2*rand(size(sig));
% Plot the signal
figure(1);
clf;
plot(x);
xlabel('Sample no');
ylabel('Signal (v)');
title('Stay Alive Music');
% Filter music with moving average filter
N = size(x);
N_tap = 20
for i=N_tap:N
    temp = 0;
    for j = 0:N_tap-1
        temp = temp + x(i-j);
    end
    y(i) = temp/N_tap;
end
% Play the original
sound(x, fs)
disp('Playing the original - press return when finished')
pause;
sound(y, fs)
disp('Playing the filter music')
```

Appendix B – Code for "buffer.py"

```
import pyb
from pyb import Pin, Timer, ADC, DAC, LED
from array import array
from oled_938 import OLED_938   # Use OLED display driver

# I2C connected to Y9, Y10 (I2C bus 2) and Y11 is reset low active
oled = OLED_938(pinout={'sda': 'Y10', 'scl': 'Y9', 'res': 'Y8'}, height=64,
                    external_vcc=False, i2c_devid=61)
oled.poweron()
oled.init_display()
oled.draw_text(0,0, 'Lab 6 - Exercise 1')
oled.display()

# define ports for microphone in and trigger out
mic = ADC(Pin('Y11'))
trigger = Pin('X5',Pin.OUT_PP)

N = 160             # size of sample buffer
s_buf = array('H', 0 for i in range(N))  # reserve buffer memory
global ptr
global buffer_full
ptr = 0             # buffer index
buffer_full = False # semaphore - ISR communicate with main program
```

```
#  Function to plot data on OLED - used only for diagnosis
def plot_sig(signal,message):
    index = len(signal)
    if index >= 128:
        step = min(index,int(index/128))
    else:
        step = 1
    oled.clear()
    oled.draw_text(0,0,message)
    x = 127
    max_sig = max(max(signal),3000)
    min_sig = min(min(signal),1000)
    range_sig = max_sig - min_sig
    for i in range(0,index,step):
        y = 63 - int((signal[i] - min_sig)*63/range_sig)
        oled.set_pixel(x,y,True)
        x = x - 1
    oled.display()
```

```
# Interrupt service routine to fill sample buffer s_buf
def isr_sampling(dummy):    # timer interrupt at 8kHz
    global ptr              # need to make ptr visible in here
    global buffer_full  # need to make buffer_filled visible in here

    s_buf[ptr] = mic.read()     # take a sample every timer interrupt
    ptr += 1
    if (ptr == N):
        ptr = 0
        buffer_full = True

# Create timer interrupt - one every 1/8000 sec or 125 usec
speed_timer = pyb.Timer(7, freq=8000)
speed_timer.callback(isr_sampling)

while True:
    if buffer_full:     # semaphore signal from ISR to say buffer full
        plot_sig(s_buf,'Hello World!')
        if (trigger.value()):   # Measure how long it takes on X5 - top BNC
            trigger.low()
        else:
            trigger.high()
        buffer_filled = False
```