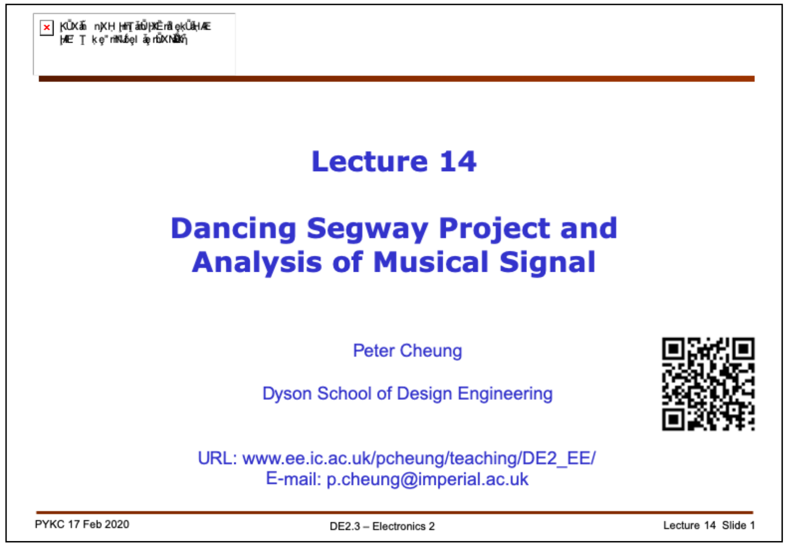


Lecture 14

Dancing Segway Project and Analysis of Musical Signal

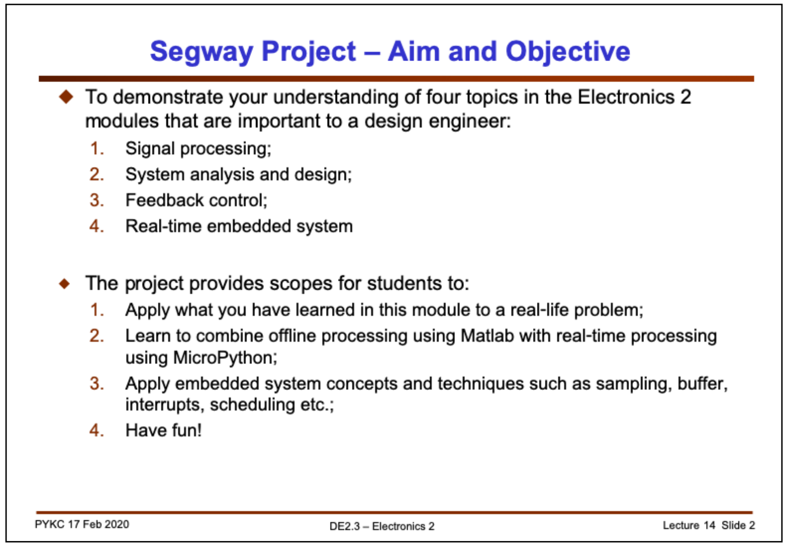
Peter Cheung

Dyson School of Design Engineering

URL: www.ee.ic.ac.uk/pcheung/teaching/DE2_EE/
E-mail: p.cheung@imperial.ac.uk

In this lecture, we will introduce the team project – the dancing Segway.

You will be conducting further Lab Sessions main at helping you to achieve the milestones for this project. This lecture will provide some basic background for these lab sessions. In particular, you will learn how to detect beat and handle real-time data capture.

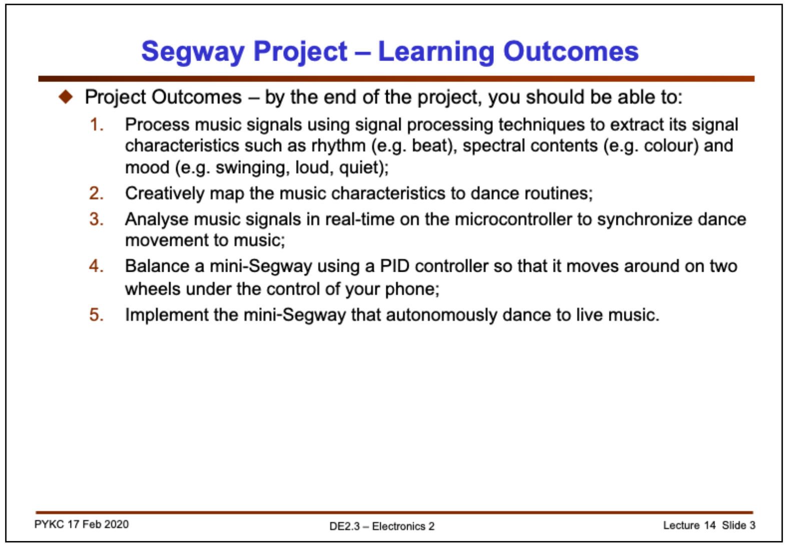


Segway Project – Aim and Objective

- To demonstrate your understanding of four topics in the Electronics 2 modules that are important to a design engineer:
  1. Signal processing;
  2. System analysis and design;
  3. Feedback control;
  4. Real-time embedded system
- The project provides scopes for students to:
  1. Apply what you have learned in this module to a real-life problem;
  2. Learn to combine offline processing using Matlab with real-time processing using MicroPython;
  3. Apply embedded system concepts and techniques such as sampling, buffer, interrupts, scheduling etc.;
  4. Have fun!

This slide shows the aims and objectives of the team project. As in last year, the project helps 2 to glue together everything you have learned in this module.

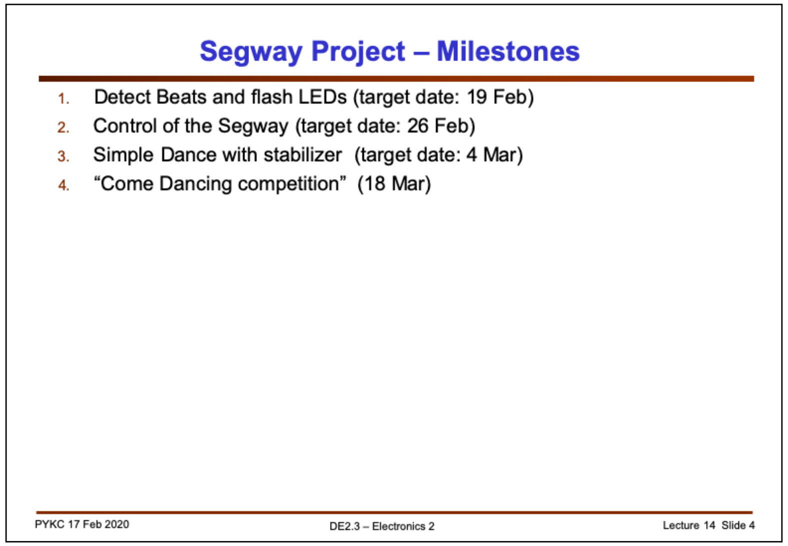
So far, I have covered the necessary theory for aims 1, 2 and 4, either through lectures or through the lab experiments. I will be covering the topic of feedback control starting next lecture.



Segway Project – Learning Outcomes

- Project Outcomes – by the end of the project, you should be able to:
  1. Process music signals using signal processing techniques to extract its signal characteristics such as rhythm (e.g. beat), spectral contents (e.g. colour) and mood (e.g. swinging, loud, quiet);
  2. Creatively map the music characteristics to dance routines;
  3. Analyse music signals in real-time on the microcontroller to synchronize dance movement to music;
  4. Balance a mini-Segway using a PID controller so that it moves around on two wheels under the control of your phone;
  5. Implement the mini-Segway that autonomously dance to live music.

For any module you take on your course, you should be very clear about learning outcomes. Here is the project's learning outcomes – stating what you should be able to do when you have finished this project.



Segway Project – Milestones

1. Detect Beats and flash LEDs (target date: 19 Feb)
2. Control of the Segway (target date: 26 Feb)
3. Simple Dance with stabilizer (target date: 4 Mar)
4. "Come Dancing competition" (18 Mar)

MILESTONES

1. Detect Beats and flash LEDs – To analyse the music using the microphone on the PyBench and detect when the beat occurs. Flash LEDs to indicate beats.

Target completion date: 19th Feb.

2. Control of the Segway – To fit a pair of "stabilizers" to the two-wheel chassis so that the vehicle will not fall over, but have enough clearance to implement self-balancing later. To drive the Segway with stabilizer along a defined path under the control of the mobile phone connect to the PyBench via Bluetooth.

Target completion date: 26st Feb.

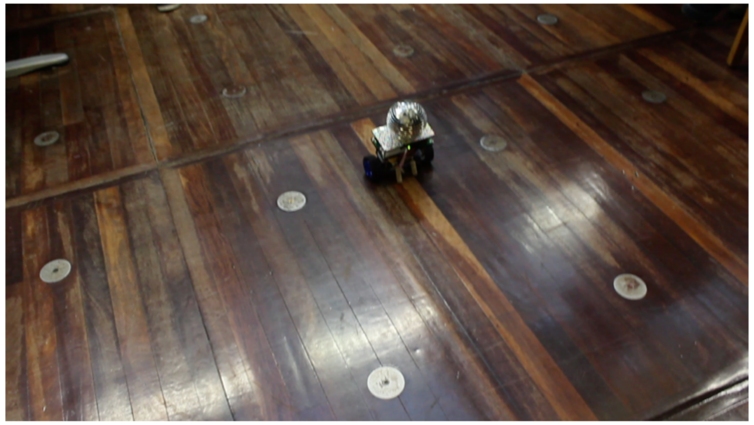
3. Simple Dance with stabilizer – To analyse the music "Staying Alive" and one other song of your choice in Matlab, create simple dance routines, transfer to the PyBench to store on the SD card and write a Python program on the PyBench to control the Segway with stabilizer so that it moves to music.

Target completion date: 4th March.

4. "Come Dancing competition" – The final assessment will involve demonstrating the dancing Segway that synchronizes to life music, while balancing on two-wheels. Assessment will be based on the robustness of the vehicle, the creativity of the dance routine, the quality of the synchronization to the music etc.

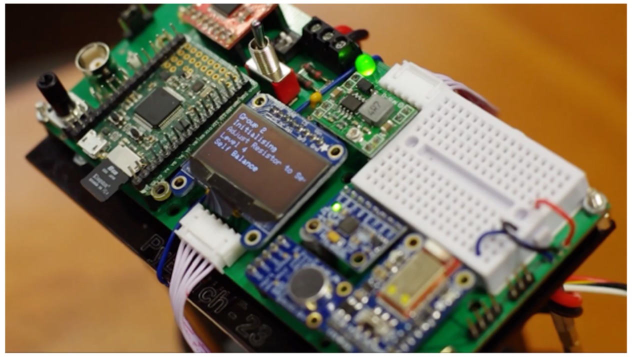
Final assessment of the Team will take place on: Wednesday 18th March.

Segway Project – Milestone 3

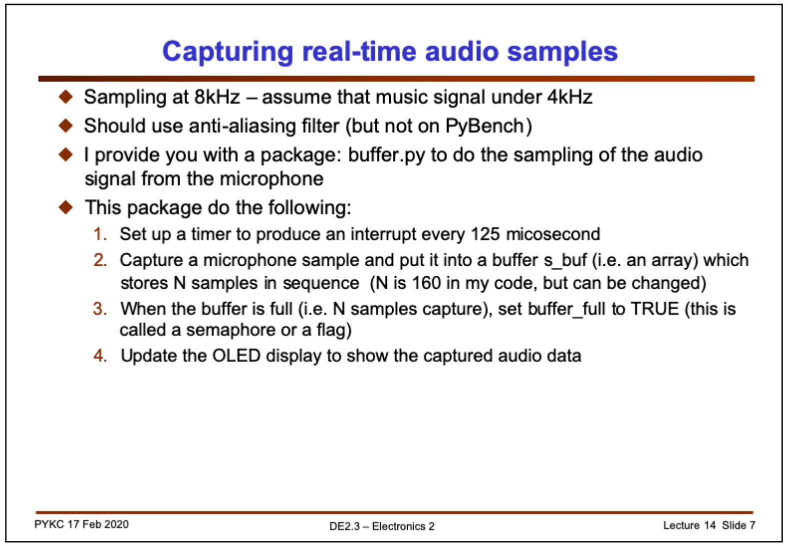


Here is what milestone 3 could look like. This Segway does not self-balance, but rely on the stabilizer to keep it upright. However, it does synchronise to music and dance!

Segway Project – Milestone 4



This is a video showing different groups achievement from a previous year. Some Segways only perform dance to music. Others can do dance and balance at the same time. The final clip shows driving a self-balancing Segway with a camera onboard!

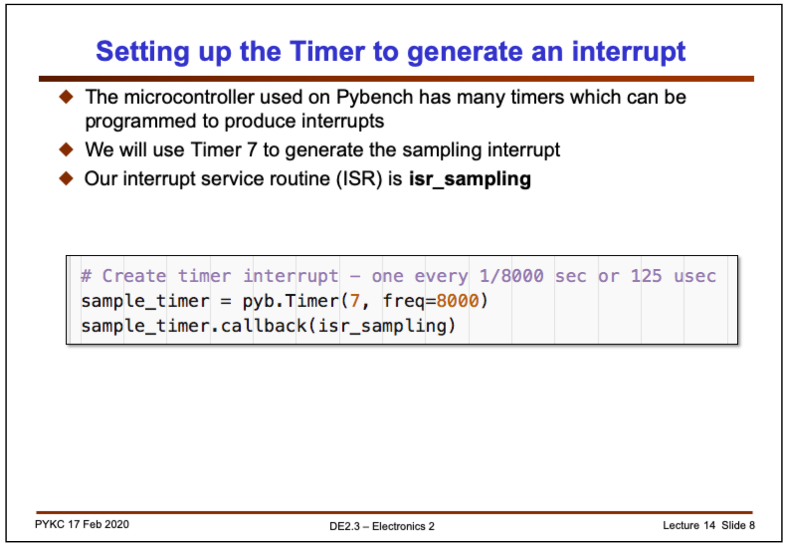


Capturing real-time audio samples

- Sampling at 8kHz – assume that music signal under 4kHz
- Should use anti-aliasing filter (but not on PyBench)
- I provide you with a package: buffer.py to do the sampling of the audio signal from the microphone
- This package do the following:
  1. Set up a timer to produce an interrupt every 125 micosecond
  2. Capture a microphone sample and put it into a buffer s_buf (i.e. an array) which stores N samples in sequence (N is 160 in my code, but can be changed)
  3. When the buffer is full (i.e. N samples capture), set buffer_full to TRUE (this is called a semaphore or a flag)
  4. Update the OLED display to show the captured audio data

I now want to introduce you to the next Lab. From now on, Labs are not assessed. However, I will be providing you two more instructions to help you with the project.

The goal for Wednesday's lab is to capture audio data in real-time using interrupts, so that you can perform beat detection and eventually synchronize your dance moves to the beat of the music.



Setting up the Timer to generate an interrupt

- The microcontroller used on Pybench has many timers which can be programmed to produce interrupts
- We will use Timer 7 to generate the sampling interrupt
- Our interrupt service routine (ISR) is **isr_sampling**

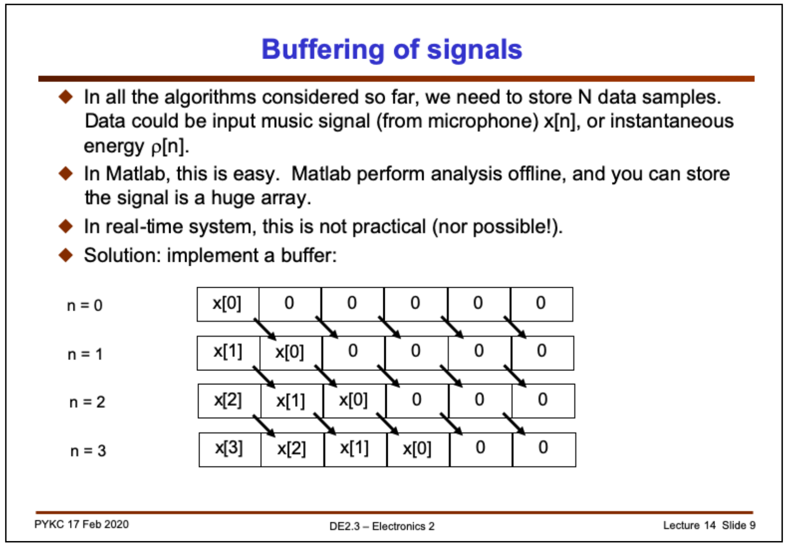
```
# Create timer interrupt - one every 1/8000 sec or 125 usec
sample_timer = pyb.Timer(7, freq=8000)
sample_timer.callback(isr_sampling)
```

This is how you can program a timer (Timer 7) to produce an interrupt every 125 microseconds. The interrupt service routine is specified with the callback function.

Ask yourself this question: what does the microprocessor do when an interrupt occurs?

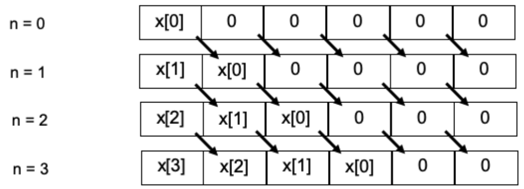
The answer is:

1. Assuming the microprocessor is enabled to respond to interrupts, it will complete the current instruction;
2. It stores away (in some memory called "stack") the location of the next instruction, so that it can return later to continue your main program;
3. It stores away the internal values of the processor (called processor state or contex);
4. It jumps to the interrupt service routine and do whatever that specifies;
5. On completing the ISR, it recovers the state of the processor (or contex);
6. It return to the place in the main program where it was previously interrupted.



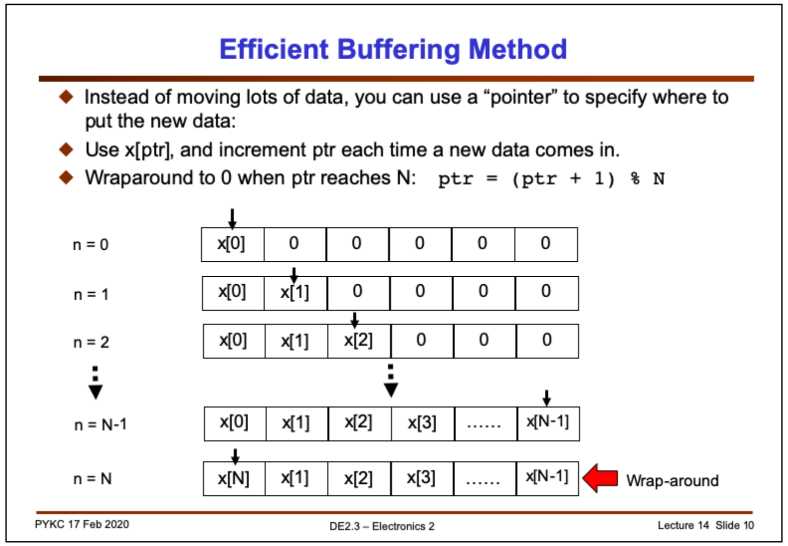
Buffering of signals

- In all the algorithms considered so far, we need to store N data samples. Data could be input music signal (from microphone) x[n], or instantaneous energy ρ[n].
- In Matlab, this is easy. Matlab perform analysis offline, and you can store the signal is a huge array.
- In real-time system, this is not practical (nor possible!).
- Solution: implement a buffer:



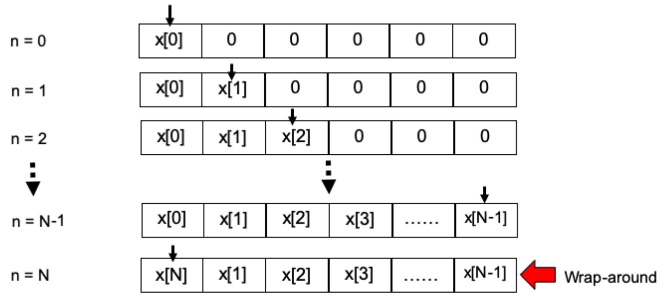
We need to take N samples of audio data in order to work out when a beat occurs. To do that, we use something called a buffer.

There is a type of buffer known as FIFO: first-in first-out buffer. This is shown in the diagram below. This is like a queue – queue to get onto a bus.



Efficient Buffering Method

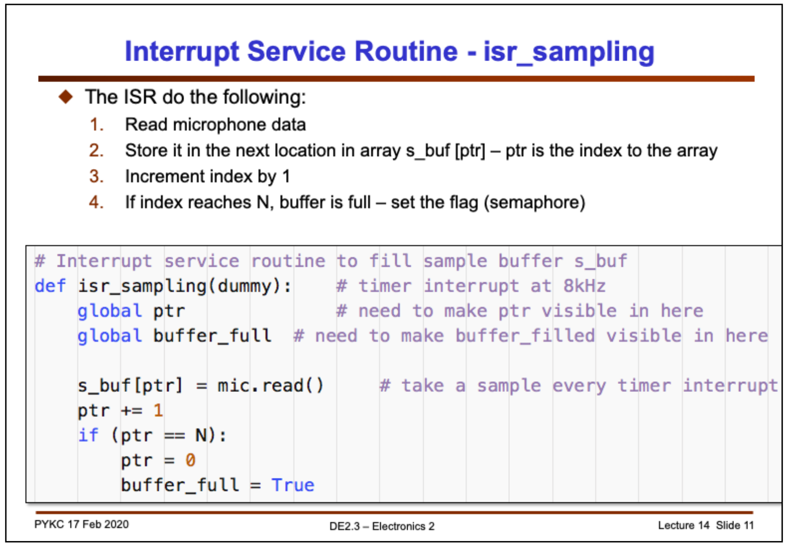
- Instead of moving lots of data, you can use a “pointer” to specify where to put the new data:
- Use x[ptr], and increment ptr each time a new data comes in.
- Wraparound to 0 when ptr reaches N: `ptr = (ptr + 1) % N`


n = 0 x[0] 0 0 0 0 0
n = 1 x[0] x[1] 0 0 0 0
n = 2 x[0] x[1] x[2] 0 0 0
n = N-1 x[0] x[1] x[2] x[3] …… x[N-1]
n = N x[N] x[1] x[2] x[3] …… x[N-1]
Wrap-around

Using a FIFO is not efficient because you are moving data around a lot – that costs time and energy.

A better method is to use a concern call pointer. This is shown in the diagram above. You fill an area of memory (or array) in sequence, and move the the pointer (shown as arrow) up a position as shown above.

Note that if you keep incrementing the pointer, it wraps around at the end back to the beginning in a circular way. This is known as a circular buffer.



Interrupt Service Routine - isr_sampling

- The ISR do the following:
  1. Read microphone data
  2. Store it in the next location in array s_buf [ptr] – ptr is the index to the array
  3. Increment index by 1
  4. If index reaches N, buffer is full – set the flag (semaphore)

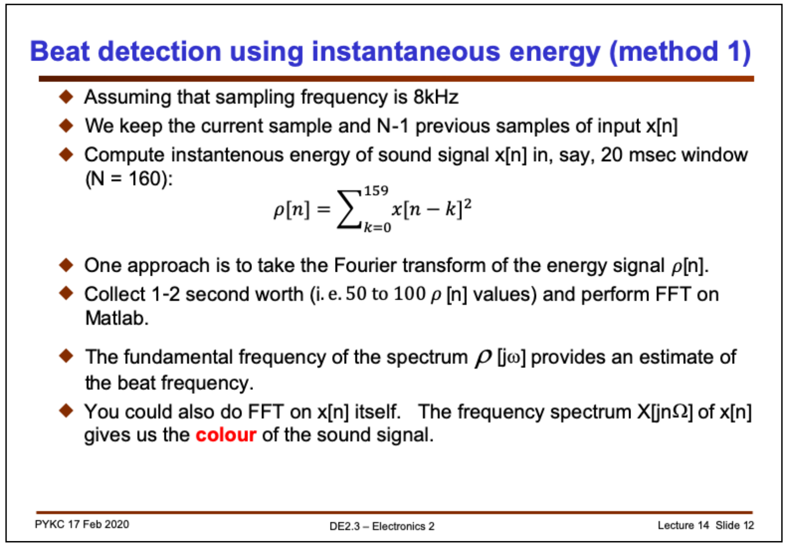
```python
# Interrupt service routine to fill sample buffer s_buf
def isr_sampling(dummy):     # timer interrupt at 8kHz
    global ptr               # need to make ptr visible in here
    global buffer_full  # need to make buffer_filled visible in here

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

Now let us consider how we write code in MicroPython to achieve all these. You will be doing it tomorrow in the Lab.

The interrupt service routine (ISR) is isr_sampling. We have to make ptr and buffer_full global variables because we need to access these OUTSIDE the routine. s_buf is also global – defined OUTSIDE the ISR.

The rest of the code is simple.



Beat detection using instantaneous energy (method 1)

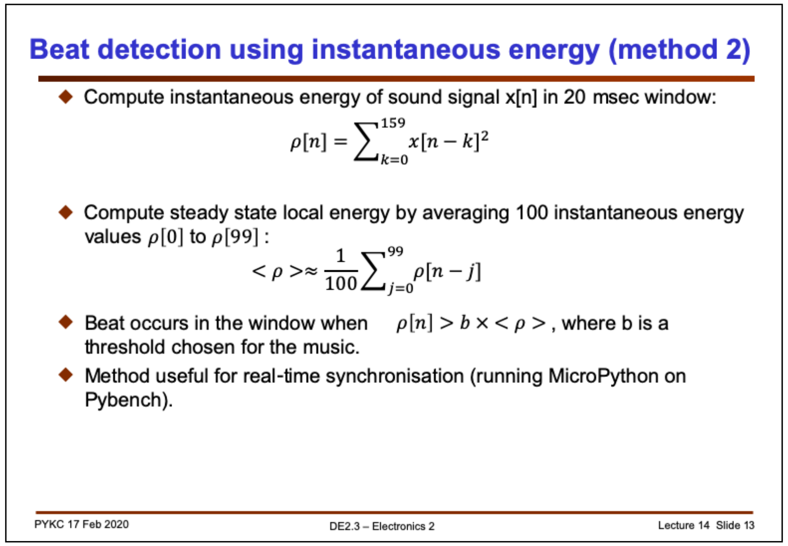
- Assuming that sampling frequency is 8kHz
- We keep the current sample and N-1 previous samples of input x[n]
- Compute instantenous energy of sound signal x[n] in, say, 20 msec window (N = 160):

$$\rho[n] = \sum_{k=0}^{159} x[n-k]^2$$

- One approach is to take the Fourier transform of the energy signal ρ[n].
- Collect 1-2 second worth (i.e. 50 to 100 ρ [n] values) and perform FFT on Matlab.
- The fundamental frequency of the spectrum ρ [jω] provides an estimate of the beat frequency.
- You could also do FFT on x[n] itself. The frequency spectrum X[jnΩ] of x[n] gives us the **colour** of the sound signal.

Now we will consider three methods in determining when a beat occurs. The simples way is the calculate the instantaneous energy of the sound signal. We have done this before. Once you have calculated the instantaneous energy for every 20 msec, we can work out the periodicity of this energy using FFT (Fast Fourier Transform) in Matlab.

In addition, you may also find the spectrum of x[n]. This gives you some information about the "colour" of the music. However, I found that it is rather difficult to deduce characteristic of music from the spectrum.



Beat detection using instantaneous energy (method 2)

- Compute instantaneous energy of sound signal x[n] in 20 msec window:

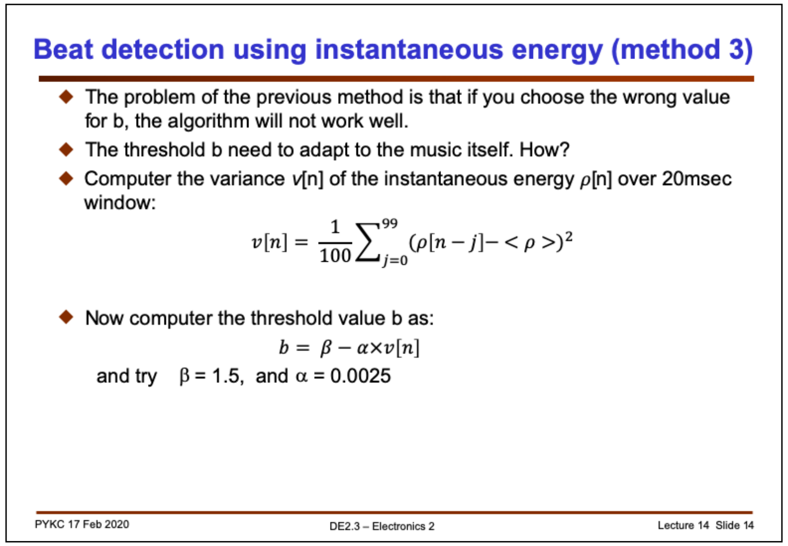
$$\rho[n] = \sum_{k=0}^{159} x[n-k]^2$$

- Compute steady state local energy by averaging 100 instantaneous energy values $\rho[0]$ to $\rho[99]$:

$$< \rho > \approx \frac{1}{100}\sum_{j=0}^{99} \rho[n-j]$$

- Beat occurs in the window when $\rho[n] > b \times < \rho >$, where b is a threshold chosen for the music.
- Method useful for real-time synchronisation (running MicroPython on Pybench).

Second method is an improvement. Apart from computing the instantaneous energy, you can also compute the steady state energy by averaging 100 instantaneous energy readings. A beat is deemed to have occurred if determining the energy rises above a threshold.



Beat detection using instantaneous energy (method 3)

- The problem of the previous method is that if you choose the wrong value for b, the algorithm will not work well.
- The threshold b need to adapt to the music itself. How?
- Computer the variance *v*[n] of the instantaneous energy *ρ*[n] over 20msec window:

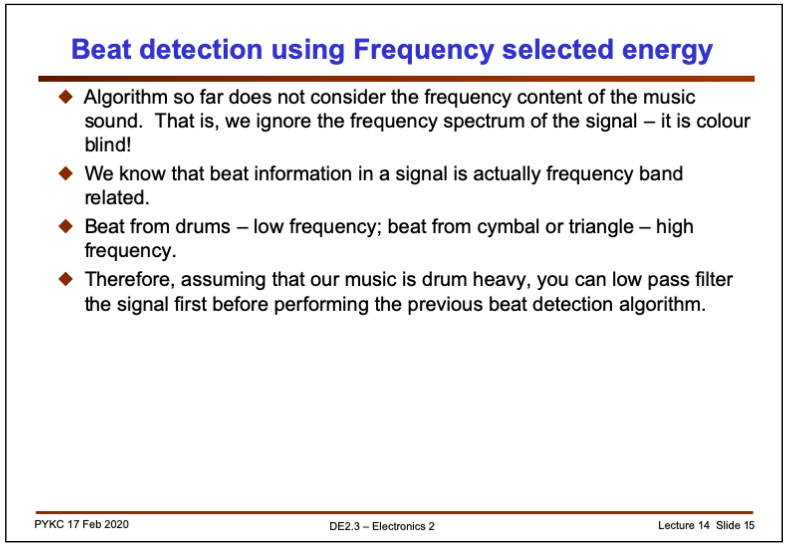
$$v[n] = \frac{1}{100}\sum_{j=0}^{99} (\rho[n-j] - < \rho >)^2$$

- Now computer the threshold value b as:

$$b = \beta - \alpha \times v[n]$$

and try β = 1.5, and α = 0.0025

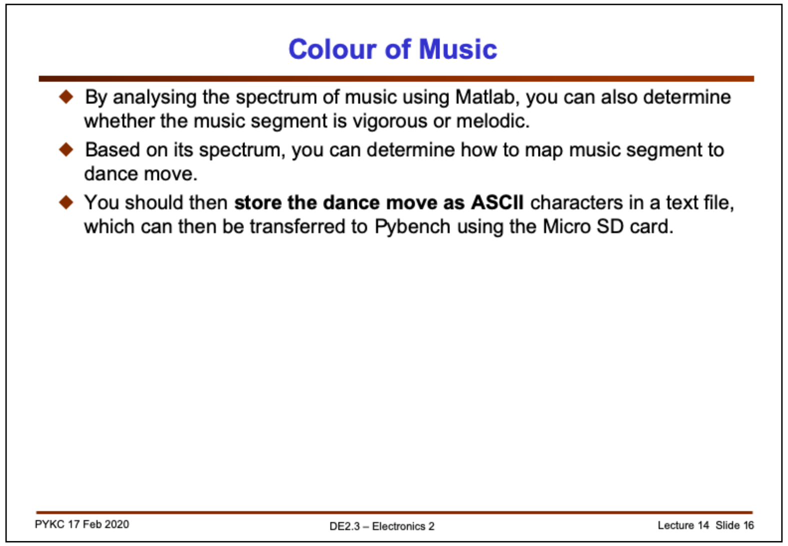
Finally, you can even adapt the threshold value b according the music itself by computing the variance of the instantaneous energy. This is shown above.



Beat detection using Frequency selected energy

- Algorithm so far does not consider the frequency content of the music sound. That is, we ignore the frequency spectrum of the signal – it is colour blind!
- We know that beat information in a signal is actually frequency band related.
- Beat from drums – low frequency; beat from cymbal or triangle – high frequency.
- Therefore, assuming that our music is drum heavy, you can low pass filter the signal first before performing the previous beat detection algorithm.

Finally, you could include the frequency spectrum information in order to determine the beat. However, I don't recommend you doing this on MicroPython - we do not have FFT routines written for the Pyboard, and it would take too long anyway.



Colour of Music

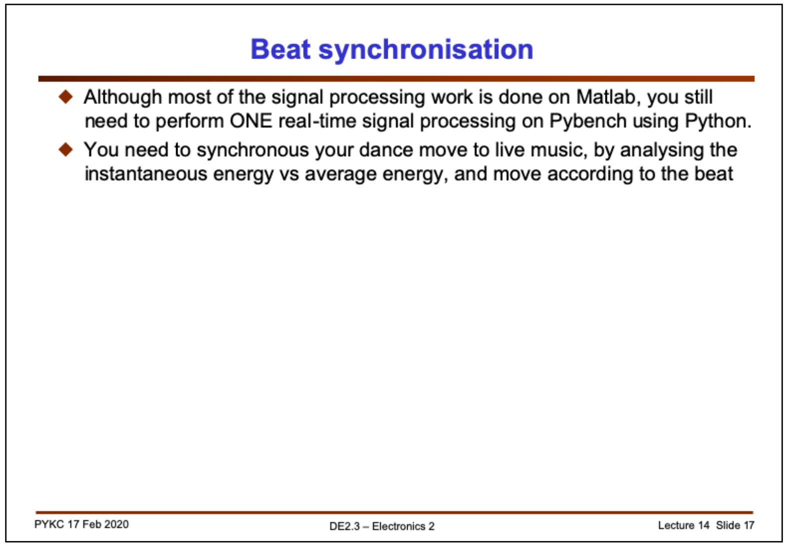
- By analysing the spectrum of music using Matlab, you can also determine whether the music segment is vigorous or melodic.
- Based on its spectrum, you can determine how to map music segment to dance move.
- You should then **store the dance move as ASCII** characters in a text file, which can then be transferred to Pybench using the Micro SD card.

For Milestone 3, you need to somehow generate a dance routine in order to synchronize to the beat of the music.

You should store the dance routine as a text file and transfer that to the SD card. For example, you could have a code such as:

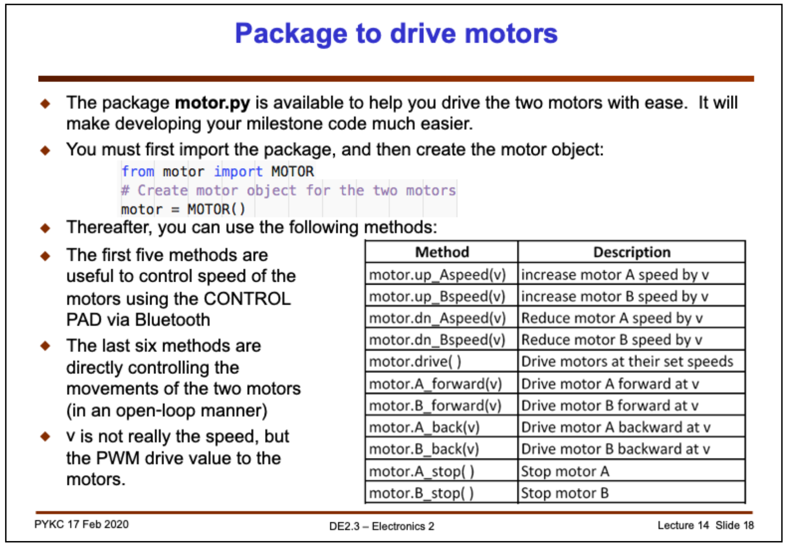
F4B4R4L4….

For forward four beats, backward 4 beats, right turn 4 beats, left turn 4 beats ….



Beat synchronisation

- Although most of the signal processing work is done on Matlab, you still need to perform ONE real-time signal processing on Pybench using Python.
- You need to synchronous your dance move to live music, by analysing the instantaneous energy vs average energy, and move according to the beat



Package to drive motors

- The package **motor.py** is available to help you drive the two motors with ease. It will make developing your milestone code much easier.
- You must first import the package, and then create the motor object:

```
from motor import MOTOR
# Create motor object for the two motors
motor = MOTOR()
```

- Thereafter, you can use the following methods:
- The first five methods are useful to control speed of the motors using the CONTROL PAD via Bluetooth
- The last six methods are directly controlling the movements of the two motors (in an open-loop manner)
- v is not really the speed, but the PWM drive value to the motors.

Method	Description
motor.up_Aspeed(v)	increase motor A speed by v
motor.up_Bspeed(v)	increase motor B speed by v
motor.dn_Aspeed(v)	Reduce motor A speed by v
motor.dn_Bspeed(v)	Reduce motor B speed by v
motor.drive( )	Drive motors at their set speeds
motor.A_forward(v)	Drive motor A forward at v
motor.B_forward(v)	Drive motor B forward at v
motor.A_back(v)	Drive motor A backward at v
motor.B_back(v)	Drive motor B backward at v
motor.A_stop( )	Stop motor A
motor.B_stop( )	Stop motor B

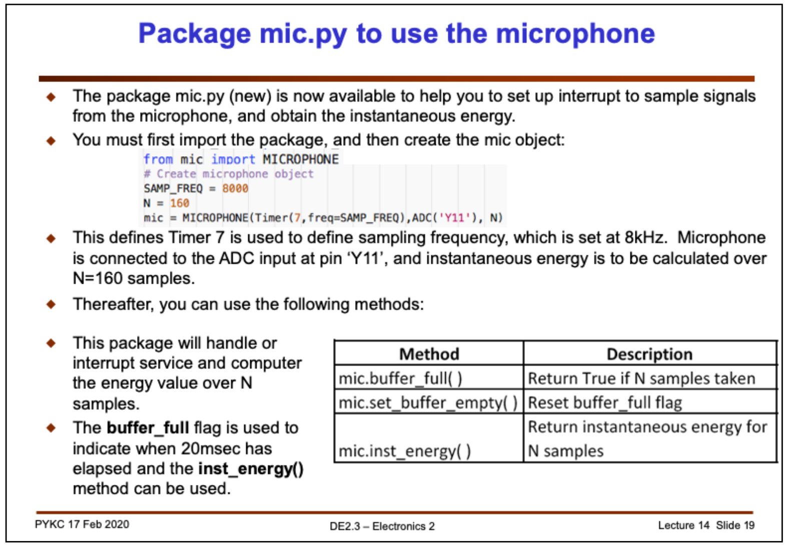
Since motor coils are essentially inductors, they have low DC impedances (resistance of the wiring). Hence when driving motors, we need to use special driver chips.

The driver chip you used in Lab 4 (the TB6612) is often called the H-Bridge Driver. Shown here is the simplified block diagram. There are four transistors connected to the supply rail and ground. (It doesn't matter which is which because the circuit is symmetrical.) The motor is connected in the middle forming the horizontal link of the H. The transistors are MOSFETs (metal oxide silicon field effect transistors) which is acting like a voltage controlled switch. When a '1' or high voltage is applied to the gate control terminal, the transistor turns ON and conduct electricity. If a '0' or low voltage is applied, the transistor is OFF. So the top diagram shows a configuration that results in the supply voltage being applied to the left terminal of the motor. The right terminal of the motor is grounded, and the motor turns in one direction. Reversing the control to the transistors results in the motor turning in the other direction.

If you use an AND gate at the control input, you can also add a PWM signal to control the speed of the motor.

Basically the '1' and '0' control signals are the A0 and A1 signals on the TB6612. The PWM signal is what you apply to the input of the AND gate.

Now you know how the TB6612 works.



Package mic.py to use the microphone

- The package mic.py (new) is now available to help you to set up interrupt to sample signals from the microphone, and obtain the instantaneous energy.
- You must first import the package, and then create the mic object:

```
from mic import MICROPHONE
# Create microphone object
SAMP_FREQ = 8000
N = 160
mic = MICROPHONE(Timer(7,freq=SAMP_FREQ),ADC('Y11'), N)
```

- This defines Timer 7 is used to define sampling frequency, which is set at 8kHz. Microphone is connected to the ADC input at pin 'Y11', and instantaneous energy is to be calculated over N=160 samples.
- Thereafter, you can use the following methods:
- This package will handle or interrupt service and computer the energy value over N samples.
- The **buffer_full** flag is used to indicate when 20msec has elapsed and the **inst_energy()** method can be used.

Method	Description
mic.buffer_full( )	Return True if N samples taken
mic.set_buffer_empty( )	Reset buffer_full flag
mic.inst_energy( )	Return instantaneous energy for N samples

PYKC 17 Feb 2020 DE2.3 – Electronics 2 Lecture 14 Slide 19

Here is another package provided for you to implement Milestone 3. It relates to the microphone buffer, but packaged together as a python package. It is very easy to understand.