xDyson School of Design Engineering

Imperial College London

DE2 Electronics 2

Lab Experiment 4: 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 without Matlab.
- Use the OLED display to show messages and graphics.
- Learn to use the 5k ohm potentiometer as input.

Before you start this Lab, download from the course webpage lab4.zip and put these files in the Lab 4 folder. You will need these Matlab scripts written specifically for this Lab. Make sure that the Lab 4 folder is your current Matlab folder. I also provide solutions to Lab 4 in case you are stuck. However, I strong recommend that you TYPE IN the Matlab code yourself so that you have a chance to thing about each line of the code.

Task 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 velocities in radians or radians/sec, not in degrees!

- Create the following Matlab code as the file**: lab4task1a.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. (Effectively rocking it side-to-side.) You are changing the roll angle. (Which direction is positive?)
- Now slide 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 4 - Task 1a: testing the accelerometer
2   clear all
3   close('all')
4   ports = serialportlist;
5   pb = PyBench(ports(end));  % create a PyBench object
6   N = 500;        % each graph is 500 time points
7   end_time = 10.0; % initial guess of time axis range
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('Accelerometer: Pitch & Roll Angles','FontSize', 16);
14      ylabel('Angles (deg)','FontSize', 14);
15      xlabel('Time (sec)','FontSize', 14);
16      grid on; hold on;
17      tic;
18      % read and plot accelerometer data
19      for i = 1:N
20          [p, r] = pb.get_accel();    % in radians
21          timestamp = toc;
22          pitch = p*180/pi;        % convert to degrees
23          roll = r*180/pi;
24          plot(timestamp, pitch,'.b');    % plot pitch in blue
25          plot(timestamp, roll,'.r');     % plot roll in red
26          pause(0.001);                % delay for 1 ms
27      end  % for loop
28      end_time = toc;     % use actual time range from now on
29  end  % while
```

Lines 5: Use 'COMx' for PCs if needed.

Line 7: Define range of x-axis, 10 seconds for now. Update later.

Line 12: Fix axis scaling between ±90.

16: Overlay plotting on same axes scaling.

17: Define start time.

20: Get pitch & roll angles.

21: Get time point reading.

24, 25: Plot two points only.

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

28: update end_time.

NOTE: This Matlab script runs in an infinite loop due to the "while true" statement. To stop Matlab running to gain back control, you can type CTRL-C in the Command Window of Matlab.

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 is "noisy" if there is motion. You should be able to detect this from the graph.

Now we will learn to use the gyroscope to derive the pitch and roll angles. Enter the following code as a new Matlab script: **lab4task1b.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 this calculation: $\delta\theta = \dot{\theta} \times \delta t$ where $\dot{\theta}$ is the gyro reading and δt is the time increment since the last reading.
The explanation for the code, in addition to that from Task 1a, is as follows:

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

Line 8: Initialize gyro angles to zero for x and y. gy is pitch angle, gx is roll angle.

22: take gyro reading in rad/sec on all axes.

23 & 24: toc will now provide incremental time δt since last tic.

26 & 27: Accumulate gx and gy, and limit this to $\pm\pi/2$. These two lines teaches you how to perform integration through summation in a processor.

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 introduces offset that increases over time.

Task 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.

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. Further, you will need to install **the Aerospace Toolbox** for Matlab. To do that, you go to Matlab HOME tab, and click the **Add-Ons** ICON and follow the instructions.


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

Enter the Matlab code is as shown below as **lab4task2.m**. Try and test this yourself.

Line 6: Create a 3D object "model" using the IMU_3D class library.

10: Create a figure object fig1 used by IMU_3D.

23, 25: 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.

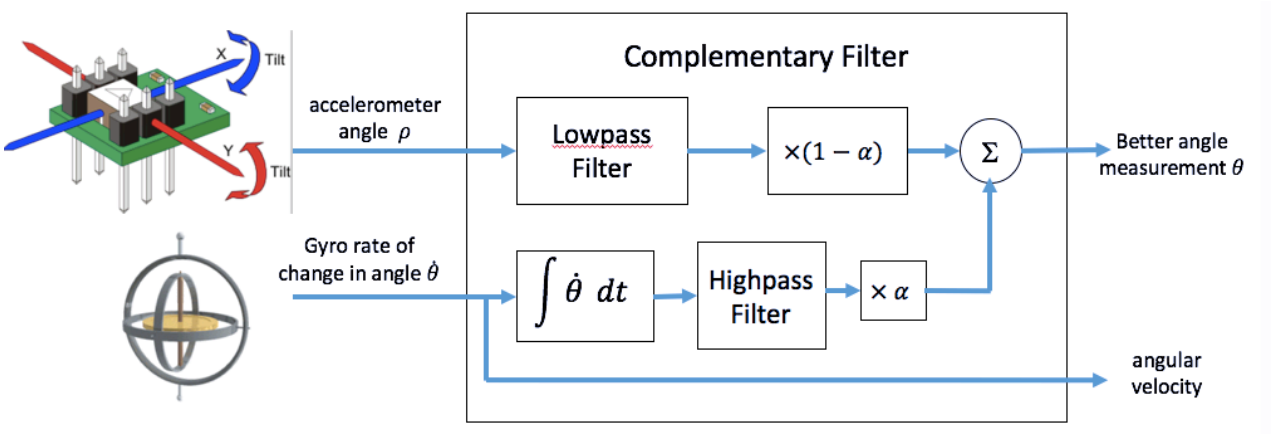
Task 3: Combining the two measurements using Complementary Filter

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). Exactly WHY this implements a lowpass filter will be explained in a later lecture.

The gyro readings are rates of change in angle. We use numerical integration to estimate the angles. However, this produces dc offset error as drift. To mitigate this error, we pass the gyro data through a highpass filter to 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 Task 2 (as **lab4task3.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, the solution is given in the Appendix.

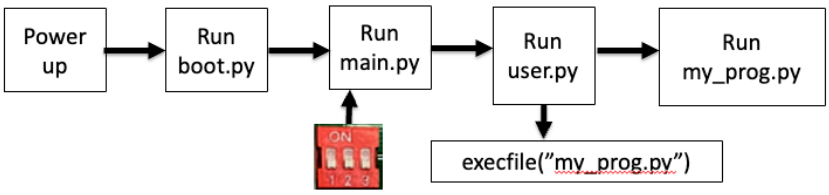
Make sure that you fully understand what is happening. I will also go through the theory and the Matlab code during a tutorial session.

How to run PyBench Board without Matlab or Laptop?

So far, you have been using the PyBench board via Matlab. For the rest of this experiment, you will switch to using MicroPython (uPy) running on the Microcontroller module (called the Pyboard, which is NOT the Raspberry Pi, but one that runs MicroPython after power is applied). In this task, you will start writing your own uPy programs that run on PyBench without tethering to your laptop. Instead, you will be creating a stand-alone system that can run on battery.

Put the 3-way DIP Switch setting to '000', 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. On booting up, PyBench will run the program **boot.py**, which will immediately execute the file **main.py**. "**main.py**" will check the configuration switches. Since it is set to '000', it will execute the file **user.py**. For now, **user.py** is "empty" in that it only contains a message. If you want to run your own uPy script, say, **my_prog.py**, you are recommended to create **my_prog.py** independently using a suitable editor and then modify user.py to include the MicroPython specific function: **execfile("my_prog.py")**.

These steps are depicted in the diagram below.



You are strongly advised to use VSCode as your uPy editor. If you do not have VSCode installed, you must first install this on your laptop. Assuming that you have **my_prog.py** created, and **user.py** modified as explained above, you can run **my_prog.py** by:

1. Make sure that the Pybench board is connected to your laptop using a USB cable. This provides both power to the hardware and allows communication between your laptop and hardware.
2. Set the DIP switches to 000 and press RESET button (left). This forces the Pyboard to do through the steps shown above.
3. Create your program using VSCode, either on your laptop disk or directly on the MicroSD card in the Pybench board. If you edit your uPy script on your computer, remember to copy it to the MicroSD card. If you edit your script directly on the SD card, remember to make a backup copy on your computer.

Running MicroPython via a terminal program

This part is different depending on whether you are using a Windows PC or a MacBook.

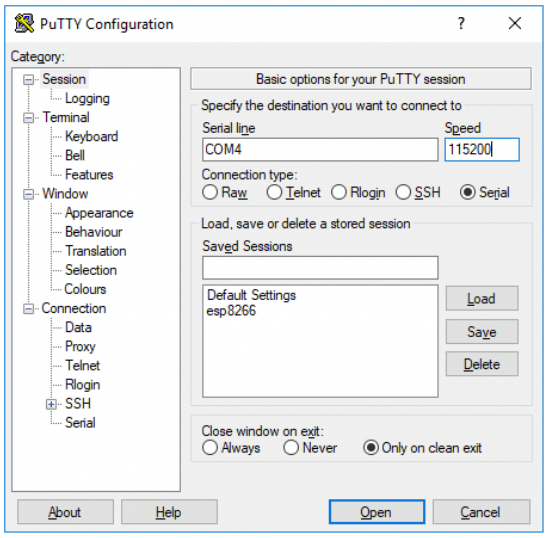
For MacBook Users

- Open the Terminal App and enter:

  screen /dev/tty.usb* 115200
- You should now be communicating with uPy and you should see a message on the terminal screen.
- Alternative, you may also start the terminal from VSCode.

For Windows PC Users

- Your Pybench hardware will appear as a USB disk and as device connect to one of the COM port as COMx, where x is a number which may change each time you connect.
- You can find out which COM port you should use by examining the Device Manager. You should see this under Ports. If you don't know how to do this, ask one of the GTAs.
- Install PuTTY program allows you to create a terminal to the USB port. The link to install this is: https://the.earth.li/~sgtatham/putty/latest/w64/putty-64bit-0.76-installer.msi . You can find this on the course webpage.
- Run PuTTY and select serial port as shown above (as an example).



Using MicroPython

When you are communicating with uPy using via either Terminal or PuTTY, you are controlling the Pybench hardware directly via the computer keyboard. Here are two most important keys:

CTRL-C: This interrupts the processor and returns the user to the Python REPL >>>. Now you can enter any uPy commands.

CTRL-D: Soft RESET – this forces the processor to go through the sequence as if you turned the power OFF and then ON.

IMPORTANT: The RED LED on the Pyboard will blink during SD Card read or write. If you pull the USB cable off when the computer is accessing the SD Card, you can corrupt its contents. The only way to recover would be to download all your files to the SD Card again.

Task 4: Using the OLED driver on the PyBench Board

Create **user.py** that contains only one line: **execfile("lab4task4.py")**. This statement searches in the SD card for the uPy program **lab4task4.py**, and execute this. In this way, you can easily run other uPy program by simply replacing the filename in between the quotations with any user program. **execfile()** is MicroPython specific function not found in standard Python 3.

Use an editor to create the file: **lab4task.py** on the SD Card with the following code.

```
2  ------------------------------------------------------
3  Name: Lab 4 Task 4
4  ------------------------------------------------------
5  Learning to use the OLED display driver
6
7  Creator:    Peter YK Cheung
8  Date:       9 Feb 2021
9  Revision:   1.1
10 ------------------------------------------------------
11 '''
12 import pyb                      # Pyboard basic  library
13 from pyb import LED, ADC, Pin   # Use various class libraries in pyb
14 from oled_938 import OLED_938   # Use OLED display driver
15
16 # Create peripheral objects
17 b_LED = LED(4)                  # blue LED
18 pot = ADC(Pin('X11'))           # 5k ohm potentiometer to ADC input on pin X11
19
20 # I2C connected to Y9, Y10 (I2C bus 2) and Y11 is reset low active
21 i2c = pyb.I2C(2, pyb.I2C.MASTER)
22 devid = i2c.scan()              # find the I2C device number
23 oled = OLED_938(
24     pinout={"sda": "Y10", "scl": "Y9", "res": "Y8"},
25     height=64,
26     external_vcc=False,
27     i2c_devid=i2c.scan()[0],
28 )
29 oled.poweron()
30 oled.init_display()
31
32 #  Simple Hello world message
33 oled.draw_text(0,0,'Hello World!')   # each character is 6x8 pixels
34
35 tic = pyb.millis()          # store start time
36 while True:
37     b_LED.toggle()
38     toc = pyb.millis()      # read elapsed time
39     oled.draw_text(0,20,'Delay time:{:6.3f}sec'.format((toc-tic)*0.001))
40     oled.draw_text(0,40,'POT5K reading:{:5d}'.format(pot.read()))
41     tic = pyb.millis()      # start time
42     oled.display()
43     delay = pyb.rng()%1000  # Generate random number btw 0 and 999
44     pyb.delay(delay)        # delay in milliseconds
```

Line 1: Specify a block of comments.

10: Use OLED driver.

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.

Task 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 **lab4task5.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. This is the same as the second self-test provided to you when you have the setting at 110.

Appendix A: Model Answers for lab4task3.m

```
% Lab 4 - Task 3: 3D display of effects of Complementary Filter
clear all
close('all')
ports = serialportlist;
pb = PyBench(ports(end));  % create a PyBench object
model = IMU_3D();
N = 50;
fig1 = figure(1);
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(0.1);
    end  % for
end  % while
```

Appendix B: Model answer for lab4task5.py

```
3  Name: Lab 4 Task 5 - display pitch angle on OLED
4  Creator:    Peter YK Cheung
5  Date:       5 Feb 2022
6  Revision:   2
7  ------------------------------------------------
8  Put comments here to explain what this does
9  ------------------------------------------------
10 '''
11 import pyb
12 from pyb import LED
13 from oled_938 import OLED_938
14 from mpu6050 import MPU6050
15
16 # Define LEDs
17 b_LED = LED(4)
18
19 # I2C connected to Y9, Y10 (I2C bus 2) and Y11 is reset low active
20 i2c = pyb.I2C(2, pyb.I2C.MASTER)
21 devid = i2c.scan()              # find the I2C device number
22 oled = OLED_938(
23     pinout={"sda": "Y10", "scl": "Y9", "res": "Y8"},
24     height=64, external_vcc=False, i2c_devid=i2c.scan()[0],
25 )
26 oled.poweron()
27 oled.init_display()
28
29 # IMU connected to X9 and X10
30 imu = MPU6050(1, False)     # Use I2C port 1 on Pyboard
31
32 def read_imu(dt):
33     global g_pitch
34     alpha = 0.7    # larger = longer time constant
35     pitch = int(imu.pitch())
36     roll = int(imu.roll())
37     g_pitch = alpha*(g_pitch + imu.get_gy()*dt*0.001) + (1-alpha)*pitch
38     # show graphics
39     oled.clear()
40     oled.line(96, 26, pitch, 24, 1)
41     oled.line(32, 26, g_pitch, 24, 1)
42     oled.draw_text(0,0," Raw | PITCH |")
43     oled.draw_text(83,0, "filtered")
44     oled.display()
45
46 g_pitch = 0
47 tic = pyb.millis()
48 while True:
49     b_LED.toggle()
50     toc = pyb.millis()
51     read_imu(toc-tic)
52     tic = pyb.millis()
```