Dyson School of Design Engineering

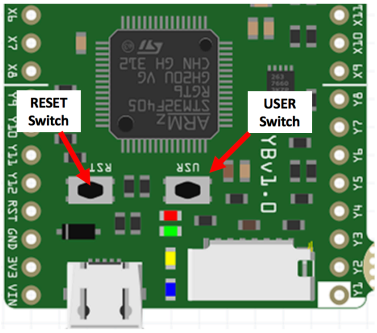
Imperial College London

DE2.3 Electronics 2

Lab Experiment 8: Putting Everything Together

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

INTRODUCTION



This lab instruction is intended to help you to put everything that you have learned together to complete the Team project. It contains various tips and guidelines, snippets of Python code, pseudo-code of the top-level program, related to the different tasks that your team is expected to achieve.

TIPS 1: FROM IDLE TO RUNNING

Running an embedded programme on the PyBench board can cause problem because as soon as you press the RESET button, you usually start the programme (which programme you run depends on the DIP switch setting). It is often helpful if the Segway is in an idle mode, and only run your programme when you press the USER button.

Here we use the OLED display to tell us what the Segway is doing. Remember that drawing anything on the OLED is VERY SLOW. Therefore, only do that outside the main program loop. We also include the print statements to help debugging. These will show up on the Putty or the Terminal window on your laptop.

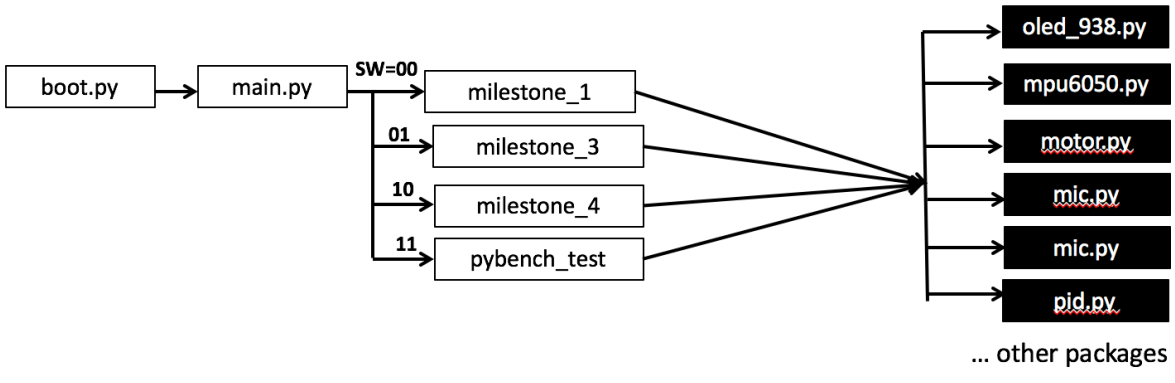
```
# Use OLED to say what Segway is doing
oled = OLED_938(pinout={'sda': 'Y10', 'scl': 'Y9', 'res': 'Y8'},
        height=64, external_vcc=False, i2c_devid=61)
oled.poweron()
oled.init_display()
oled.draw_text(0,0, 'Group xx')
oled.draw_text(0,10, 'Milestone 1: Driving')
oled.draw_text(0,20, 'Press USR button')
oled.display()

print('Performing Milestone 1')
print('Waiting for button press')
trigger = pyb.Switch()  # create trigger switch object
while not trigger():    # wait for trigger pressed
    time.sleep(0.001)
while trigger(): pass   # wait for release
print('Button pressed - Running')

# ....   rest of program
```

TIPS 2: MODIFY THE MAIN.PY FILE

So far, the way I have organized the Pybench system on the Segway is to run Pybench driver **pybench_main.py**, with the DIP switches are set to 00, run a self-test program if SW=01 or 10, and finally run user.py if SW=11. To make things easier for the team project, I recommend that you modify the program **main.py** so that you run the various milestones depending on the DIP switch setting as shown below.



TIPS 3: PSEUDO-CODE FOR MILESTONE 3

I assume that by now, you should have completed milestones 1 and 2. For milestone 3, your goal is to use your improved beat-detection routine to synchronous music with dancing moves with the stabilizer (i.e. no self-balancing). You should store the dance move in a text file (ASCII format), read this file at the start of the program and store the moves in an array BEFORE the main program loop. This is because reading ASCII characters from a text file is very slow. It is far better to store information in memory (i.e. an array) and access this array inside the program loop.

Furthermore, if you have not learned Python Keywords **Try** and **Finally**, you should. Here is the pseudo-code for Milestone 3.

```
#  Import relevant packages that you use ...
import pyb
from pyb import Pin, Timer, ADC, DAC, LED
import micropython   # Needed for interrupt
micropython.alloc_emergency_exception_buf(100)
........

#  Initialise various peripherals e.g. OLED, IMU etc

#  Initialise different constants, variables, arrays etc

#  Read the dancing steps from file into array

#  Wait for USR switch pressed

tic = pyb.millis()          # mark time now in msec

try:            # Try to handle exception
    while True: # always have infinite loop
        if buffer is full, detect beat
        if beat detected, do next dance move

finally:        # always executed if exception
    motor.A_stop()
    motor.B_stop()
```

TIPS 4: PITCH ANGLE ESTIMATION

By now you should be familiar with using the IMU to estimate the pitch angle using Complementary Filter. Here is a function that estimate the pitch angle, just to help you along a bit quicker. **dt** is delta time, the time since the last reading in the program loop. You find **dt** with **tic** and **pyb.millis()** or **pyb.micros()**. Don't forget to adjust **dt** to seconds in your equation.

```
# Pitch angle calculation using complementary filter
def pitch_estimate(pitch, dt, alpha):
    theta = imu.pitch()
    pitch_dot = imu.get_gy()
    pitch = alpha*(pitch + pitch_dot*dt) + (1-alpha)*theta
    return (pitch, pitch_dot)
```

TIPS 5: PID CONTROLLER

The basic PID controller equation is straigthforward:

$$w(t) = K_p e(t) + K_d \dot{e}(t) + K_i \int e(\tau) d\tau$$

I recommend you to create a PID controller function (or if you are good in Python coding, create a class) that do the following:

Input to function: pitch angle, (rate of change of pitch (pitch_dot), target (or set-point), cumulative pitch error (integral term).

Output of the function: PWM drive value limited to ±100.

You should also limit the set-point to a small value, such as ±3 degrees (say).

Finally, you will find that the IMU may NOT return a pitch angle of zero when it is upright. This is because the centre of gravity of the Segway may not be dead centre. This depends on the position of the battery and other factors. You therefore may need to take this into account.

The pseudo-code for the controller function is:

```
#  Calculate the pitch error pError

#  Computer the output W as sum of P, I and D terms

#  Accumulate the pError term for integration

#  Limit the output W to ±100

#  Return the W drive value
```

TIPS 6: TUNING THE PID CONTROLLER

Finally, you would need to tune the PID controller. One way to do this is to modify the program and change the three gain values: Kp, Ki and Kd in the Python code. A much better way to do this (thanks to Ben Greenberg last year) is the use the potentiometer on the Pybench board together with the USR switch to adjust these gain values live! Here is the code to do that:

```
trigger = pyb.Switch()
scale = 2.0
while not trigger(): # wait to tune Kp
    time.sleep(0.001)
    K_p = pot.read() * scale / 4095 # use pot to set Kp
    oled.draw_text(0, 30, 'Kp = {:5.2f}'.format(K_p)) # display live value on oled
    oled.display()
while trigger(): pass # wait for button release
while not trigger(): # wait to tune Ki
    time.sleep(0.001)
    K_i = pot.read() * scale / 4095 # use pot to set Ki
    oled.draw_text(0, 40, 'Ki = {:5.2f}'.format(K_i)) # display live value on oled
    oled.display()
while trigger(): pass # wait for button release
while not trigger(): # wait to tune Kd
    time.sleep(0.001)
    K_d = pot.read() * scale / 4095 # use pot to set Kd
    oled.draw_text(0, 50, 'Kd = {:5.2f}'.format(K_d)) # display live value on oled
    oled.display()
while trigger(): pass # wait for button release

print('Button pressed. Running script.')
oled.draw_text(0, 20, 'Button pressed. Running.')
oled.display()

... the rest of the program ....
```

Once you have tuned the PID controller, you can replace the **pot.read()** statement with the tuned gain value. Then to start the program, you just press the USR switch few times and the self-balancing program will run.

TIPS 7: PSEUDO-CODE FOR SELF-BALANCING, DANCING SEGWAY

Now put all these together, here is the pseudo-code of a possible final program to do both self-balancing and dancing to music:

```
#  Import relevant packages that you use ...
import pyb
from pyb import Pin, Timer, ADC, DAC, LED
import micropython   # Needed for interrupt
micropython.alloc_emergency_exception_buf(100)
........

#  Initialise various peripherals e.g. OLED, IMU etc

#  Initialise different constants, variables, arrays etc

#  Read the dancing steps from file into array

#  Use Potentiometer and USR switch to tune Kp, Ki and Kd

tic2 = pyb.millis()         # mark time now in msec

try:            # Try to handle exception
    tic1 = pyb.micros()
    while True:     # infinite loop
        dt = pyb.micros() - tic1
        if (dt > 5000):     # loop period is 5 msec or 200Hz
            estimate pitch angle and pitch_dot
            update tic1
            do PID control
            use returned value to move motor
        if microphone buffer is full
            test if beat has occurred
            if yes, update the target of pitch angle
            update tic2

finally:        # always executed if exception
    motor.A_stop()
    motor.B_stop()
```

Note that this program loop has two time scales: the controller uses tic1, in microseconds. The polling loop checks for 5 msec elapse time. This means the control loop is running at round 200Hz.

The beat detection uses tic2 in milliseconds. The loop is detecting beats at much lower frequency.