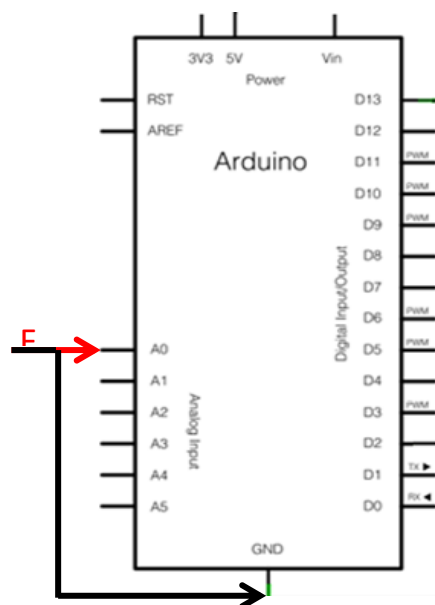


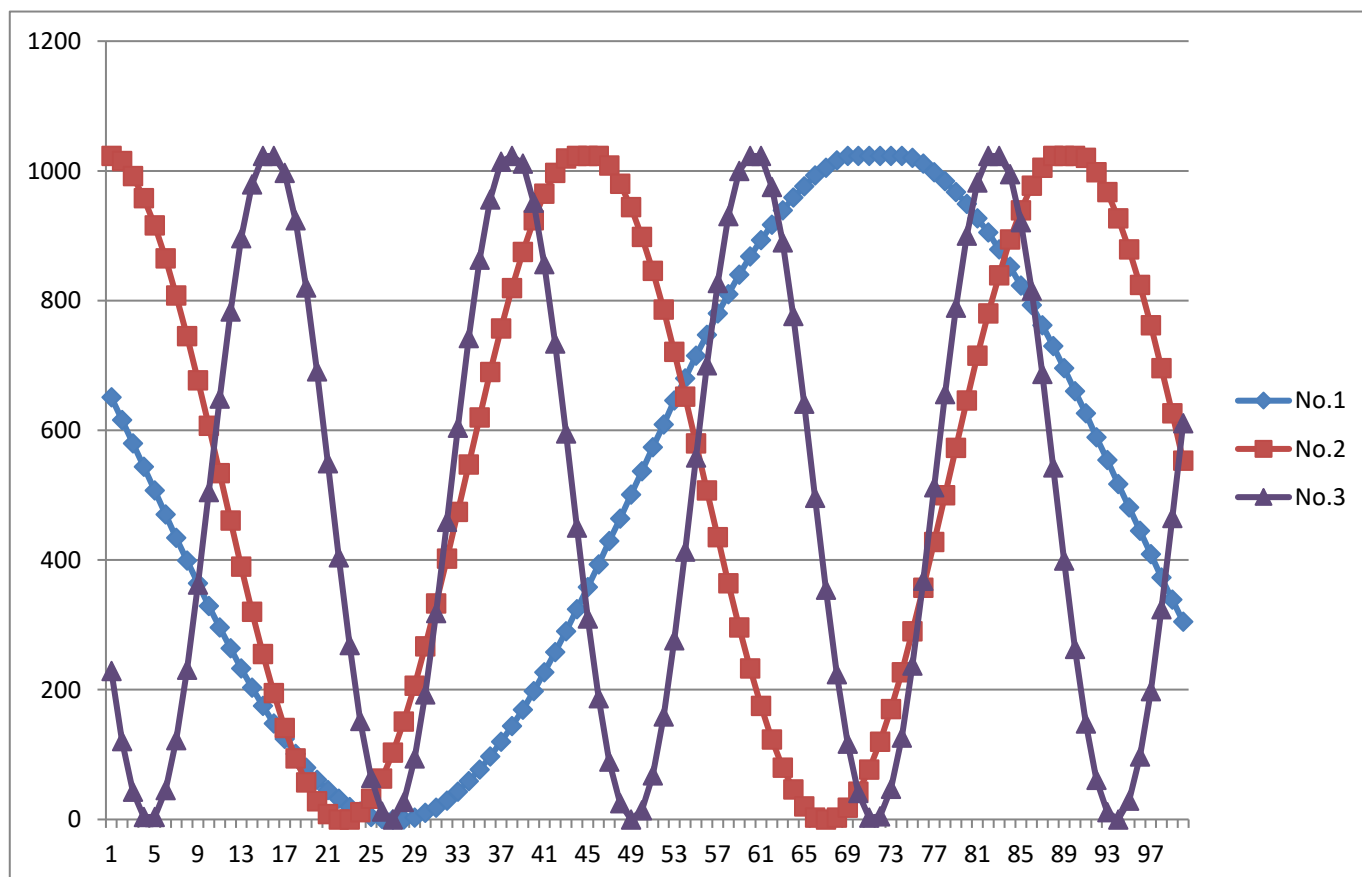
REPORT

Experiment 1: Sampling and Aliasing.



1.

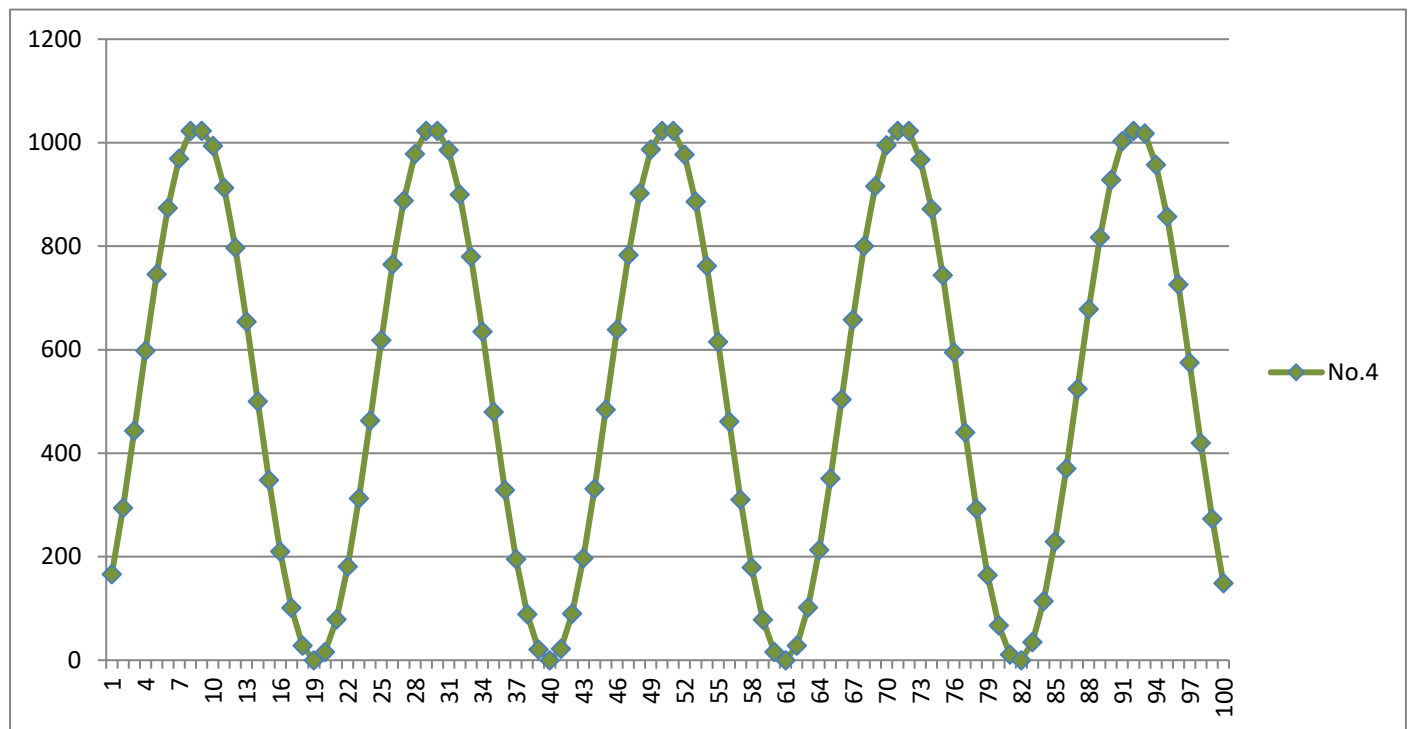
Arduino serial monitor results plotted by excel – No.1, 2, 3 / Sine / Vpp:0-5V / freq:100/200/400Hz



從上圖可以看到我們透過訊號產生器產生了三種不同頻率的類比訊號並且透過 Arduino 內部的 ADC 將類比訊號轉換為數位訊號，而者個 ADC 的 resolution 為 10bit，因此輸入會被對應到 0~1023 的數位訊號，此外也沒有發生因為 sampling frequency 選擇不良而造成的 aliasing 失真問題。

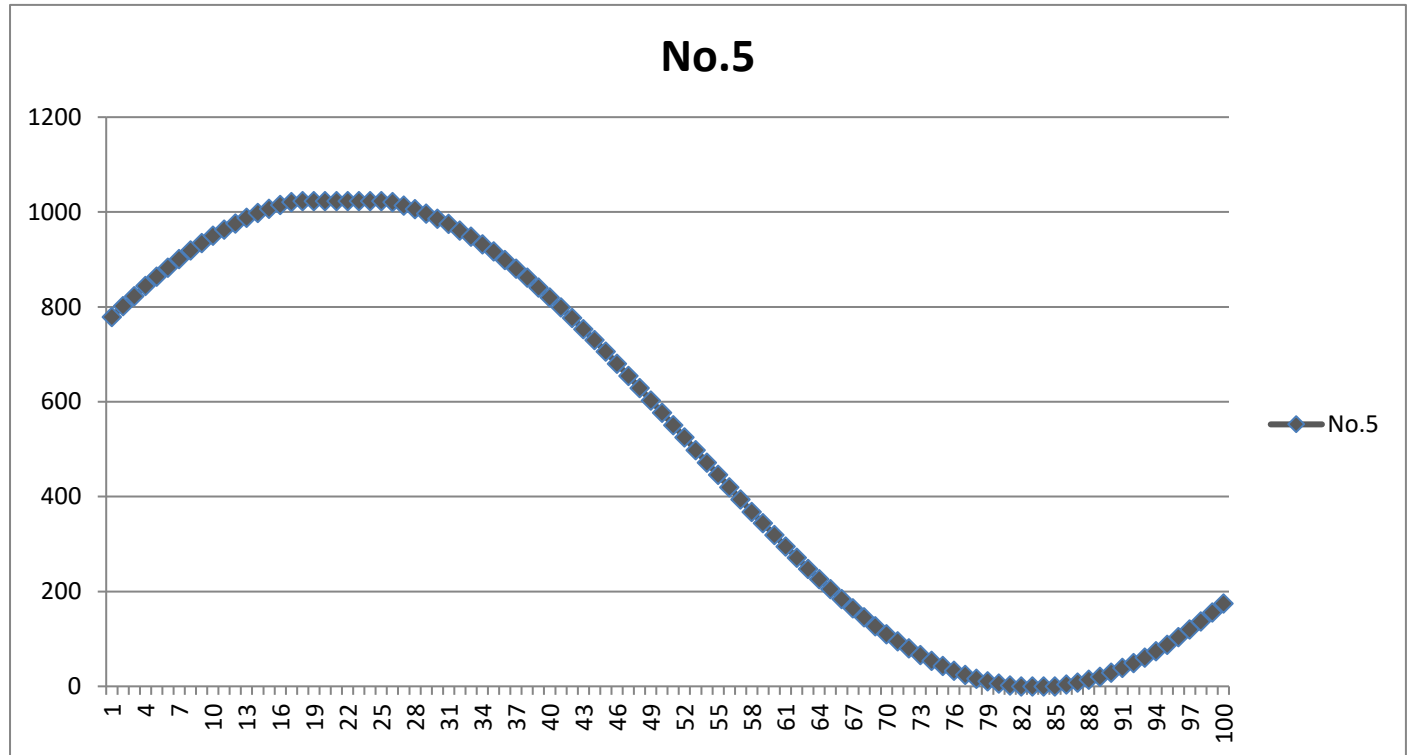
2.

Arduino serial monitor results plotted by excel – No.4 / Sine / Vpp:0-5V / freq:8.5kHz



3.

Arduino serial monitor results plotted by excel – No.5 / Sine / Vpp:0-5V / freq:9kHz



從上兩圖可以得知經過 Arduino ADC sampling 時並沒有很好的將原本的 9kHz 的訊號 sampling 出來，反倒分別將其轉成大約 500Hz / 100Hz 的弦波數位訊號。因此可以判斷應該是產生了因為 sampling frequency 選擇不良而造成 Aliasing 導致訊號失真的問題。

4.

Please estimate the sampling frequencies by your experiment results of No.1~No.3, and explain how you obtain it. Are the three sampling frequencies the same?

根據序列埠監控視窗印出的數字，找出在一個週期中，sample 了多少次，再將其乘上 FG 輸入波型的頻率即可算出 sampling frequency 為多少。

NO1: $89 * 100 \text{ Hz} = 8900\text{Hz}$

NO2: $44 * 200 \text{ Hz} = 8800\text{Hz}$

NO3: $22 * 400 \text{ Hz} = 8800\text{Hz}$

Question:

Use the result of No1., No.2, and No.3 to estimate the correct sampling rate (calculate the average of these three conditions). And apply this correct sampling rate to derive the signal frequency of No.4 and No.5 respectively. (Notice that the frequencies you calculated are false signal frequency.)

Please try to explain them in your report

三者皆接近 8900 Hz 附近，計算三者頻率的平均為:8833Hz，因此可以推得 Arduino 內部的 ADC 的 sampling frequency 為 8833 Hz 左右，與官方公告的 10kHz 大約相差 1kHz。

因此我們便可以透過這個 sampling frequency 去反推出 No.4 與 No.5 的頻率為:

No.4: $8833\text{Hz} / 16 = 552 \text{ Hz}$

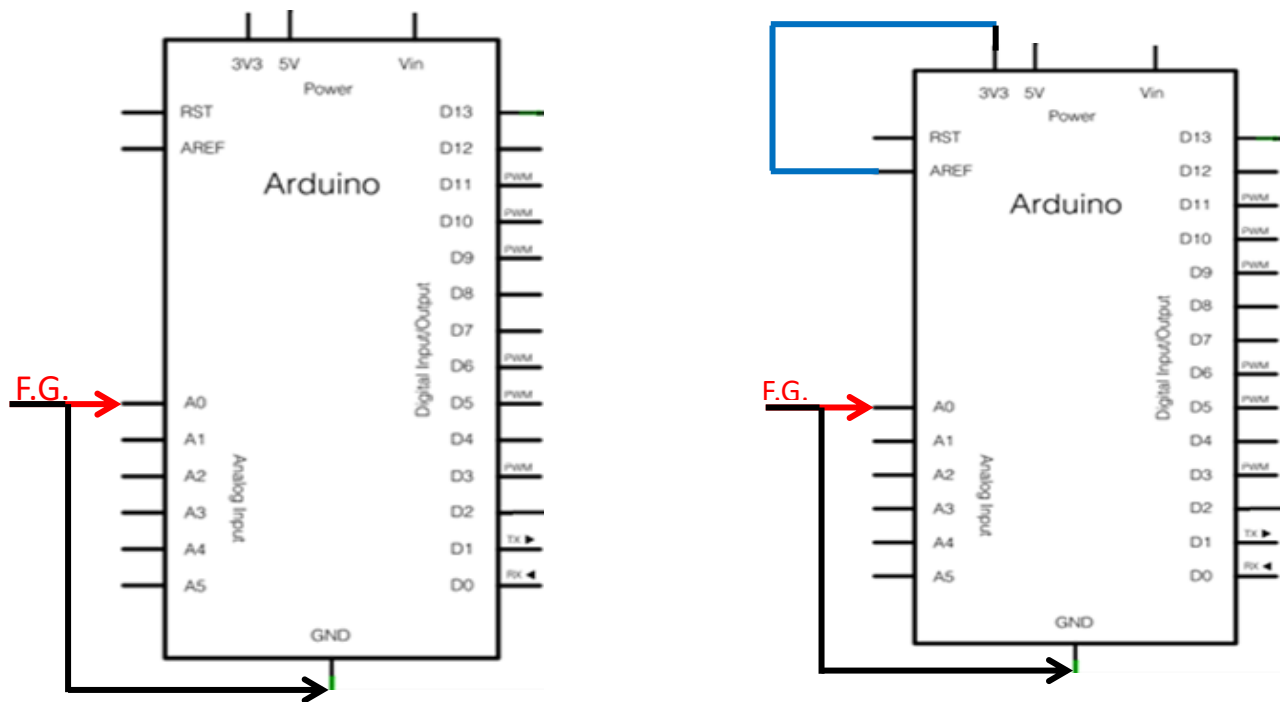
No.5: $8833\text{Hz} / 100 = 88.33 \text{ Hz}$

而為何會造成這種 aliasing 的問題呢，是因為 Arduino ADC 的 sampling frequency 與輸入訊號的頻率過於接近，因此將其轉換為數位訊號輸出後會產生 aliasing 的失真問題。

心得:

在日後進行訊號處理或是繪製波型時要注意 sampling frequency 的選擇，必須要使 sampling frequency 大於輸入訊號的頻率，如此一來才能夠避免因為 sampling frequency 不夠大而造成波型因為 aliasing 而失真的問題。想起之前訊號與系統或是機器學習課程在作圖時，完全沒有這種 sampling 要選多少的概念，而也都很幸運沒有遇到相關的 aliasing 的問題，而後續我也有去與訊號與系統的教授討論一般來說 sampling frequency 應該要比輸入訊號的頻率大 10 倍以上，如此一來便比較能夠避免相關問題發生。

Experiment 2: Reference Voltage for Analog Input.

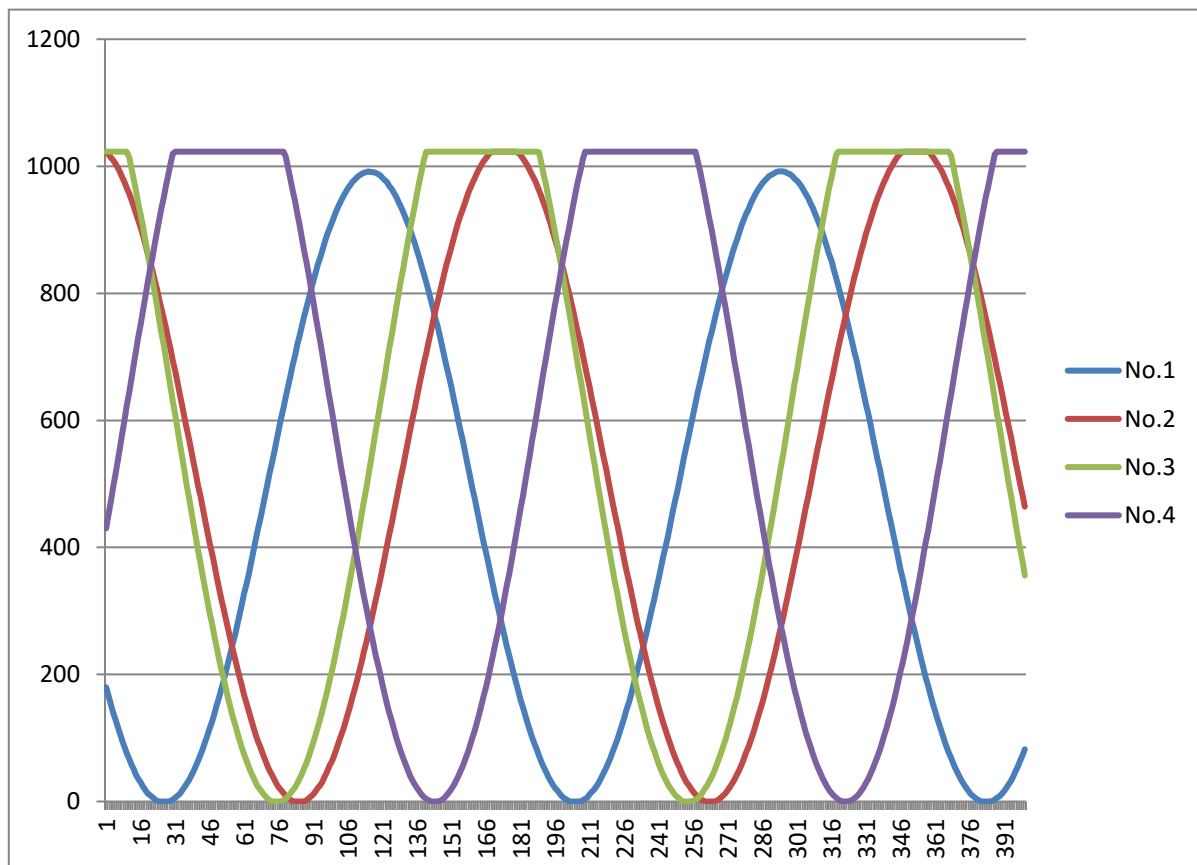


NOTE : Remember modify the sketch in Exp1, change MAX_SIZE from 100 to 400.

NOTE : Remember uncomment `analogReference(EXTERNAL);` when you connect pin 3.3V to AREF.

1.

Arduino serial monitor results plotted by excel – Sine / Vpp:0-4.8V/5V/6V/4V+Aref:3.3V / freq:50Hz



Question:

1). Please estimate the full-scale input range A_{FS} of ADC of Arduino by your experiment results of No.1~No.3, and explain how you obtain it.

從上圖可以得知當輸入的類比訊號在 0~5V 之間皆能夠透過 ADC 成功轉換成數位訊號，但是當輸入類比訊號在 0~6V 時，便可以發現他超過 5V 的部分被砍掉了，因此可以推測 A_{FS} 大約界在 5V 與 6V 之間。原因是在 ADC 內部將輸入的類比訊號經過量化後必須要將其放進 0~1023 這些數值(解析度為 10 bit)，但是當輸入大於 5V 之後，便可以發現已經沒有多餘的 bit 去存放量化之後大於 1023 的數值，若要解決這個因為量化解析度不構而造成數位訊號失真的問題，我認為可以採用較 A_{FS} 較大或是解析度較小的 ADC，如此一來便能夠接收更多輸入範圍的類比訊號。

2). Please try to explain the experiment results of No.4 in your report

因為電位是一個相對的概念，在前面三種訊號的都是與接地端 GND 相對。而這邊將 Aref 與 3.3V 相接，將低電位設為 3.3V，又輸入的類比訊號的最高電位與最低電位差值位 4V，因此可以得知，在相對於 3.3V 的情況下，最高的電位會到 $3.3 + 4 = 7.3V$ ，已經超過該 ADC 的 A_{FS} 了，因此也可以看到數位訊號被截掉了。

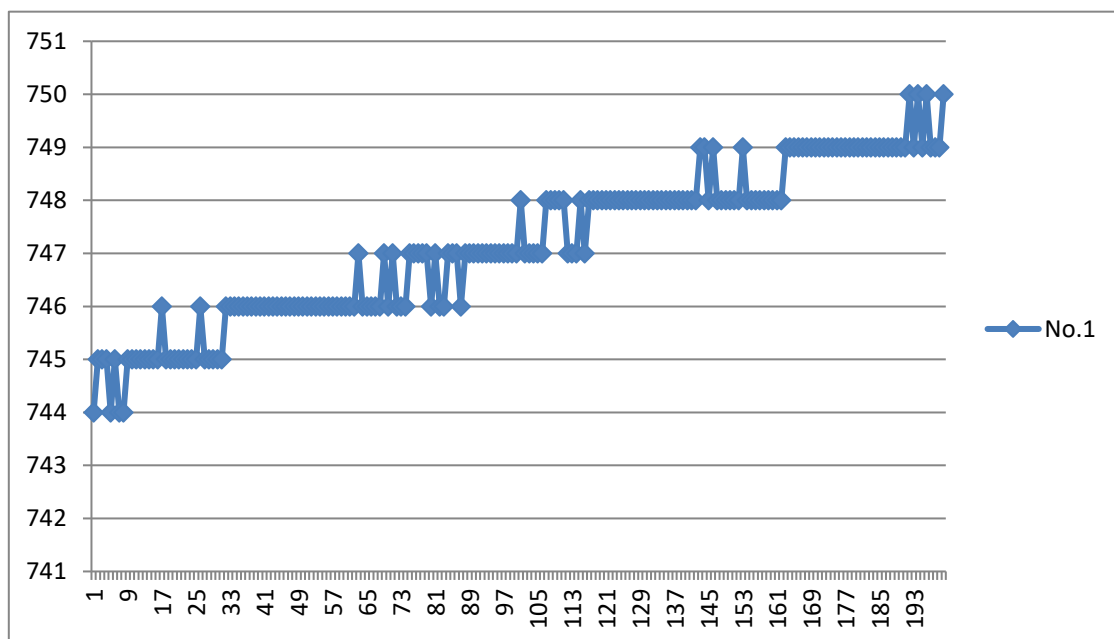
Experiment 3: Quantization Error.

NOTE : Remember modify the sketch in Exp1, change MAX_SIZE from 100 to 200.

NOTE : Remember uncomment analogReference(EXTERNAL); when you connect pin 3.3V to AREF.

1.

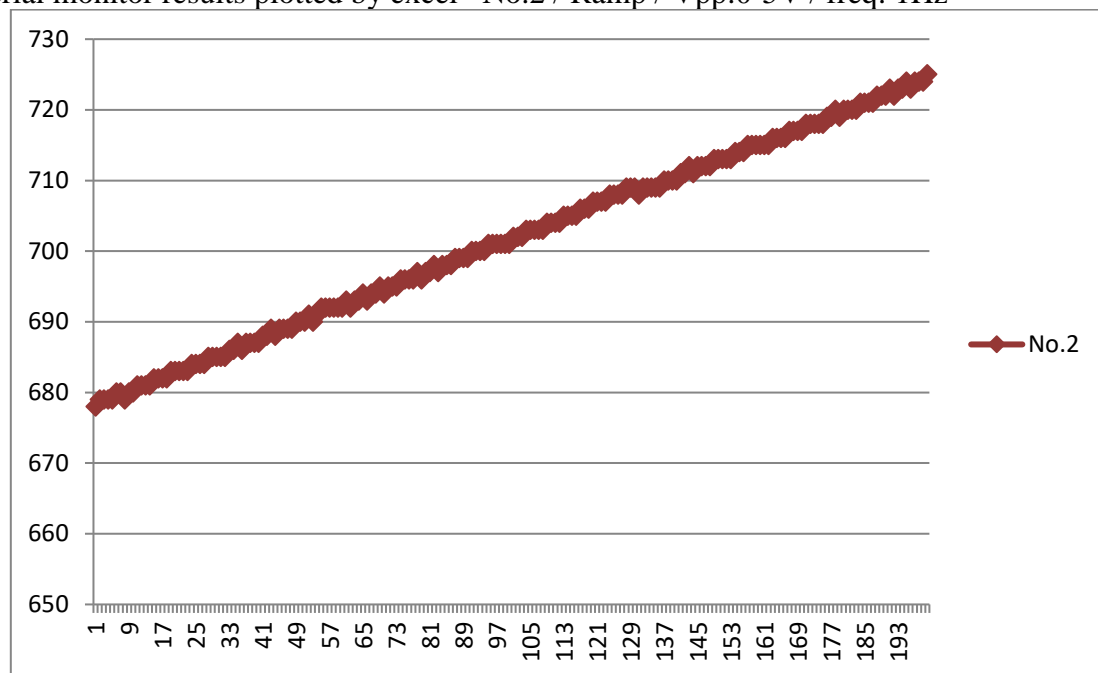
Arduino serial monitor results plotted by excel –No.1 / Ramp / V_{pp} :0-5V / freq:0.1Hz



挑選輸入類比 ramp 訊號正在爬升的階段，可以看到因為 quantization 而造成的 error，因為經過 ADC 的類比訊號一定要被放進 0~1023 之間，而在類比訊號中便可能會發生數值不同的時刻因為 ADC 解析度不夠高而被歸類在同樣的數位訊號數值中，如此一來便產生了 Quantization error。

2.

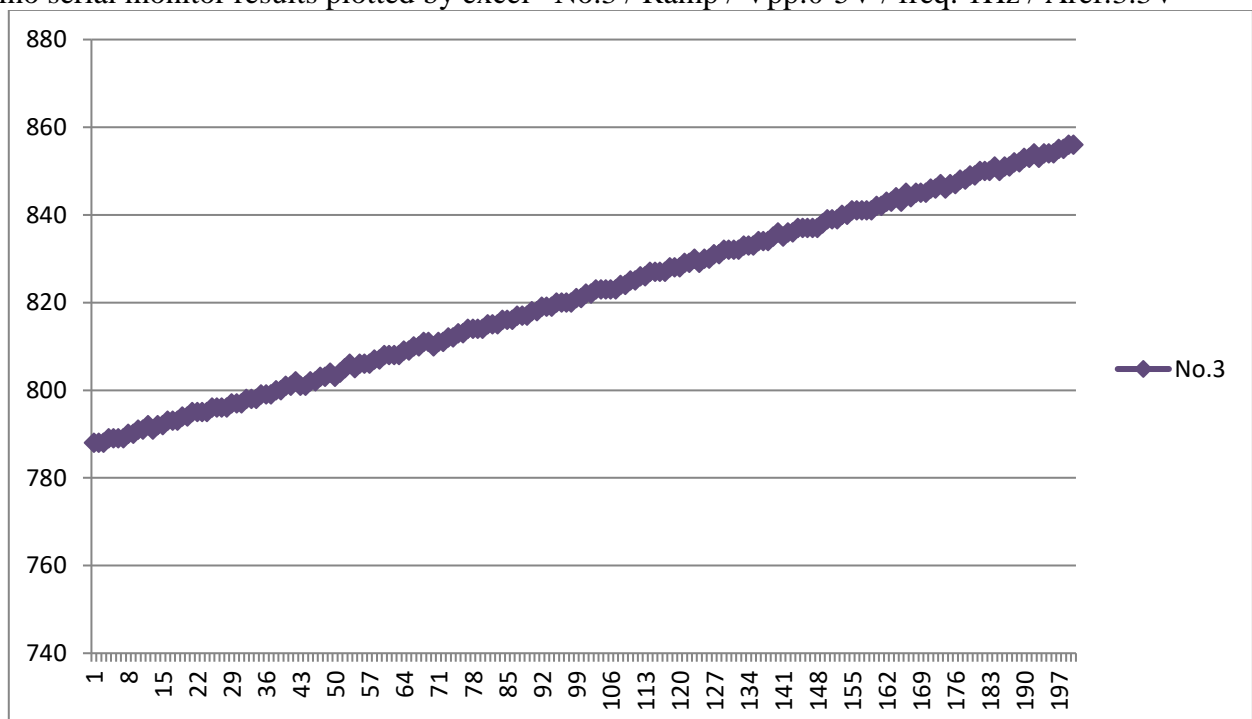
Arduino serial monitor results plotted by excel –No.2 / Ramp / Vpp:0-5V / freq: 1Hz



No.1 有點像是將上圖 No.2(頻率大 10 倍)經過放大而將 y 軸的刻度縮小以方便觀察因為量化而產生的誤差問題。而將整張圖以宏觀的方式來觀察，便會發現其實 Quantization error 其實蠻小的，仍然可以看到 ramp 的數位訊號往上爬升的趨勢。

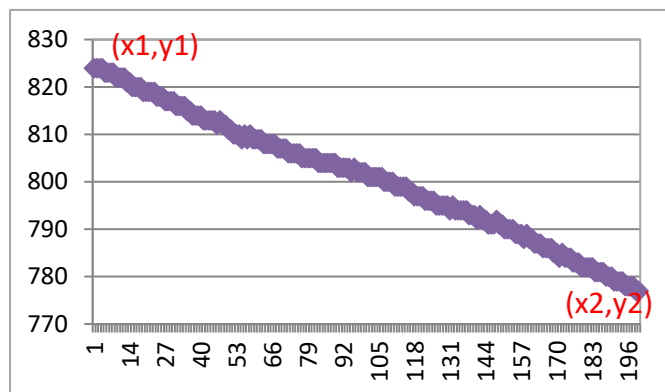
3.

Arduino serial monitor results plotted by excel –No.3 / Ramp / Vpp:0-5V / freq: 1Hz / Aref:3.3V



4.

Please calculate average LSB (Least Significant Bit) by experiment results of No.1 ~ No.3 in those intervals.
Please use the average sampling frequency f_s which you calculate from Exp1.

System sampling frequency: f_s (from exp1)

Input signal: ramp wave,

Peak to peak voltage: V_{pp} Frequency: F Hz

$$\begin{aligned}
 \text{Average LSB voltage} &= \frac{\text{Voltage Interval}}{\text{Digital Interval}} \\
 &= \frac{(V_{PP} \times 2F) \times \frac{x_2 - x_1}{f_s}}{|y_2 - y_1|}
 \end{aligned}$$

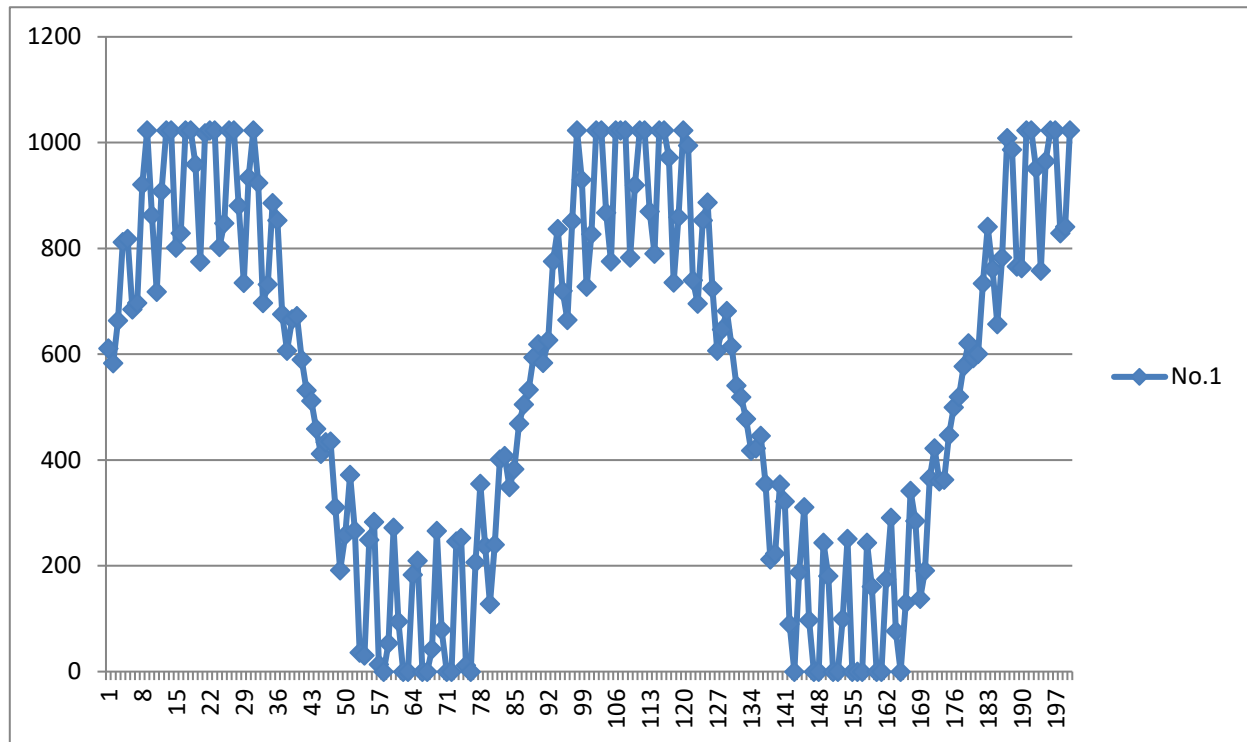
No.	1	2	3
Average LSB voltage	3.75mV	4.79mV	3.31mV

F	0.1	1	1
X2	200	200	200
X1	1	1	1
Y2	744	678	788
Y1	750	725	856
f_s	8833	8833	8833
LSB	0.00375	0.00479	0.00331

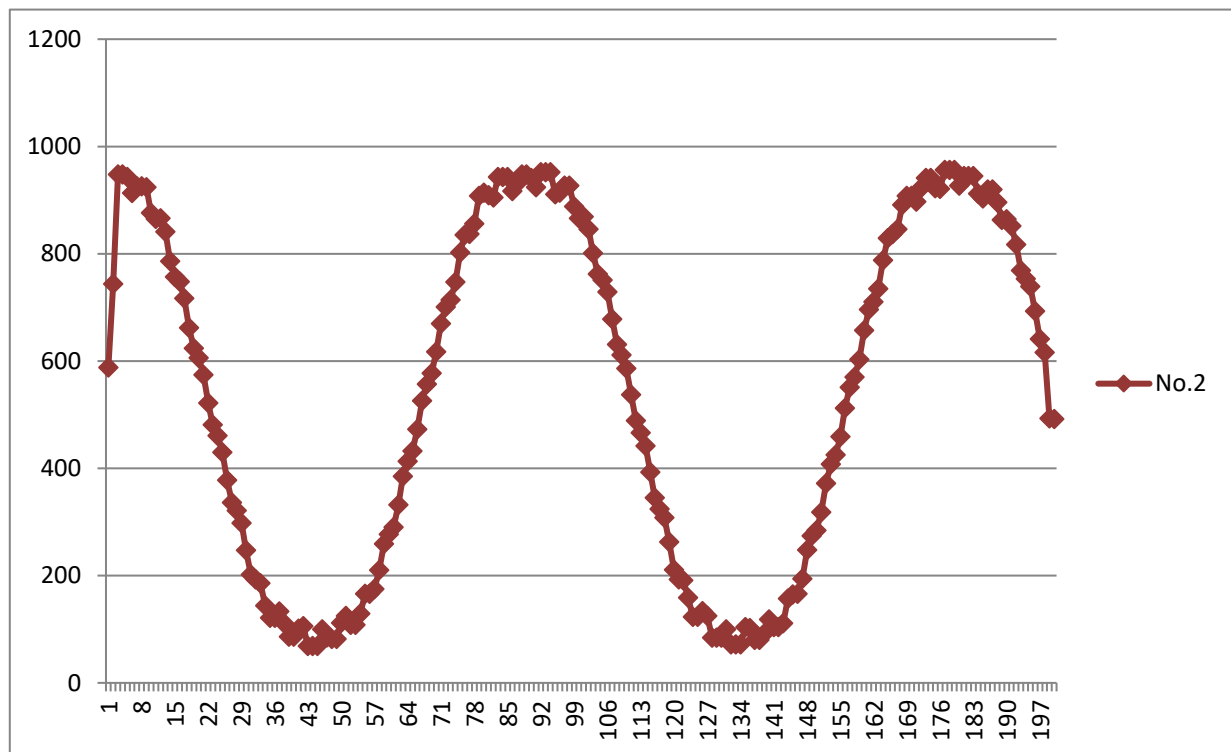
透過這個 lab，我們成功地觀察到了 ADC 因為需要將類比訊號量化到 0~1023 的數值中而產生的量化誤差，也計算了 Average Least Significant Bit 為多少。

Experiment 4: Moving average filter.

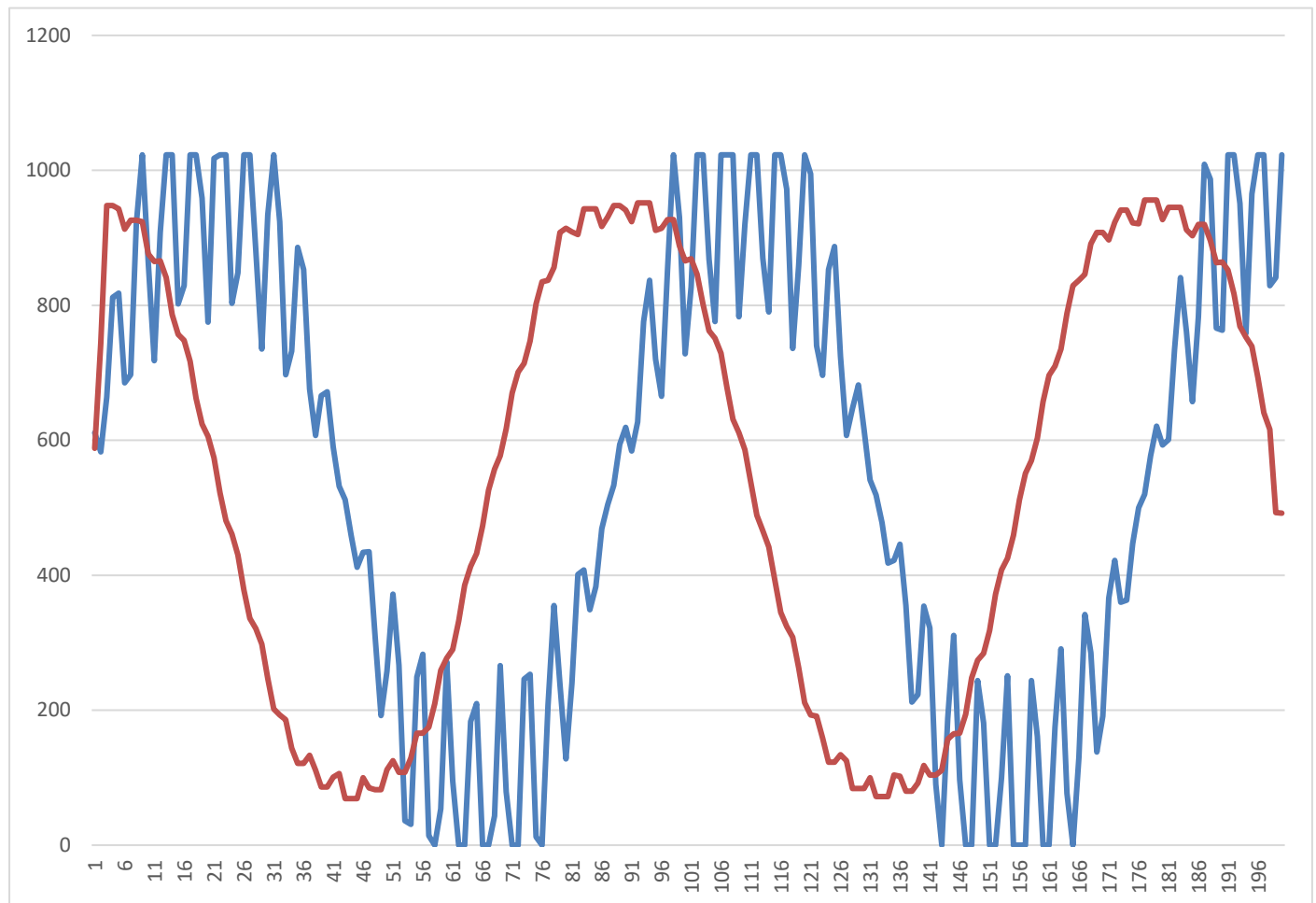
Excel plot (Before filter)



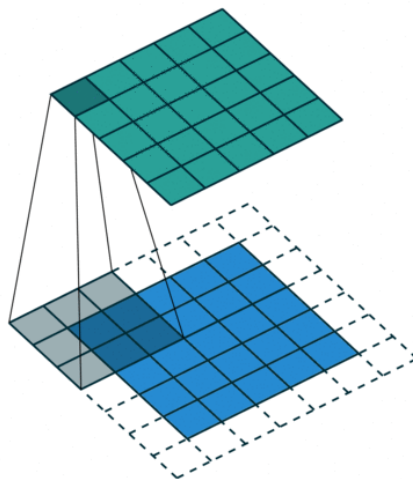
Excel plot (After filter)



Excel plot (Both condition)



在這個實驗中，我們將 sine 波透過 AM 調變的方式將其視為有高頻的雜訊，並且我們透過訊號與系統學過的 moving average 以將訊號變得比較平滑，以得到類似原本的載波訊號。將因為高頻訊號產生的高低起伏以平均的方式去掉，但是會使訊號的解析度變差，有點像是濾鏡下的網美，透過濾鏡把臉上的痘痘變得比較不明顯，但是照片的解析度就會下降。在機器學習中的 CNN 在做 padding 的時候，運用的原理就有點像是 moving average。因此我猜測那些照片的濾鏡應該也是使用類似的 moving average 所做出來的，不管是灰階還是銳化抑或是抽去特定的特徵。



所以我就自己上網亂查，真的做了個簡單的吉娃娃濾鏡，希望能夠減少吉娃娃在世人眼裡的討厭，使其得到喜愛。原理為將一維的 moving average 變成二維的 moving weighted average，也就是 padding

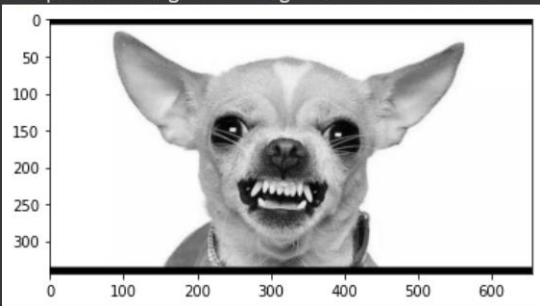
```
[52] from google.colab import drive
import os
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
```

```
[53] from google.colab import drive
drive.mount('/content/drive')
path = '/content/drive/MyDrive/'
os.chdir(path)
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

```
img = np.array(Image.open("dog.jpg").convert('L'))
plt.imshow(img, cmap = 'gray')
```



<matplotlib.image.AxesImage at 0x7ff522ea9750>



```
from scipy.signal import convolve2d
def show_difference(image, kernal):
    convolved = convolve2d(image, kernal)
    fig = plt.figure(figsize = (8, 8))
    plt.subplot(121)
    plt.title('Original Image')
    plt.axis('off')
    plt.imshow(image, cmap = 'gray')

    plt.subplot(122)
    plt.title('Convolved Image')
    plt.axis('off')
    plt.imshow(convolved, cmap = 'gray')
    return convolved
```

```
kernal = np.array([[1, 0, 1],
                  [-2, 0, 2],
                  [-1, 0, 1]], np.float32)
dx = show_difference(img, kernal)
```

Original Image	Convolved Image
	

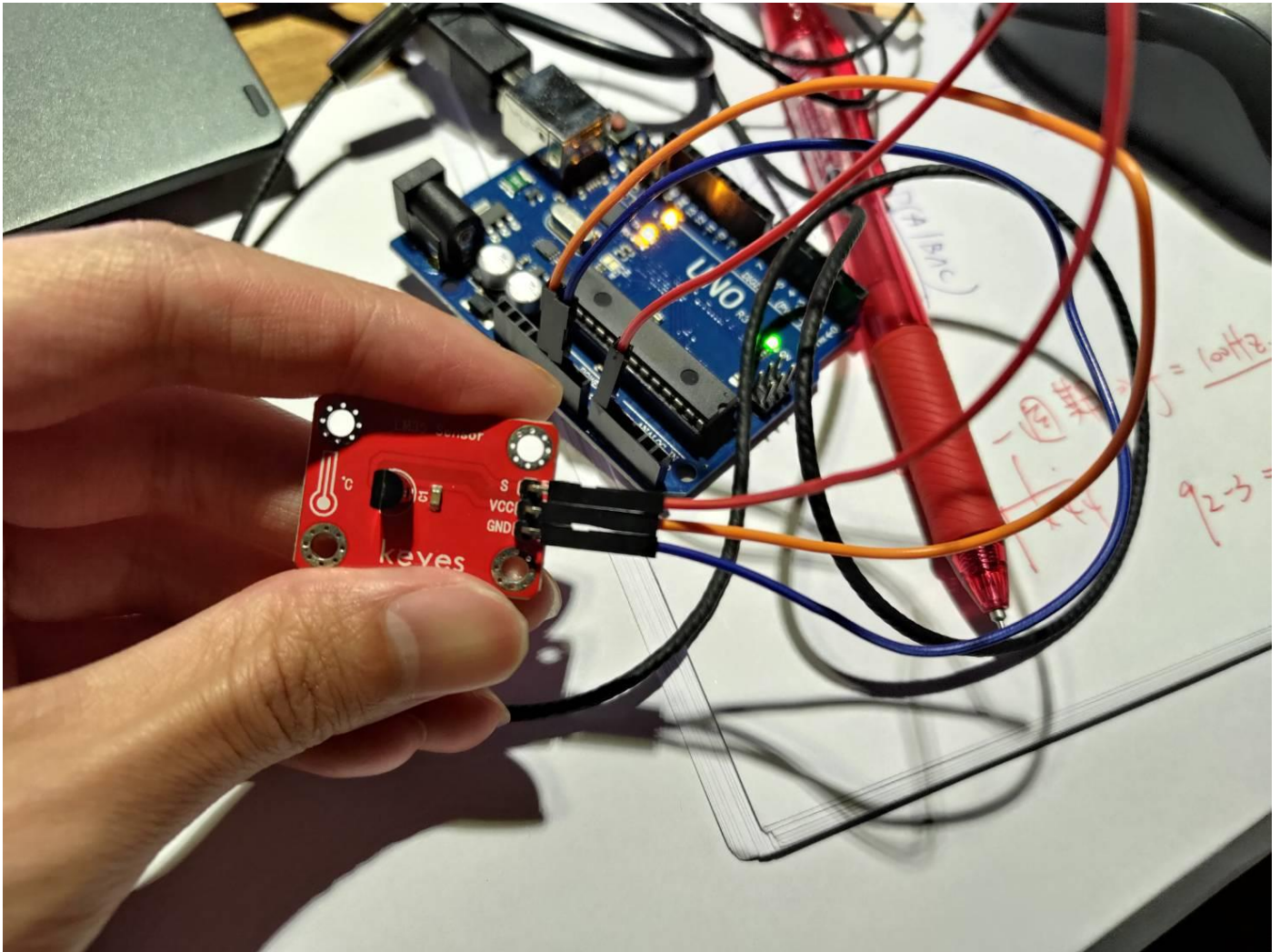
在經過自製濾鏡之後的吉娃娃是不是就比較討喜了呢?似乎也不會有那麼多人討厭他了!這個 filter 就是拿 3x3 的矩陣對原始吉娃娃的圖片做類似 moving average 的 padding 過程，在這之中將原始圖片的解析度降低，但是也使得照片看起來比較不那麼銳利，因此便達到我的目的:使吉娃娃變可愛。

Reference:

<https://syshen.medium.com/%E5%85%A5%E9%96%80%E6%B7%B1%E5%BA%A6%E5%AD%B8%E7%BF%92-2-d694cad7d1e5>

Experiment 5: Analog temperature sensor

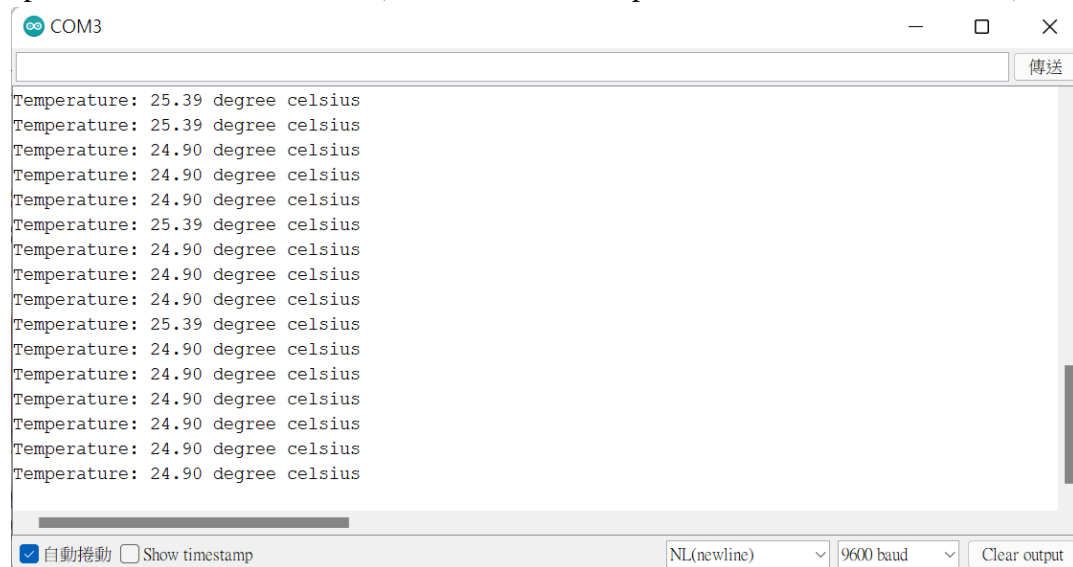
The circuit diagram of your design: (label every port clearly)



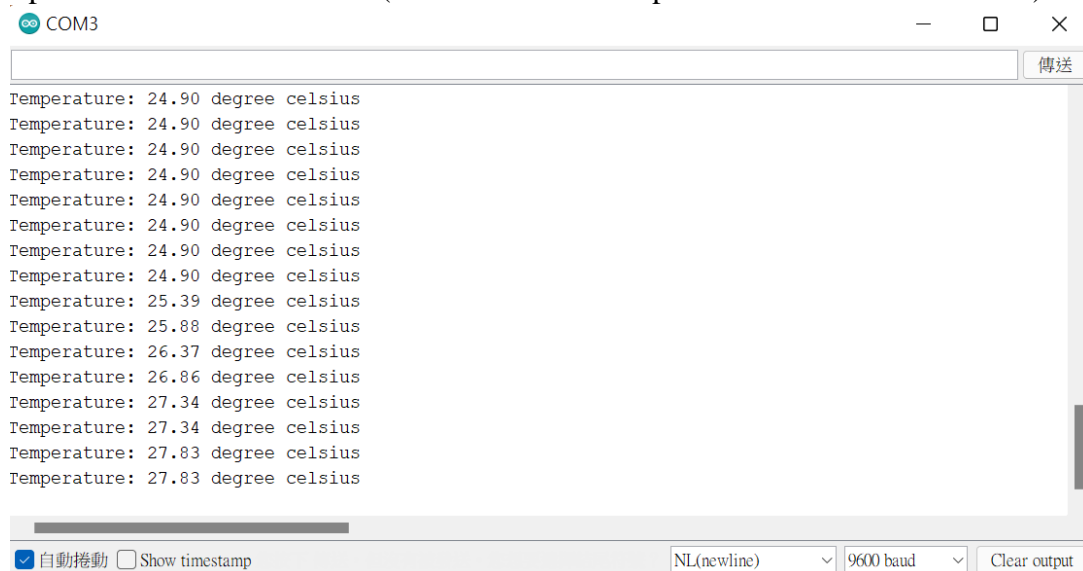
The sketch of your design: (copy from the Arduino IDE window and paste here)

```
1  float Temperature;
2
3  void setup() {
4      Serial.begin(9600);
5      pinMode(A0, INPUT);
6  }
7
8  void loop() {
9      Temperature = analogRead(A0);
10     Temperature = (Temperature / 1024 * 5) / 0.01;
11     Serial.print("Temperature: ");
12     Serial.print(Temperature);
13     Serial.println(" degree celsius");
14     delay(300);
15 }
```

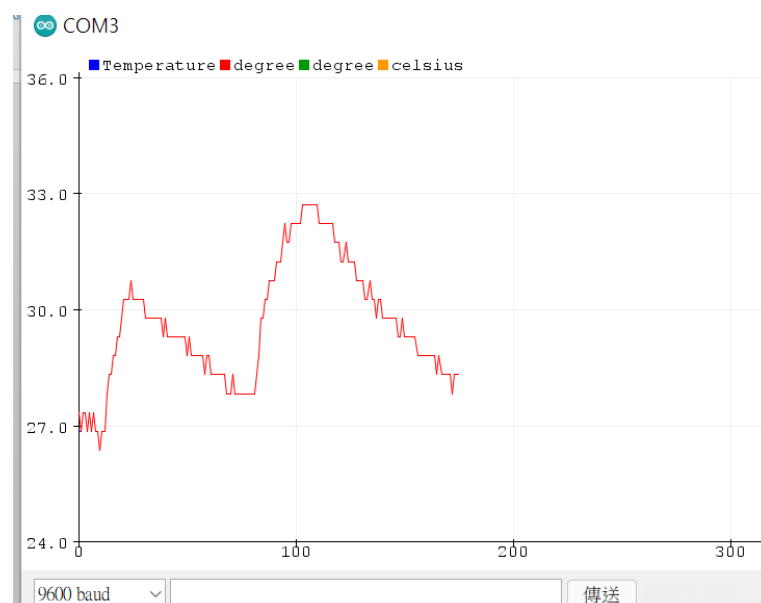
The screen capture of the serial monitor: (show the room temperature value on the window)



The screen capture of the serial monitor: (show the heated temperature value on the window)

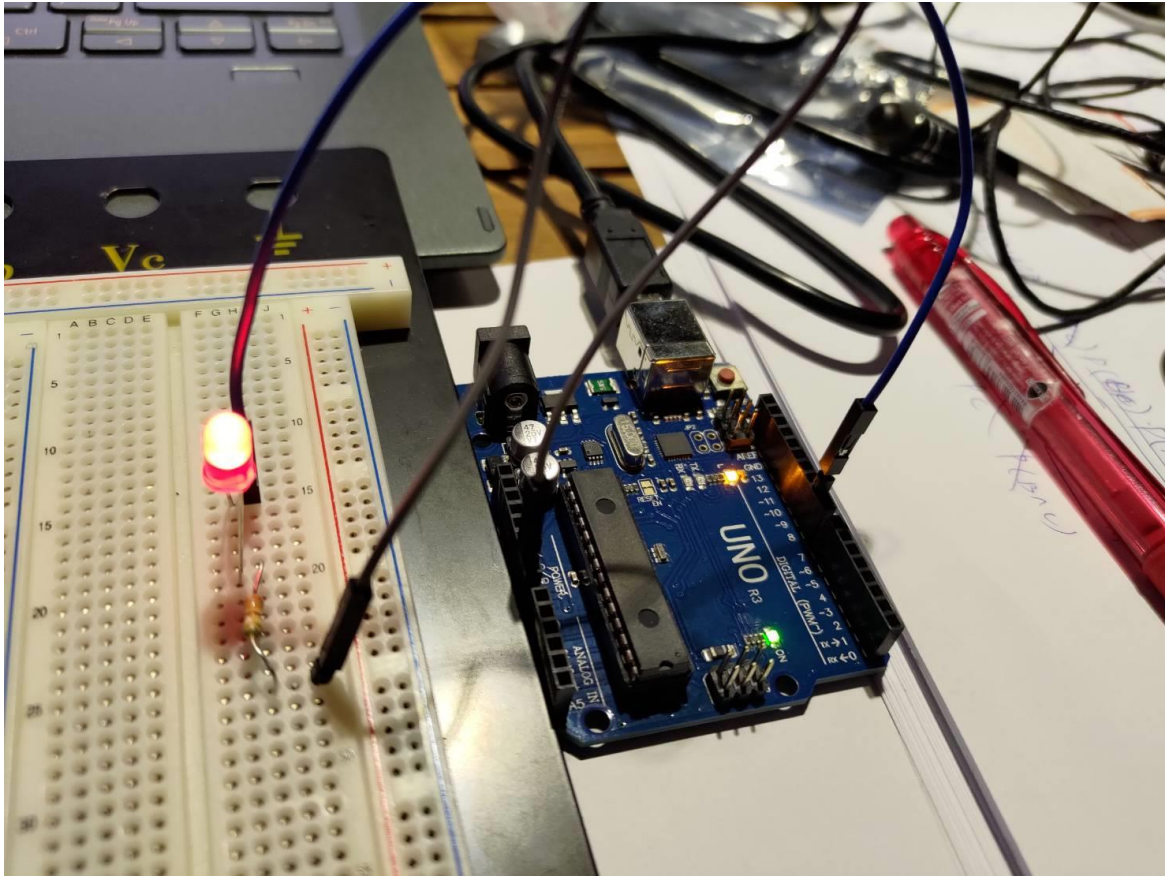


The screen capture of the serial plotter: (show the change of the temperature)



Experiment 6: Breathing light

The circuit diagram of your design: (label every port clearly)



The sketch of your design: (copy from the Arduino IDE window and paste here)

```
1  int brightness = 0;
2  int stepAmount = 5;
3
4  void setup() {
5      pinMode(9, OUTPUT);
6  }
7
8  void loop() {
9      analogWrite(9, brightness);
10     brightness += stepAmount;
11     if(brightness <= 0 || brightness >= 255){
12         stepAmount = -1 * stepAmount;
13     }
14     delay(40);
15 }
```

Your demo video(both LED and waveform) link: <https://youtu.be/xrj3jt31NvM>