Dyson School of Design Engineering

Imperial College London

DE2.3 Electronics 2

Lab Experiment 3: IMU and OLED Display

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

Objectives

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

- Learned how to use accelerometer and gyroscope to derive orientation.
- Combine these readings to get reliable pit and roll angles.
- Control the PyBench board in Python.
- Use the OLED display to show messages and graphics.
- Learn to use the 5k ohm potentiometer as input.

Exercise 1: The Inertia Measurement Unit (IMU)

In this experiment, you will learn to use the accelerometer, then the gyroscope, to derive the pitch and roll angle of the PyBench board. The IMU we use is MPU6050 by InvenSense. When you are using Matlab with the PyBench class library (in **pybench.m**), you should use the Matlab functions:

```
[p, r] = pb.get_accel();        % p, r  = pitch & roll angle in radians
[x, y, z] = pb.get_gyro();      % x, y, z = rate of rotation in 3-axes in rad/sec
```

These functions return angles or angular velocity in radians or radians/sec, not in degrees!

- Create the following Matlab code as the file**: lab3_ex1a.m** and check that your IMU on your board works correctly. Make sure that you understand the Matlab code and how it works.
- Rotate the PyBench board along the horizontal axis towards you and away from you to change the pitch angle. (Which direction is positive?)
- Rotate the PyBench board along the vertical axis to your left and to your right. You are changing the roll angle. (Which direction is positive?)
- Now move to PyBench board forward and back (as if you are driving the board on a vehicle) while rotating. What happens?
- Remember to record your results and reflections in your electronic Logbook.

```
1  % Lab 3 - Exercise 1a: testing the accelerometer
2  clear all
3  close('all')
4  pb = PyBench('/dev/tty.usbmodem1422');
5  N = 500;        % each graph is 500 time points
6  end_time = 10.0; % initial guess of time axis range
7  while true
8      % Plot the axes first for plot later
9      figure(1)
10     clf(1)
11     axis([0 end_time -90 90]);
12     title('Accelerometer: Pitch & Roll Angles','FontSize', 16);
13     ylabel('Angles (deg)','FontSize', 14);
14     xlabel('Time (sec)','FontSize', 14);
15     grid on; hold on;
16     tic;
17     % read and plot accelerometer data
18     for i = 1:N
19         [p, r] = pb.get_accel();    % in radians
20         timestamp = toc;
21         pitch = p*180/pi;        % convert to degrees
22         roll = r*180/pi;
23         plot(timestamp, pitch,'.b');    % plot pitch in blue
24         plot(timestamp, roll,'.r');     % plot roll in red
25         pause(0.001);                % delay for 1 ms
26     end  % for loop
27     end_time = toc;     % use actual time range from now on
28 end  % while
```

Lines 4: Use 'COM4' for PCs.

Line 6: Define range of x-axis, 10 seconds for now. Update later.

Line 11: Fix axes scaling.

15: Overlay plotting on same axes scaling.

16: Define start time.

19: Get pitch & roll angles.

20: Get time point reading.

23, 24: Plot two points only.

25: Pause for 1ms, needed by Matlab plot function (not sure why).

27: update end_time.

Key understanding: Accelerometer provides roll and pitch angles through measuring the forces in the x and y directions due to gravity. If the accelerometer is also in motion, the movement adds extra forces to the sensor on top of the gravitational force. Therefore, the pitch and roll angles measured by the accelerometer on is "noisy" if there is motion. You should be able to detect this from the graph.

```
1   % Lab 3 - Exercise 1b: testing the gyroscope
2   clear all
3   close('all')
4   pb = PyBench('/dev/tty.usbmodem1422');
5   N = 500;              % each graph is 500 time points
6   end_time = 10.0;      % initial guess of time axis range
7   gx = 0; gy = 0; % initialise angles gy pitch, gx roll
8   while true
9       % Plot the axes first for plot later
10      figure(1)
11      clf(1)
12      axis([0 end_time -90 90]);
13      title('Gyroscope Pitch & Roll Angles','FontSize', 16);
14      ylabel('Angles (deg)','FontSize', 14);
15      xlabel('Time (sec)','FontSize', 14);
16      grid on; hold on;
17      timestamp = 0;
18      tic;
19      % read gyroscope data
20      for i = 1:N
21          [x, y, z] = pb.get_gyro();  % angular rate in rad/sec
22          dt = toc;         % get elapsed time
23          tic;
24          timestamp = timestamp + dt;
25          gx = max(min(gx+x*dt,pi/2),-pi/2);  % limit to +/- pi/2
26          gy = max(min(gy+y*dt,pi/2),-pi/2);
27          plot(timestamp, gy*180/pi,'.b');    % plot pitch in blue
28          plot(timestamp, gx*180/pi,'.r');    % plot roll in red
29          pause(0.001);          % delay for 1 ms, needed for plot
30      end  % for loop
31      end_time = timestamp;     % use actual time range from now on
32  end  % while
```

Now we will learn to use the gyroscope to derive the pitch and roll angles. Enter the following code as a new Matlab script: **lab3_ex1b.m**.

Deriving the angles from rate of change in angles (which is what the gyro gives us) is somewhat more involved. We need to perform: $\delta\theta = \dot{\theta} \times \delta t$ where $\dot{\theta}$ is the gyro reading and δt is the time increment since the last reading. The explanation for the code is as follows:

Line 7: Need to set gyro angles to zero for x and y. gy is pitch angle, gx is roll angle.

22 & 23: toc will now provide incremental time δt since last tic.

25 & 26: Accumulate gx and gy, and limit this to $\pm\pi/2$.

Key understanding: You can also derive the pitch and roll angle using the gyroscope readings. However, due to integration process (i.e. to get angle, you integrate or accumulate $\dot{\theta} \times \delta t$ over time) results in high drift because any errors in the reading get accumulated. So, accelerometer gives noisy readings, while gyroscope gives low noise readings, but introduce offsets over time.

Exercise 2: Visualization in 3D

Plotting graphs, while useful, does not provide a good way to visualize exactly what is going on with an object when you rotate it about two orthogonal axes. For that, one needs a 3D model.

```
1   % Lab 3 - Exercise 2: 3D display of roll and pitch angles
2   clear all
3   close('all')
4   pb = PyBench('/dev/tty.usbmodem1422');
5   model = IMU_3D();  % create the IMU 3D visualisation object
6   N = 50;
7   tic;
8   gx = 0; gy = 0;     % initialise gyro angles
9   fig1 = figure(1);
10  while true
11      for i = 1:N
12          [p, r] = pb.get_accel();
13          [x, y, z] = pb.get_gyro();
14          dt = toc;
15          tic;
16          pitch = p*180/pi;
17          roll = r*180/pi;
18          gx = max(min(gx+x*dt,pi/2),-pi/2);
19          gy = max(min(gy+y*dt,pi/2),-pi/2);
20          clf(fig1);
21          subplot(2,1,1);
22          model.draw(fig1, p, r, 'Accelerometer');
23          subplot(2,1,2);
24          model.draw(fig1, gy, gx, 'Gyroscope');
25          pause(0.0001);
26      end  % for
27  end  % while
```

I have written a Matlab class: **IMU_3D.m** (download from course webpage as zipped file) which displays the IMU module (and the PyBench board) as a 3-D object. Using this for visualization is much nicer! The code is as shown below. Try and test this yourself.

Line 5: Create a 3D object "model" using the IMU_3D class library.

9: Need to create a figure object fig1 used by IMU_3D.

22, 24: Draw the 3D object at the specified pitch and roll angles, and the specified title for the plots.

Explore what happens when you rotate the PyBench board on the x and y axes, and when you shake the board forward and backward.

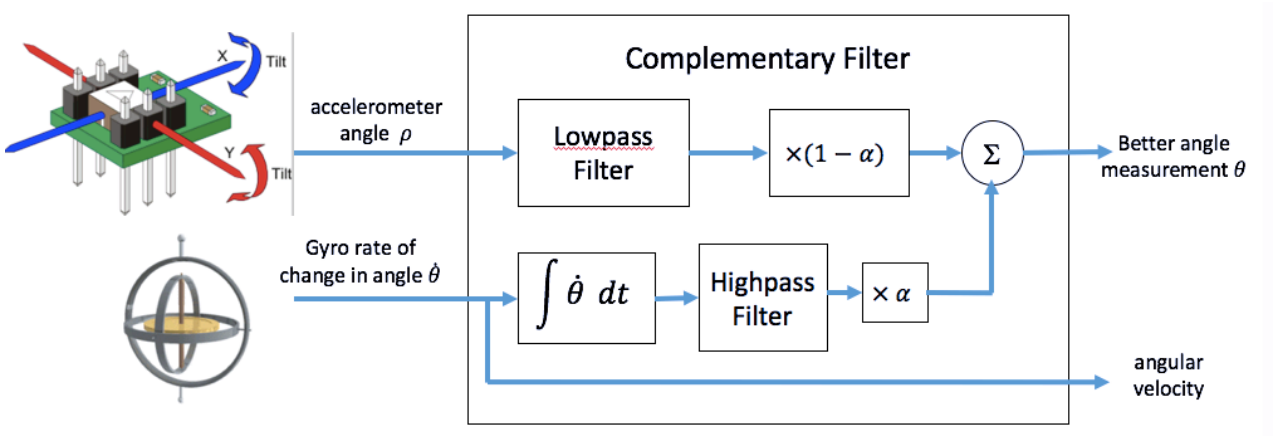
Exercise 3: Combining the two measurements using Complementary Filter

I hope you are now convinced that accelerometer provides angle measurements that are noisy but with no offset (or time dependent drift). Gyroscope provides angle measurements through numeric integration (or summation) that is not noisy, but suffers from error accumulation and therefore time dependent drift. So, how can we combine these two sets of readings together to give us better measurements?

There is an excellent article by Shane Colton entitled "*The Balance Filter*" (see course webpage) that explains how to do this easily. The basic idea can be understood by referring to the conceptual diagram below.

Since the accelerometer angles are "noisy", we can reduce the noise using a **lowpass filter**, which has a smoothing effect. (This is just like our familiar RC circuit which passes low frequencies, but suppresses high frequency components).

The gyro readings are rate of change in angles. We use numerical integration to estimate the angles. However, this produces in dc offset error. To mitigate this error, we pass process the gyro data through a highpass filter to remove (or reduce) the dc error.



The system that processes the accelerometer and gyroscope readings to yield an angle is known as **Complementary Filter**. It can be implemented using the following mathematical formula:

$$\text{angle } \theta_{new} = \alpha \times \left(\theta_{old} + \dot{\theta}\, dt \right) + (1-\alpha) \times \rho$$

where
- α = scaling factor chosen by users and is typically between 0.7 and 0.98
- ρ = accelerometer angle
- θ_{new} = new output angle
- θ_{old} = previous output angle
- $\dot{\theta}$ = gyroscope reading of the rate of change in angle
- dt = time interval between gyro readings

Modify the code from Exercise 2 (as **lab3_ex3.m**) to compute the complementary filtered pitch and roll angles. Add a third plot (in addition to the previous two 3D plots) to show how the combined pitch and roll angles manifest themselves with the 3D model of the IMU.

If this is taking you too long to do independently yourself, a "model answer" is given in the Appendix.

Make sure that you fully understand what is happening. I will also go through the theory and the Matlab code during a tutorial session.

Exercise 4: Using the OLED driver on the PyBench Board

So far, you have been using the PyBench board via Matlab. For the rest of this experiment, you will switch to using MicroPython running on the Pyboard, as you did last year in the DE1.3 Electronics 1 module.

Put the 2-way DIP Switch setting to '11', and reset the Pyboard (pressing the left button on the Board). In this setting, the Pyboard will be running the program "**user.py**" stored on the SD Card. Create user.py that contains only one line: **execfile('lab3_ex4.py')**

It will search in the SD card for the Python program **lab3_ex4.py**, and execute this. In this way, you can easily run other Python program by simply replacing the filename in between the quotations with any user program.

Run a terminal program on your PC or Macbook to talk to the Pyboard as you did last year. (If you are stuck, ask one of the GTA to help). On the PC, you need to run "putty". On a Mac, you can run OSX's **terminal** program and enter the command **screen /dev/tty.usbmodem* .**

You should see the Python REPL: >>>.

Use an editor to create the file: **lab3_ex4.py** on the SD Card with the following code.

Line 1: Specify a block of comments.

10: Use OLED driver.

```
1  '''
2  ------------------------------------------------------
3  Name: Lab 3 Exercise 4
4  ------------------------------------------------------
5  Learning to use the OLED display driver
6  ------------------------------------------------------
7  '''
8  import pyb                      # Pyboard basic  library
9  from pyb import LED, ADC, Pin   # Use various class libraries in pyb
10 from oled_938 import OLED_938   # Use OLED display driver
11
12 # Create peripheral objects
13 b_LED = LED(4)                  # blue LED
14 pot = ADC(Pin('X11'))           # 5k ohm potentiometer to ADC input on pin X11
15
16 # I2C connected to Y9, Y10 (I2C bus 2) and Y11 is reset low active
17 oled = OLED_938(pinout={'sda': 'Y10', 'scl': 'Y9', 'res': 'Y8'}, height=64,
18                  external_vcc=False, i2c_devid=61)
19 oled.poweron()
20 oled.init_display()
21
22 #  Simple Hello world message
23 oled.draw_text(0,0,'Hello World!')   # each character is 6x8 pixels
24
25 tic = pyb.millis()          # store start time
26 while True:
27     b_LED.toggle()
28     toc = pyb.millis()      # read elapsed time
29     oled.draw_text(0,20,'Delay time:{:6.3f}sec'.format((toc-tic)*0.001))
30     oled.draw_text(0,40,'POT5K reading:{:5d}'.format(pot.read()))
31     tic = pyb.millis()      # start time
32     oled.display()
33     delay = pyb.rng()%1000  # Generate random number btw 0 and 999
34     pyb.delay(delay)        # delay in milliseconds
35
```

14: 5K potentiometer connected to pin X11.

17: Create OLED display object. 128 x 64 pixels.

19: Turn display ON

20: Initialise display

23: Draw hello world. (0, 0) is top left corner. Each character is 6 x 8 pixels including space.

25: Record start time in msec.

28: Record time up to now.

33: Generate a random number and force this to the range 0 to 999.

This program teaches you various aspects of MicroPython and the OLED driver that will be useful to you throughout the rest of the module. The OLED driver, which is in the file: **oled_938.py** on the SD card, has the following methods (shaded ones are used in this exercise):

Method	Description
oled.clear( )	Clear display (i.e. blank)
oled.display( )	Update display with content of buffer. Must call after drawing to see effect
oled.set_pixel(x,y,c)	Turn pixel (x,y) ON (c=1) or OFF (c=0). (0, 0) is top RIGHT corner.
oled.init_display( )	Initialise display. Call once at the start.
oled.poweron( )	Turn on the power to the display. Call once at the start.
oled.draw_text(x,y,s)	Draw the text string s at (x, y). (0,0) is top LEFT corner.
oled.draw_circle(x,y,r,c)	Draw a circle of radius r (in pixels), colour c (ON=1, OFF=0)
oled.draw_square(x,y,s,c)	Draw a square of side s and colour c.
oled.draw_line(xa,ya,xb,yb,c)	Draw a line from (xa, ya) to (xb, yb) in colour c.
oled.line(x,y,phi,d,c)	Draw a line of length d from (x,y) at angle phi in degrees in colour c. Phi is the angle relative to

This program does the following:

1. Create a random number between 0 and 999 using Pyboard's hardware random number generator (with **pyb.rng()**). This is a 30-bit integer – very big! The '%' or modulo operator limits the variable "delay" to be within the desired range. (modulo operates give you the remainder.)
2. Use **pyb.millis**( ) to record time in millisecond as **tic** and **toc** (as in Matlab). In this way, we can measure elapsed time (from tic to toc).
3. Measure the voltage at the potentiometer, which is connected to pin 'X11'. The voltage range is 0 to 3.3V (as usual for this board). This voltage is converted by the ADC on the ARM chip and produces a 12-bit result from 0 to 4095.
4. Use the "**oled**.**draw_text(.)**" method to write text to the OLED display. Lines 29 and 30 shows you how to use the **.format(v)** method to create a string including a formatted variable v – very useful later.

Run and test this program. **Make sure that you understand the code**.

Now modify the program so that "Hello World!" is at the centre of the display. (Remember that each character is 6 x 8 pixels including a 1-pixel gap on the right and bottom.)

Additional Challenge (Optional):

In addition to drawing text on the OLED display, you can also draw lines or any arbitrary figure on the display. Use oled.**draw_line(.)** method to draw a "pendulum" whose angle (relative to vertical) is determined by the reading from the 5k ohm potentiometer. If the reading is 0, the pendulum should be +90 degrees (to the right of vertical) and if the reading is 4095, the pendulum should be -90 degrees.

Exercise 5: Using the IMU driver on PyBench Board

An IMU driver (file: **mpu6050.py** on the SD cared), written in MicroPython, is available to read information from the IMU. Before using the IMU, you need to create the IMU object with:

```
imu = MPU6050(1, False)
```

Thereafter, you can use the following methods to read accelerometer and gyroscope data.

Method	Description
pitch = imu.pitch( )	Returns the pitch angle in degrees.
roll = imu.roll( )	Returns the roll angle in degrees.
gy_dot = imu.get_gy( )	Returns d(pitch)/dt in degrees/sec.
gx_dot = imu.get_gx( )	Returns d(roll)/dt in degrees/sec.
Imu.get_acc( )	Returns accelerometer angles in x, y, z directions in radians.
Imu.get_gyro( )	Returns gyroscope reading in x, y, z directions in radians/sec.

Write a Python program **lab3_ex5.py** that reads the pitch angle and the rate of pitch (gy_dot), and display these values on the OLED display as text in an infinite loop.

Additional Challenge (Optional):

Use the oled.**draw_line** (.) method, draw two separate pendulum lines, one for the pitch angle measured using the accelerometer and a second one for the pitch angle derived by the gyroscope.

Further Challenge (Optional):

Use complementary filter and combine the two readings (accelerometer and gyroscope). Plot the pendulum lines, one showing accelerometer pitch angle, and another showing the filtered pitch angle.

Appendix A: Model Answers for lab3_ex3.m and lab3_ex5.py

```
% Lab 3 - Exercise 3: 3D display of effects of Complementary Filter
clear all
close('all')
pb = PyBench('/dev/tty.usbmodem1422');
model = IMU_3D();
N = 50;
gx = 0; gy = 0;             % gyro initial angles
angle_x = 0; angle_y = 0;   % combined angle using filter
alpha = 0.7; beta = 1-alpha; % weighting factor
tic
while true
    for i = 1:N
        [p, r] = pb.get_accel();
        [x, y, z] = pb.get_gyro();
        dt = toc;
        tic;
        % integration for gyro angles
        gx = max(min(gx+x*dt,pi/2),-pi/2);
        gy = max(min(gy+y*dt,pi/2),-pi/2);

        % complementary filtered angles
        angle_x = alpha*(angle_x + x*dt) + beta*r;
        angle_y = alpha*(angle_y + y*dt) + beta*p;

        figure(fig1)
        clf(fig1);
        subplot(3,1,1);
        model.draw(fig1, p, r, 'Accelerometer');
        subplot(3,1,2);
        model.draw(fig1, gy, gx, 'Gyroscope');
        subplot(3,1,3);
        model.draw(fig1, angle_y, angle_x, 'Filtered');
        pause(0.0001);
    end  % for
end  % while
```

```
'''
-------------------------------------------------------
Name: User's own program for Lab 4 Exercise 5
Creator:  Peter Cheung
Date:   8 Feb 2017
Revision: 1.0
-------------------------------------------------------
Draw pendulum reflecting pitch angles (raw and filtered)
-------------------------------------------------------
'''
import pyb
from pyb import LED
from oled_938 import OLED_938
from mpu6050 import MPU6050

# Define LEDs
b_LED = LED(4)

# 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()

# IMU connected to X9 and X10
imu = MPU6050(1, False)     # Use I2C port 1 on Pyboard

def read_imu(dt):
    global g_pitch
    alpha = 0.7    # larger = longer time constant
    pitch = int(imu.pitch())
    roll = int(imu.roll())
    g_pitch = alpha*(g_pitch + imu.get_gy()*dt*0.001) + (1-alpha)*pitch
    # show graphics
    oled.clear()
    oled.line(96, 26, pitch, 24, 1)
    oled.line(32, 26, g_pitch, 24, 1)
    oled.draw_text(0,0," Raw | PITCH |")
    oled.draw_text(83,0, "filtered")
    oled.display()

g_pitch = 0
tic = pyb.millis()
while True:
    b_LED.toggle()
    toc = pyb.millis()
    read_imu(toc-tic)
    tic = pyb.millis()
```