BRING HOME EXAM
KIE2006 SIGNAL AND SYSTEMS

Name : Muhammad Hafiz Bin Zulkepli

Old matrix no. : KIE190053

New matrix no. : 17082403/2

Group : 1

Appendix

Question 1a:

```
%Retrieve recorded audio
clear all; close all; clc
[y,fs]=audioread('recorded.wav'); %Recorded voice signal and 1kHz noise
tone.
info=audioinfo('recorded.wav');
fs=44100;

%Plot recorded audio in time domain
t = 0:seconds(1/fs):seconds(info.Duration);
t = t(1:end-1);
figure(1); subplot(2,1,1); plot(t,y);
title('Original Sampling rate 44.1kHz')
xlabel('Time'); ylabel('Audio Signal')

%Plot recorded audio in frequency domain (FFT)
Y = fft(y);
L = info.TotalSamples;
P2 = abs(Y/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);
f = fs*(0:(L/2))/L;
figure(1); subplot(2,1,2); semilogx(f,20*log10(P1))
title('Fast Fourier Transform')
xlim([0 44100]); grid on; xlabel('Frequency'); ylabel('Magnitude (dB)')

%Plotting graph 44.1kHz and sampling at 44.1kHz
disp("Press 'CONTINUE' to play original rate 44.1kHz")
keyboard
sound(y,fs); %Playing recorded sound at original rate 44.1kHz
figure(2)
subplot(5,1,1)
plot((0:length(y)-1)/fs,y)
title('Original Sampling rate 44.1kHz')
xlabel('Time'); ylabel('Audio Signal')

%Plotting graph 48kHz and sampling at 48kHz
disp("Playing sound at original rate 44.1kHz, press 'CONTINUE' to play in
48kHz")
keyboard
[P,Q]=rat(48000/fs);
abs(P/Q*fs-48000);
xnew = resample(y,P,Q);
audiowrite('48k.wav',xnew,48000)
sound(xnew,48000); %Playing at 48kHz sample rate
subplot(5,1,2)
plot((0:length(xnew)-1)/(P/Q*fs),xnew)
xlabel('Time'); ylabel('Audio Signal')
title('Sampling rate 48kHz')

%Plotting graph 10kHz and sampling at 10kHz
disp("Playing sound at 48kHz, press 'CONTINUE' to play in 10kHz")
keyboard
[P,Q]=rat(10000/fs);
abs(P/Q*fs-10000);
xnew = resample(y,P,Q);
```

```
audiowrite('10k.wav',xnew,10000)
sound(xnew,10000); %Playing at 10kHz sample rate
subplot(5,1,3)
plot((0:length(xnew)-1)/(P/Q*fs),xnew)
xlabel('Time'); ylabel('Audio Signal')
title('Sampling rate at 10kHz')

%Plotting graph 2kHz and sampling at 2kHz
disp("Playing sound at 10kHz, press 'CONTINUE' to play in 2kHz")
keyboard
[P,Q]=rat(2000/fs);
abs(P/Q*fs-2000);
xnew = resample(y,P,Q);
audiowrite('2k.wav',xnew,2000)
sound(xnew,2000); %Playing at 2kHz sample rate
subplot(5,1,4)
plot((0:length(xnew)-1)/(P/Q*fs),xnew)
xlabel('Time'); ylabel('Audio Signal')
title('Sampling rate at 2kHz')

%Plotting graph 1kHz and sampling at 1kHz
disp("Playing sound at 2kHz, press 'CONTINUE' to play in 1kHz")
keyboard
[P,Q]=rat(1000/fs);
abs(P/Q*fs-1000);
xnew = resample(y,P,Q);
audiowrite('1k.wav',xnew,1000)
sound(xnew,1000); %Playing at 1kHz sample rate
subplot(5,1,5)
plot((0:length(xnew)-1)/(P/Q*fs),xnew)
xlabel('Time'); ylabel('Audio Signal')
title('Sampling rate at 1kHz')
disp("Playing sound at 1kHz and see all the graph in Figure 1 and Figure
2")
```

Question 1b:

```
%Retrieve recorded audio
clear all; close all; clc
[y,Fs]= audioread('recorded.wav');
info = audioinfo('recorded.wav');
t = 0:seconds(1/Fs):seconds(info.Duration);
t = t(1:end-1);
Fs=44100;

%Plot original audio in time domain
figure(1);
subplot(2,1,1); plot(t,y)
title('Original Audio with noise tone')
xlabel('Time')
ylabel('Audio Signal')

%Plot original audio in frequency domain and play original audio
L=length(y);
NEFT = 2^nextpow2(L);
Y=abs(fft(y,NEFT)/L);
freq = Fs/2*linspace(0,1,NEFT/2+1);
```

```
subplot(2,1,2)
semilogx(freq, 20*log10(Y(1:length(freq))))
title('Fast Fourier Transform')
xlim([0 44100]); grid on; xlabel('Frequency'); ylabel('Magnitude (dB)')
sound(y,Fs) %Play original recorded audio

%Filter noise audio and plot clean audio and play clean audio
disp("Playing original recorded audio, press 'CONTINUE' to play clean
audio")
keyboard
o = 5;
wn = [800 1200]*2/Fs;
[b, a] = butter(o,wn,'stop');
figure (2);
freqz(b,a,2^14,Fs);
[h,w] = freqz(b,a,2^14,Fs);
subplot(2,1,2)
semilogx(w,20*log10(abs(h)))
title('Fast Fourier Transform')
xlim([0 44100]); grid on; xlabel('Frequency'); ylabel('Magnitude (dB)')
z_filt = filtfilt(b,a,y);
subplot (2,1,1)
plot(t,z_filt)
title('Clean audio after decompose')
xlabel('Time')
ylabel('Audio Signal')
audiowrite('clean audio.wav',z_filt,Fs)
sound(z_filt,Fs) %Play clean audio

%Get the noise audio and plot noise audio and play noise audio
disp("Playing clean audio, press 'CONTINUE' to play noise audio")
keyboard
[b, a] = butter(o,wn,'bandpass');
figure (3);
freqz(b,a,2^14,Fs);
[h,w] = freqz(b,a,2^14,Fs);
subplot(2,1,2)
semilogx(w,20*log10(abs(h)))
title('Fast Fourier Transform')
xlim([0 44100]); grid on; xlabel('Frequency'); ylabel('Magnitude (dB)')
n_filt = filtfilt(b,a,y);
subplot (2,1,1)
plot(t,n_filt)
title('Noise audio')
xlabel('Time')
ylabel('Audio Signal')
audiowrite('noise audio.wav',n_filt,Fs)
sound(n_filt,Fs) %Play noise audio
disp("Playing noise audio, press 'CONTINUE' to see all the graph")

%Plot clean audio and noise audio in time domain
keyboard
figure(4); subplot(2,1,1);
plot(t,z_filt)
hold on
plot(t,n_filt,"r-")
hold off
legend(["Clean audio" "Noise audio"])
title('Decompose clean audio and noise audio')
xlabel('Time'); ylabel('Audio Signal')
```

```
%Plot spectrum of clean audio
Y1 = fft(z_filt);
L = info.TotalSamples;
P2 = abs(Y1/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);
f1 = Fs*(0:(L/2))/L;

%Plot spectrum of noise audio
Y2 = fft(n_filt);
P3 = abs(Y2/L);
P4 = P3(1:L/2+1);
P4(2:end-1) = 2*P4(2:end-1);
f2 = Fs*(0:(L/2))/L;

%Decompose clean audio and noise audio in frequency domain
subplot(2,1,2);
semilogx(f1,20*log10(P1))
hold on
semilogx(f2,20*log10(P4),"r-")
hold off
title('Fast Fourier Transform')
xlim([0 44100]); grid on; xlabel('Frequency'); ylabel('Magnitude (dB)
(dB)')
legend(["Clean audio" "Noise audio"])
```

Question 2a:

```
clear all; close all; clc
image=imread("hafiz.jpg");        %retrieve image
info=imfinfo("hafiz.jpg")
info.BitDepth
image1=rgb2gray(image);      %Convert RGB image to grayscale image
info1=imfinfo("hafiz_gray.jpg")
info1.BitDepth
imwrite(image1,"hafiz_gray.jpg")

figure
imshow(image1)           %Display the original gray image
title("Original image 3479x2248")

F = griddedInterpolant (double(image1));
[size_x,size_y,size_z] = size(image1);    %Dimensions of the image

rx1=size_x/2; %downsampling rate by 2^1 in x dimension
ry1=size_y/2; %downsampling rate by 2^1 in y dimension
x_sampled1=(1:size_x/rx1:size_x)';
y_sampled1=(1:size_y/ry1:size_y)';
z_sampled1=(1:size_z);
downsample_1=uint8(F({x_sampled1,y_sampled1}));
figure
imshow(downsample_1)
imwrite(downsample_1,"1740x1124.jpg")
title("Downsample to 1740x1124")

rx2=size_x/4;    %downsampling rate by 2^2 in x dimension
ry2=size_y/4;    %downsampling rate by 2^2 in y dimension
```

```
x_sampled2=(1:size_x/rx2:size_x)';
y_sampled2=(1:size_y/ry2:size_y)';
z_sampled2=(1:size_z);
downsample_2=uint8(F({x_sampled2,y_sampled2}));
figure
imshow(downsample_2)
imwrite(downsample_2,"870x562.jpg")
title("Downsample to 870x562")

rx3=size_x/8;     %downsampling rate by 2^3 in x dimension
ry3=size_y/8;     %downsampling rate by 2^3 in y dimension
x_sampled3=(1:size_x/rx3:size_x)';
y_sampled3=(1:size_y/ry3:size_y)';
z_sampled3=(1:size_z);
downsample_3=uint8(F({x_sampled3,y_sampled3}));
figure
imshow(downsample_3)
imwrite(downsample_3,"435x281.jpg")
title("Downsample to 435x281")

rx4=size_x/16;    %downsampling rate by 2^4 in x dimension
ry4=size_y/16;    %downsampling rate by 2^4 in y dimension
x_sampled4=(1:size_x/rx4:size_x)';
y_sampled4=(1:size_y/ry4:size_y)';
z_sampled4=(1:size_z);
downsample_4=uint8(F({x_sampled4,y_sampled4}));
figure
imshow(downsample_4)
imwrite(downsample_4,"218x141.jpg")
title("Downsample to 218x141")

rx5=size_x/32;    %downsampling rate by 2^5 in x dimension
ry5=size_y/32;    %downsampling rate by 2^5 in y dimension
x_sampled5=(1:size_x/rx5:size_x)';
y_sampled5=(1:size_y/ry5:size_y)';
z_sampled5=(1:size_z);
downsample_5=uint8(F({x_sampled5,y_sampled5}));
figure
imshow(downsample_5)
imwrite(downsample_5,"109x71.jpg")
title("Downsample to 109x71")
```

Question 2b:

```
clear all; close all; clc
image=imread("hafiz_gray.jpg");
size_image=size(image)
figure
imshow(image)  %Display the original gray image
info1=imfinfo("hafiz_gray.jpg")
title("Original image 8-bit")

%Using gray2ind function to quantize image to desired bits
[image16,map16] = gray2ind(image,16); %convert to 4bit image
[image8,map8] = gray2ind(image,8); %convert to 3bit image
[image4,map4] = gray2ind(image,4); %convert to 2bit image
[image2,map2] = gray2ind(image,2); %convert to 1bit image
```

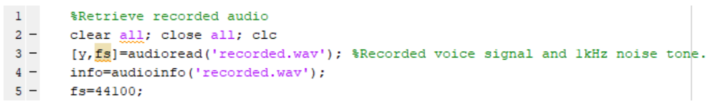
```
%Show all the quantize image
figure
imshow(image16,map16)
title("4-bit of image")
figure
imshow(image8,map8)
title("3-bit of image")
figure
imshow(image4,map4)
title("2-bit of image")
figure
imshow(image2,map2)
title("1-bit of image")

%Save all the quantized image
imwrite(image,"8bit.jpg")
imwrite(image16,map16,"4bit.jpg")
imwrite(image8,map8,"3bit.jpg")
imwrite(image4,map4,"2bit.jpg")
imwrite(image2,map2,"1bit.jpg")
```

Question 1a: Design of solution

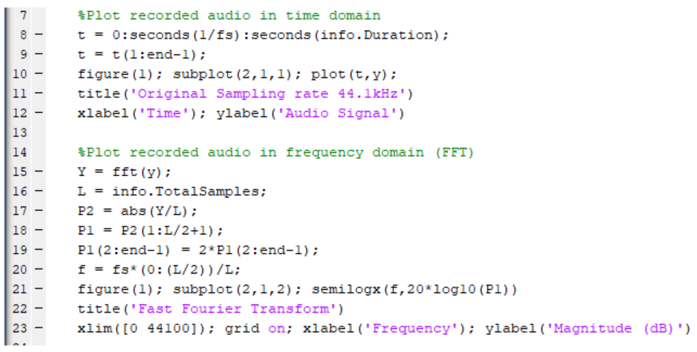
Audio signal is an analogue signal which continuous in amplitude and time. In electronic devices, the data process digitally, hence, the analogue signal need to convert to digital signal using converter. Inside the converter, the process of changing the signal is called sampling.

In this assignment, the voice saying “My name is Muhammad Hafiz Bin Zulkepli” and a noise tone with 1kHz is recorded in “recorded.wav” file for 5 seconds. The recorded voice is recorded at a sample rate 44.1kHz , 16-bit rate audio and in mono audio or mono channel. Then, the recorded audio is read using code below.



```
%Retrieve recorded audio
clear all; close all; clc
[y,fs]=audioread('recorded.wav'); %Recorded voice signal and 1kHz noise tone.
info=audioinfo('recorded.wav');
fs=44100;
```

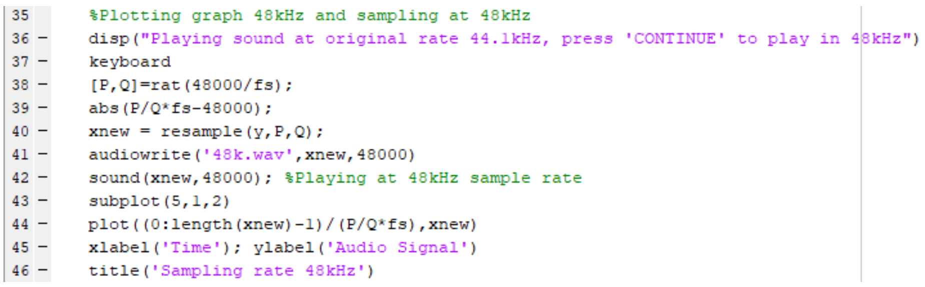
After that, the audio is plotted in time domain and frequency domain (FFT) to investigate the behaviour of the signal as shown in code below. For FFT, we simply used ‘fft’ to convert the time domain to fast fourier transform.



```
%Plot recorded audio in time domain
t = 0:seconds(1/fs):seconds(info.Duration);
t = t(1:end-1);
figure(1); subplot(2,1,1); plot(t,y);
title('Original Sampling rate 44.1kHz')
xlabel('Time'); ylabel('Audio Signal')

%Plot recorded audio in frequency domain (FFT)
Y = fft(y);
L = info.TotalSamples;
P2 = abs(Y/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);
f = fs*(0:(L/2))/L;
figure(1); subplot(2,1,2); semilogx(f,20*log10(P1))
title('Fast Fourier Transform')
xlim([0 44100]); grid on; xlabel('Frequency'); ylabel('Magnitude (dB)')
```

In order to sampling the signal, we can simply use ‘resample’. For upsampling 48kHz, `resample(y,P,Q)` resamples the input sequence, y, at P/Q times the original sample rate and ‘resample’ applies an FIR Antialiasing Lowpass Filter to y and compensates for the delay introduced by the filter. The audio is played and save in ‘.wav’ file. Then the graph of the sampling rate is plotted as shown in code below. For downsampling, we sample at 10kHz, 2kHz and 1kHz and it used the same method for upsampling where the ratio of the P/Q is manipulated.



```
%Plotting graph 48kHz and sampling at 48kHz
disp("Playing sound at original rate 44.1kHz, press 'CONTINUE' to play in 48kHz")
keyboard
[P,Q]=rat(48000/fs);
abs(P/Q*fs-48000);
xnew = resample(y,P,Q);
audiowrite('48k.wav',xnew,48000)
sound(xnew,48000); %Playing at 48kHz sample rate
subplot(5,1,2)
plot((0:length(xnew)-1)/(P/Q*fs),xnew)
xlabel('Time'); ylabel('Audio Signal')
title('Sampling rate 48kHz')
```

Question 1a: Results and discussion

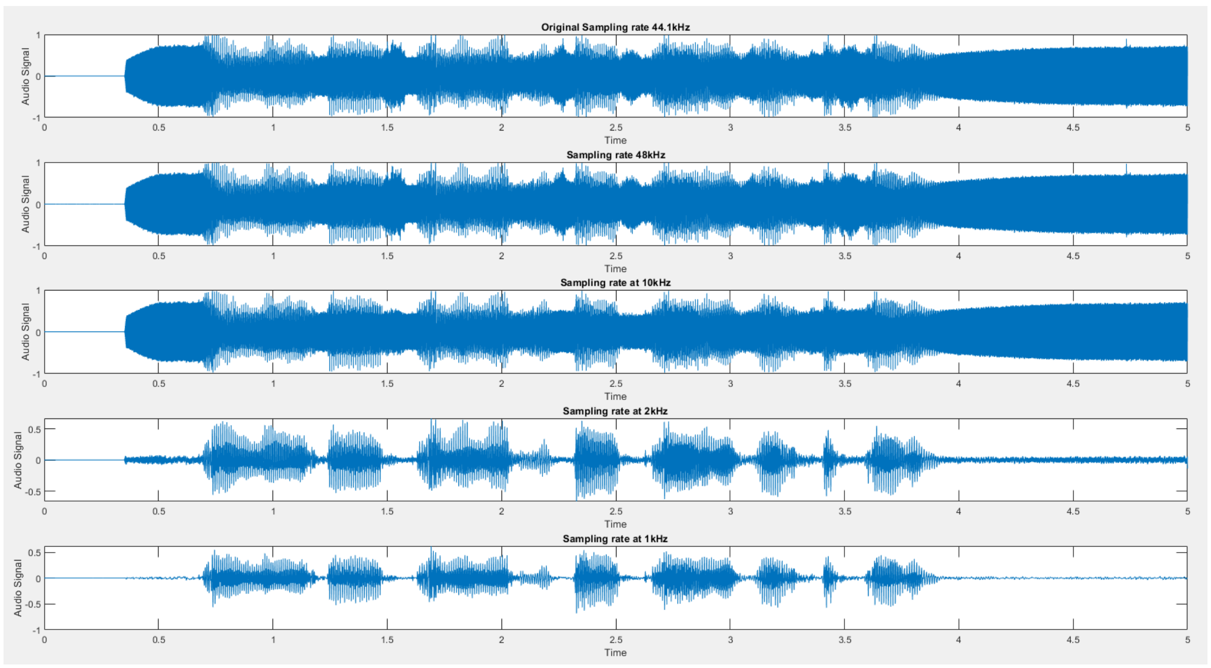


Figure shows the sampling rate at 44.1kHz, 48kHz, 10kHz, 2kHz and 1kHz

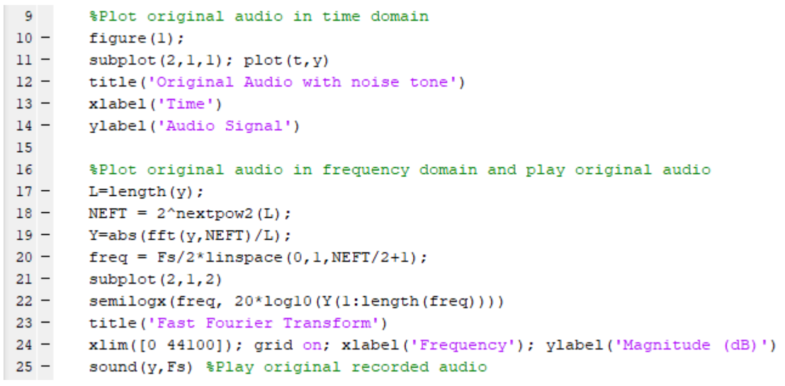
Sampling rate (kHz)	Discussion
44.1	-The quality of the audio is good and can hear the audio clearly. -The size of the audio is 430kb.
48	-The quality of the audio is good and can hear the audio clearly. -The different of the audio is unnoticeable between 44.1kHz and 48kHz sampling rate. -The size of the audio is 468kb
10	-The volume of the audio and the quality of the audio slightly reduced -The size of the audio is 97.6kb
2	-The volume of the audio and the quality of the audio moderately reduced -The size of the audio is 19.5kb
1	-The volume of the audio and the quality of the audio largely reduced -The size of the audio is 9.8kb

The sample rate 44.1khz is common sampling frequency in many audio devices. This is because 20kHz is the highest frequency audible by human, and based on Nyquist theorem the sample frequency should be twice the maximum frequency to avoid aliasing and also based on this theorem, it states that a signal can be exactly reconstruct from the sample. Moreover, further increase the sampling frequency above 44.1kHz will make the size of the audio bigger and make the audio waveform smoothier but the different is unnoticeable. On the other hand, if we downsampling the signal below Nyqueist rate, it will reduce the quality of the audio and reduce the size of the audio as well.

Most of the audio converter use a sample rate 44.1kHz and bit depth of 16-bit. Any music file recorded higher than this value is considered high definition (HD) audio and will take more memory to save it.

Question 1b: Design of solution

The recorded audio has noise with 1kHz noise tone. The recorded audio is plotted in time domain and frequency domain to investigate the behaviour of the signal , then, the recorded audio is played as shown in code below.

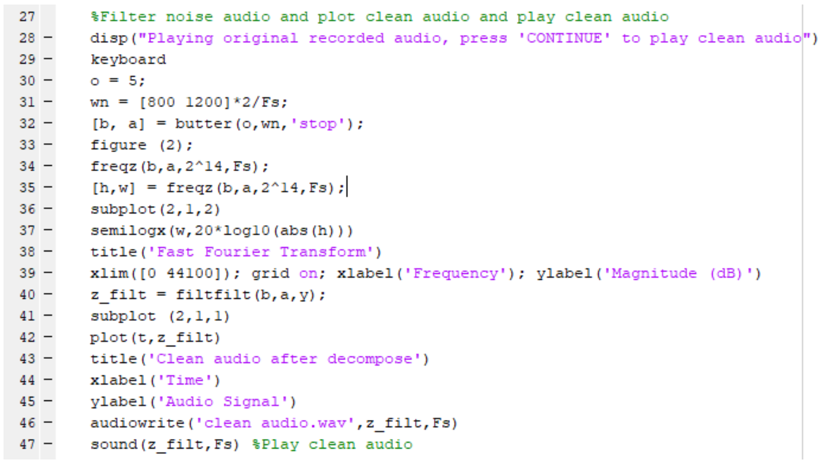


```
%Plot original audio in time domain
figure(1);
subplot(2,1,1); plot(t,y)
title('Original Audio with noise tone')
xlabel('Time')
ylabel('Audio Signal')

%Plot original audio in frequency domain and play original audio
L=length(y);
NEFT = 2^nextpow2(L);
Y=abs(fft(y,NEFT)/L);
freq = Fs/2*linspace(0,1,NEFT/2+1);
subplot(2,1,2)
semilogx(freq, 20*log10(Y(1:length(freq))))
title('Fast Fourier Transform')
xlim([0 44100]); grid on; xlabel('Frequency'); ylabel('Magnitude (dB)')
sound(y,Fs) %Play original recorded audio
```

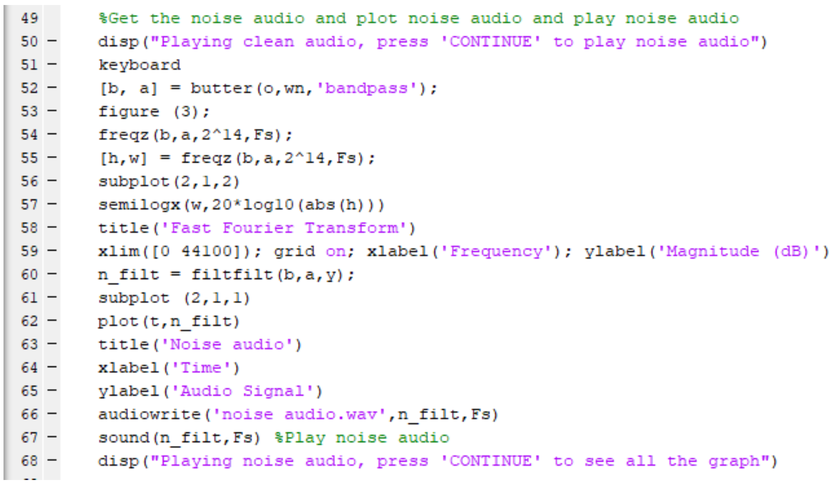
To decompose the noise audio and clean audio, one's should know what type of noise and characteristics of the noise. In this case, the noise is set to be 1kHz tone and has certain frequency only rather than all frequency. So, we apply Fast Fourier Transform to see the noise frequency and using bandreject filter to filter out the noise tone at 1kHZ. After that, we successfully filter out the noise audio. Then, to get the noise audio, we can simply use bandpass filter to passing the noise audio only.

This code below used butterworth bandreject filter of order 5 at the range of 800Hz until 1200Hz to filter out the noise audio and also will plot the clean audio in time domain and frequency domain, play clean audio and save the audio in ".wav" format.



```
%Filter noise audio and plot clean audio and play clean audio
disp("Playing original recorded audio, press 'CONTINUE' to play clean audio")
keyboard
o = 5;
wn = [800 1200]*2/Fs;
[b, a] = butter(o,wn,'stop');
figure (2);
freqz(b,a,2^14,Fs);
[h,w] = freqz(b,a,2^14,Fs);
subplot(2,1,2)
semilogx(w,20*log10(abs(h)))
title('Fast Fourier Transform')
xlim([0 44100]); grid on; xlabel('Frequency'); ylabel('Magnitude (dB)')
z_filt = filtfilt(b,a,y);
subplot (2,1,1)
plot(t,z_filt)
title('Clean audio after decompose')
xlabel('Time')
ylabel('Audio Signal')
audiowrite('clean audio.wav',z_filt,Fs)
sound(z_filt,Fs) %Play clean audio
```

This code below used butterworth bandpass filter of order 5 at the range of 800Hz until 1200Hz to pass through the noise audio only. This code below also will plot the noise audio in time domain and frequency domain, play noise audio and save the audio in “.wav” format.



```
%Get the noise audio and plot noise audio and play noise audio
disp("Playing clean audio, press 'CONTINUE' to play noise audio")
keyboard
[b, a] = butter(o,wn,'bandpass');
figure (3);
freqz(b,a,2^14,Fs);
[h,w] = freqz(b,a,2^14,Fs);
subplot(2,1,2)
semilogx(w,20*log10(abs(h)))
title('Fast Fourier Transform')
xlim([0 44100]); grid on; xlabel('Frequency'); ylabel('Magnitude (dB)')
n_filt = filtfilt(b,a,y);
subplot (2,1,1)
plot(t,n_filt)
title('Noise audio')
xlabel('Time')
ylabel('Audio Signal')
audiowrite('noise audio.wav',n_filt,Fs)
sound(n_filt,Fs) %Play noise audio
disp("Playing noise audio, press 'CONTINUE' to see all the graph")
```

Question 1b: Result and discussion

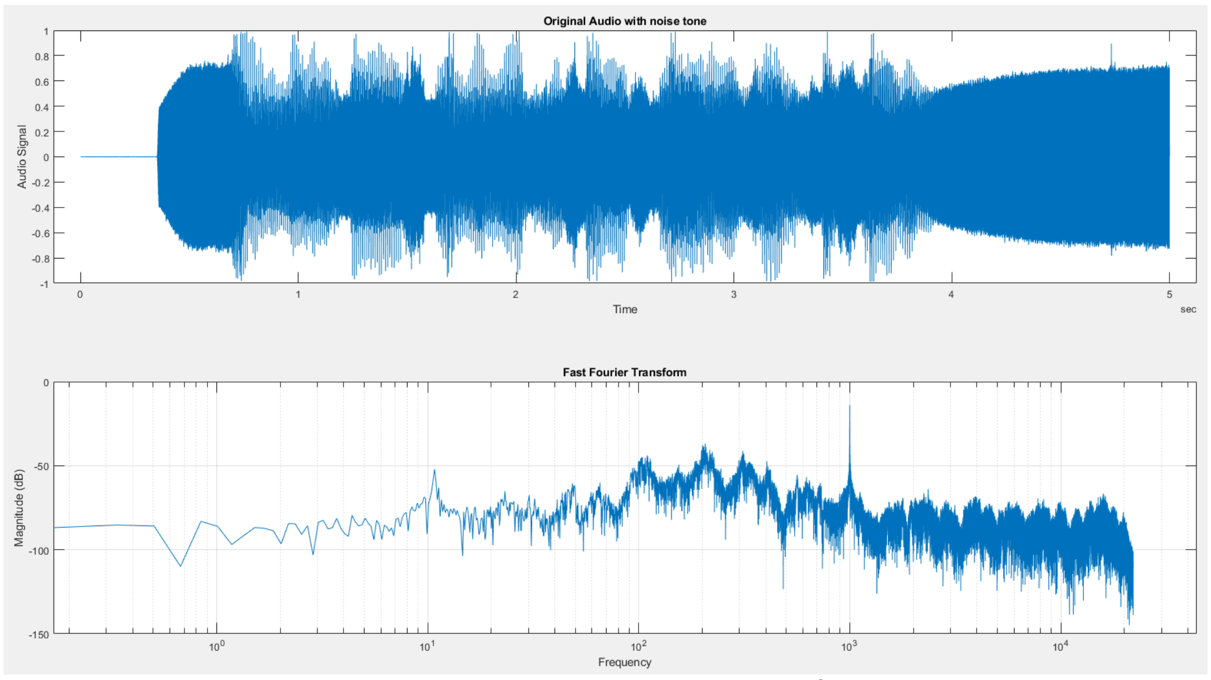


Figure 1b.1 shows the original audio in time domain and frequency domain

In time domain, we cannot differentiate noise signal and clean audio signal. After applying Fast Fourier Transform, we can see that a spike at 1kHz which refer to the noise tone 1kHz. For this type of noise, we can simply use bandreject filter to filter out the noise signal.

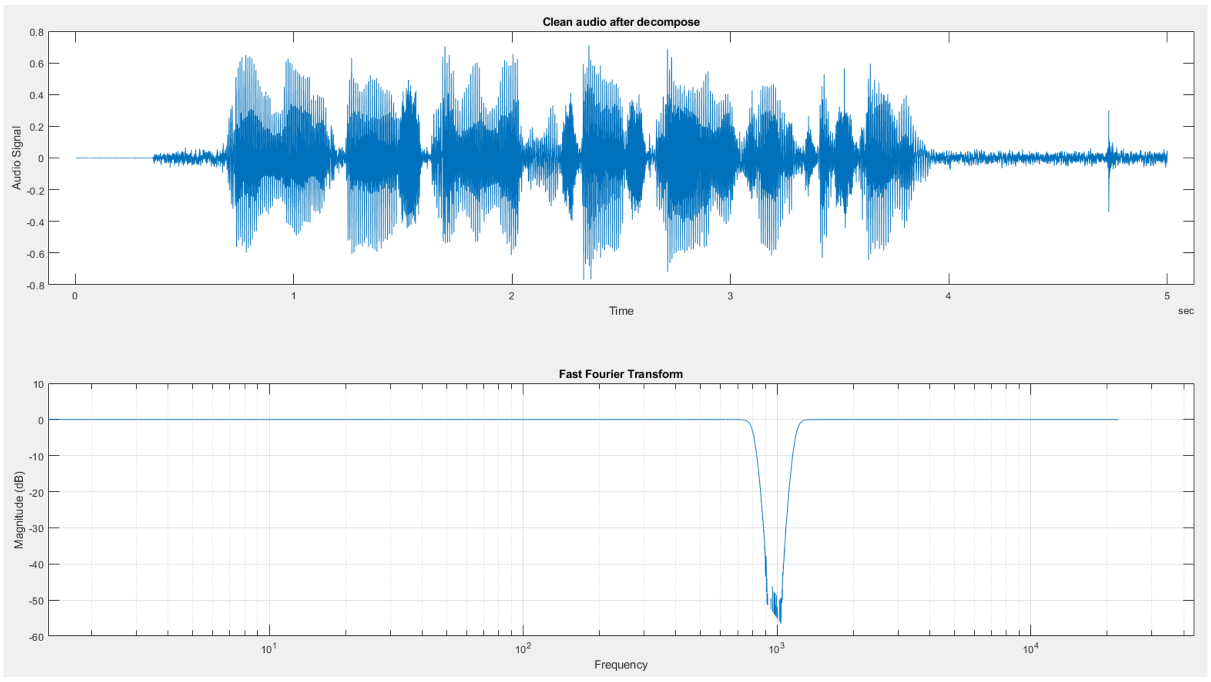


Figure 1b.2 shows the clean audio in time domain and frequency domain

In Figure 1b.2, after applying butterworth bandreject filter of order 5 at the range of 800Hz and 1200Hz, we can see that the clean audio signal is different from original audio signal. Also, the frequency at 1kHZ noise signal is removed from the frequency domain.

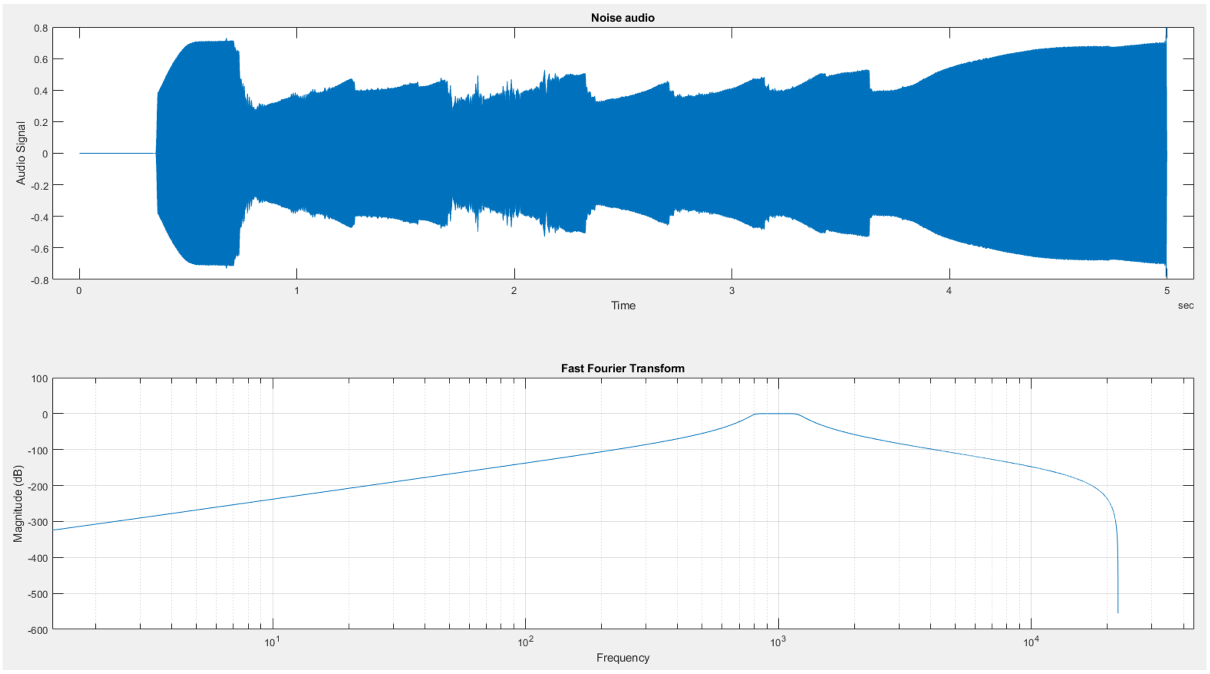


Figure 1b.3 shows the noise audio in time domain and frequency domain

In Figure 1b.3, after applying butterworth bandpass filter of order 5 at the range of 800Hz and 1200Hz, we can see that the noise audio signal has consistent amplitude and frequency, this is because the noise audio is noise tone at 1kHz.

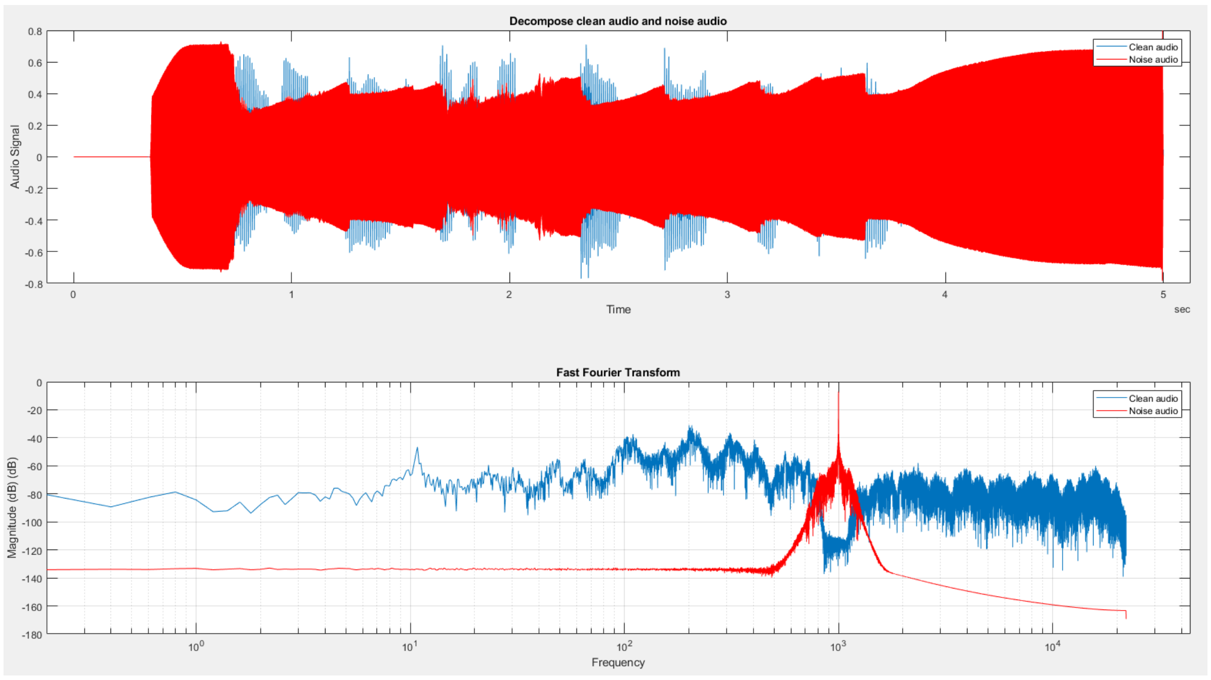


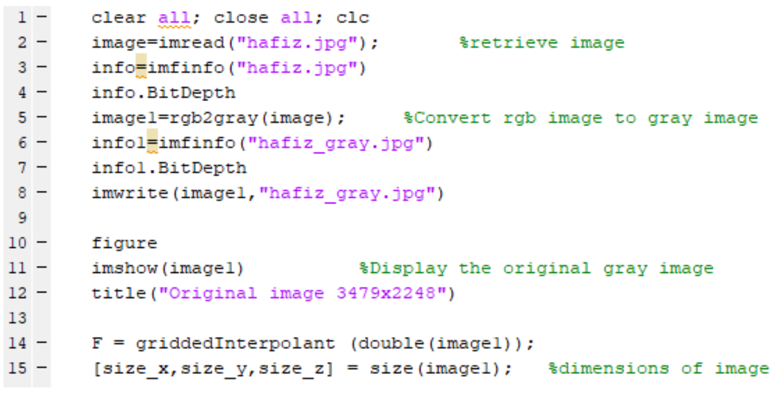
Figure 1b.4 shows the decompose noise audio and clean audio in time domain and frequency domain

Question 2a: Design of solution

For digital image processing, an image function f(x,y) must be digitized both spatially and in amplitude. Commonly, a frame grabber or digitizer is used to sample and quantize the analogue image signal. Hence, to discritize the image, we need to convert continuous data into digital using sampling and quantization technique.

For this section, let focus on sampling first, quantization will be explain in question 2b. Since the image is continuous in x and y axis, hence, it needs to give a certain value to digitize it. The sampling rate work by digitize the (x,y) coordinate of the continuous image and this will determine the spatial resolution of the digitized image. In addition, downsampling of the image will make the image smaller from the original and when resize it to the original size, the image will be blur.

In this assignment, the image of the student card is read and the information of the image is stored such as the dimensions and the bit depth. After that, the image is converted to grayscale which the value of each pixel is a single sample representing only an amount of light which carriers only intensity information as shown in the code below. The bit depth is remain constant at 8bit.

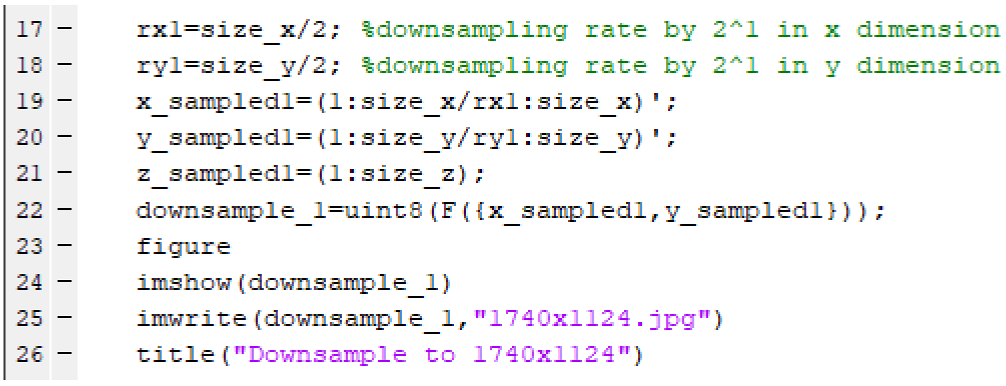


```
clear all; close all; clc
image=imread("hafiz.jpg");       %retrieve image
info=imfinfo("hafiz.jpg")
info.BitDepth
image1=rgb2gray(image);      %Convert rgb image to gray image
info1=imfinfo("hafiz_gray.jpg")
info1.BitDepth
imwrite(image1,"hafiz_gray.jpg")

figure
imshow(image1)          %Display the original gray image
title("Original image 3479x2248")

F = griddedInterpolant (double(image1));
[size_x,size_y,size_z] = size(image1);   %dimensions of image
```

After that, we downsampling the signal by divide the original size with 2 in both dimensions in x and y as shown in code below. Then, we further downsampling by divide the original size with 8, 16 and 32 in x and y dimensions. Then, the downsampling image is shown and save in jpg format.



```
rx1=size_x/2; %downsampling rate by 2^1 in x dimension
ry1=size_y/2; %downsampling rate by 2^1 in y dimension
x_sampled1=(1:size_x/rx1:size_x)';
y_sampled1=(1:size_y/ry1:size_y)';
z_sampled1=(1:size_z);
downsample_1=uint8(F({x_sampled1,y_sampled1}));
figure
imshow(downsample_1)
imwrite(downsample_1,"1740x1124.jpg")
title("Downsample to 1740x1124")
```

Question 2a: Results and discussion

All the images below are in the original size without resizing the image. Using downsampling rate at 2,4,8,16 and 32, the image will have 3479x2248, 1740x1124, 870x562, 435x281, 218x141 and 109x71 respectively.

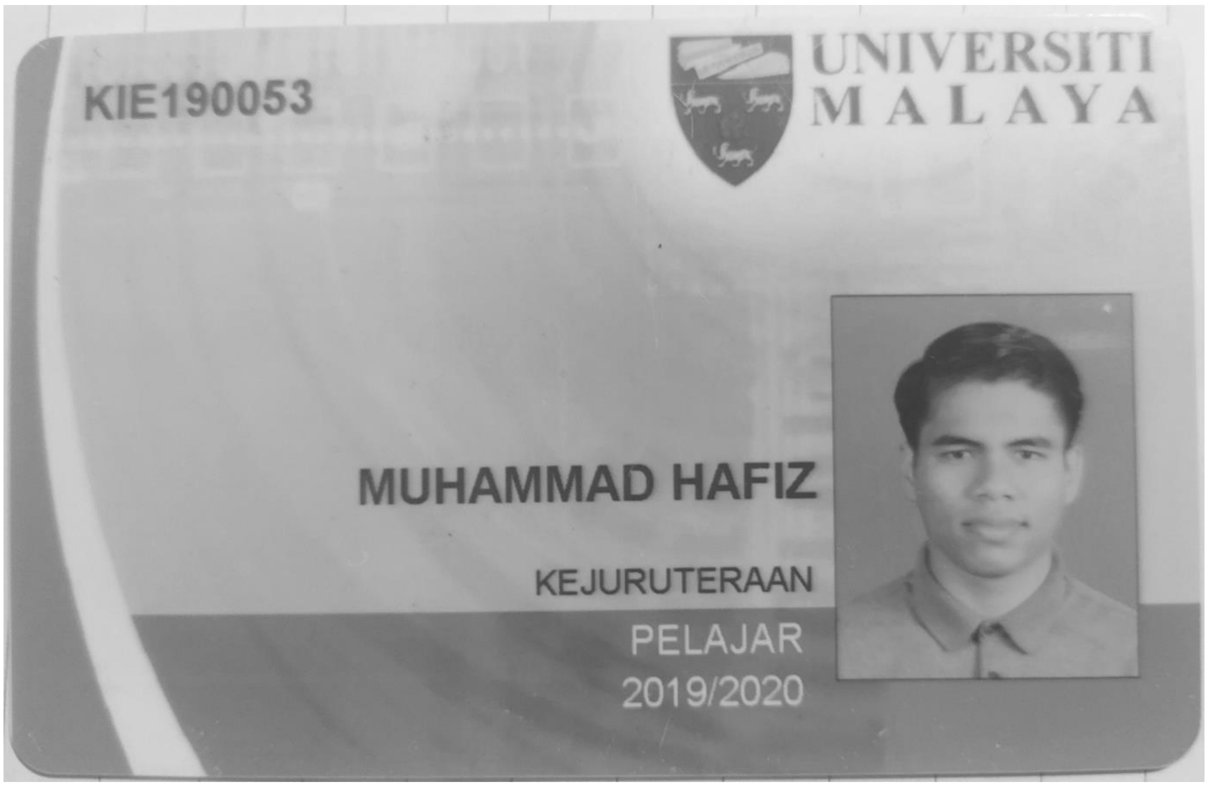
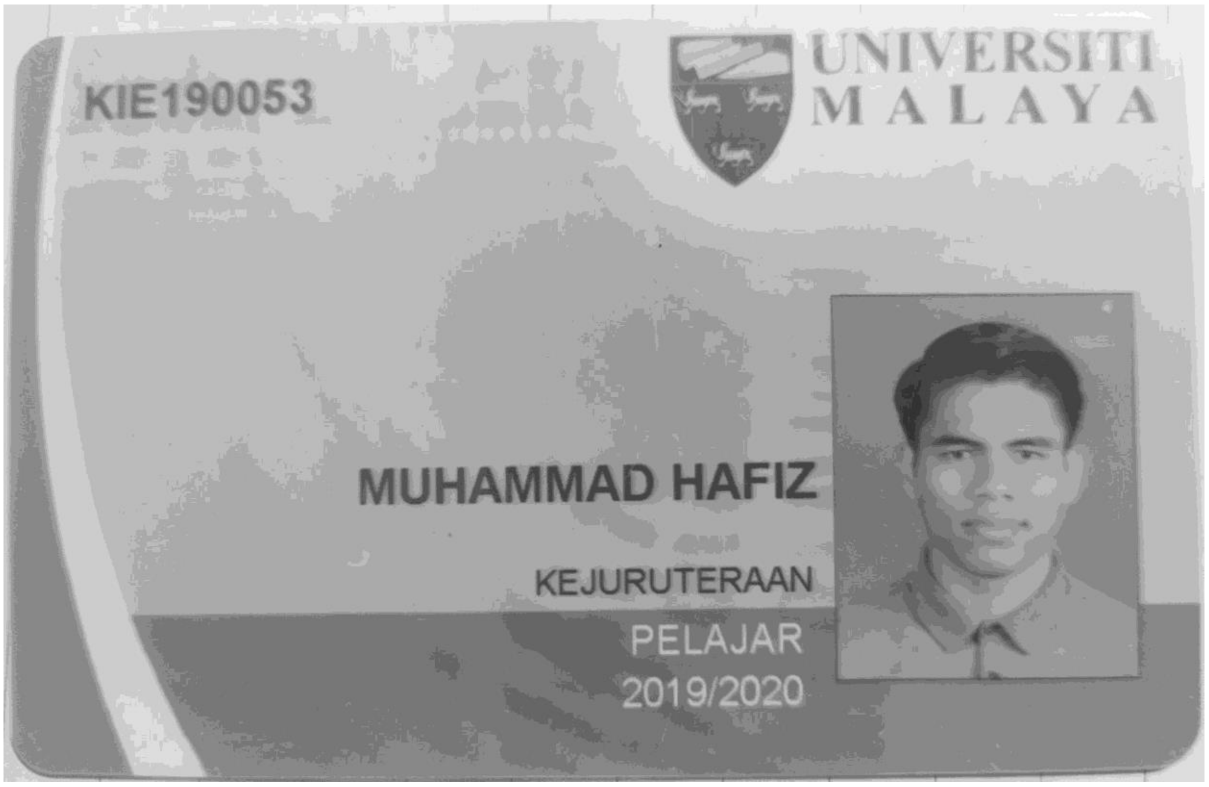


Figure 2a.1 shows original size 3479x2248 and size is 304kB

In Figure 2a.1, the image is clear and sharp and the size of the image is 304kB. All the information can be read easily.



KIE190053

UNIVERSITI MALAYA

MUHAMMAD HAFIZ

KEJURUTERAAN

PELAJAR

2019/2020

Figure 2a.2 shows 1740x1124 image and size is 81.1kB

In Figure 2a.2, the image is downsampling by 2, we can see the image slightly blur and the size of the image is reduced to 81.1kB

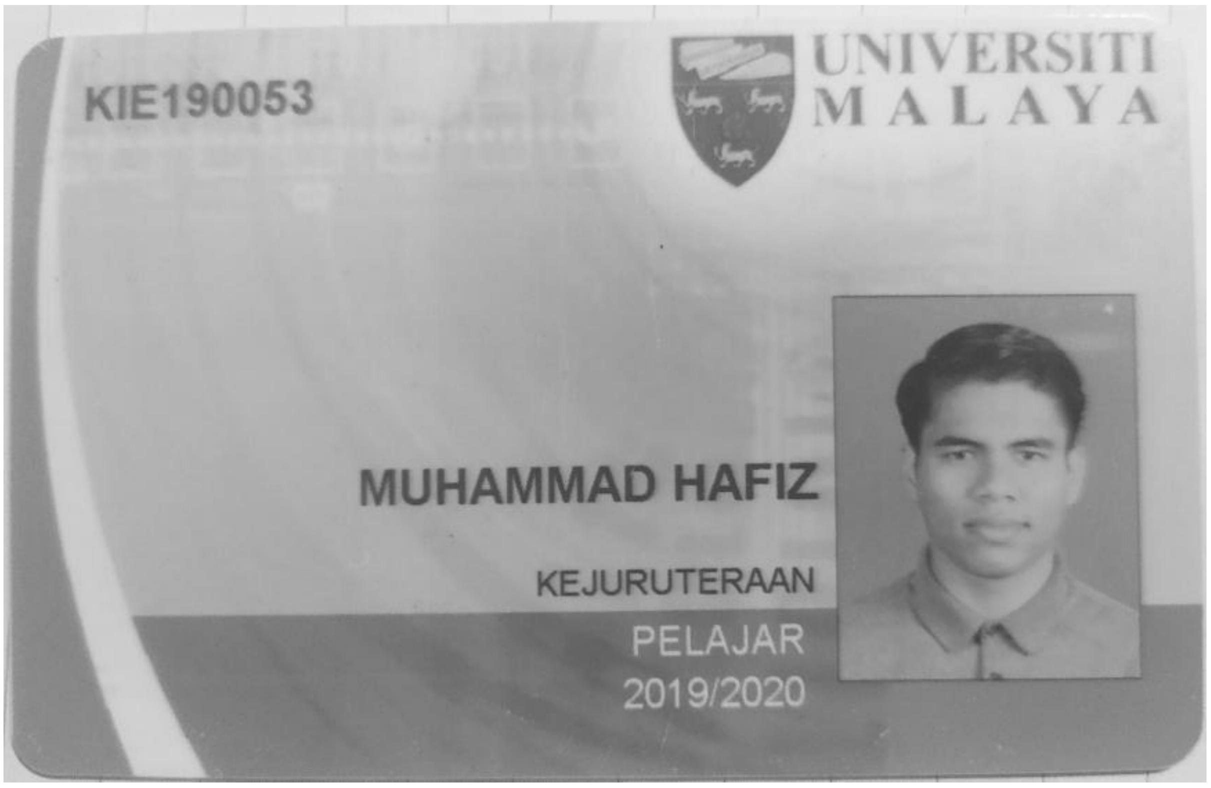


Figure 2a.3 shows 870x562 image and size is 28.7kB

In Figure 2a.3, the image is downsampling by 4, the image is slightly blur and more blur than Figure2a.2. The size of the image is further decrease to 28.7kB.

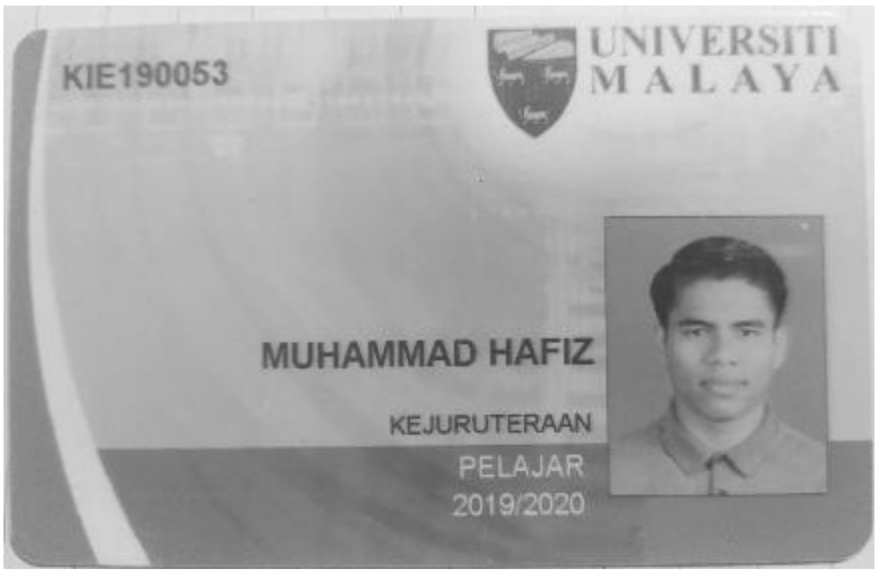


Figure 2a.4 shows 435x281 image and size is 11.1kB

In Figure 2a.4, the image is downsampling by 8, the image is moderately blur and the image sizes becomes smaller. The size of the image is further decrease to 11.1kB.

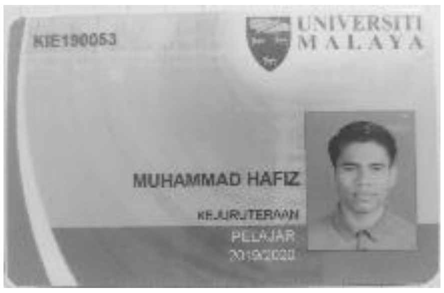


Figure 2a.5 shows 218x141 image and size is 4.42kB

In Figure 2a.5, the image is downsampling by 16, the image is very blur and the image sizes further decreased. The size of the image is further decrease to 4.42kB.

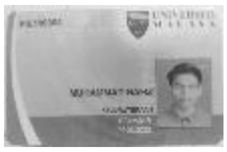


Figure 2a.6 shows 109x71 image and size is 1.67kB

In Figure 2a.6, the image is downsampling by 32, the image is extremely blur and the image sizes further decreased. The size of the image is further decrease to 1.67kB. The important information from the image cannot be analysed easily. For example, the word of ‘PELAJAR’ is unreadable.

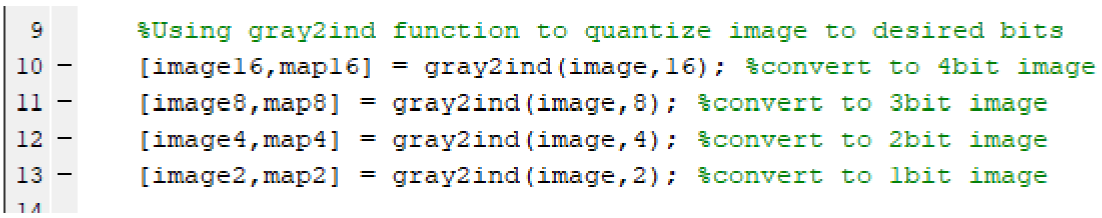
Question 2b: Design of solution

For digital image processing, the quantization level determines the number of grey levels in the digitized image. The magnitude of the sampled image is expressed as a digital value in image processing. For this grayscale image, each pixel is a single sample representing only an amount of light; that is, it carries only intensity information the image is construct in 8-bit.

Code below is used to get the bit depth of the image

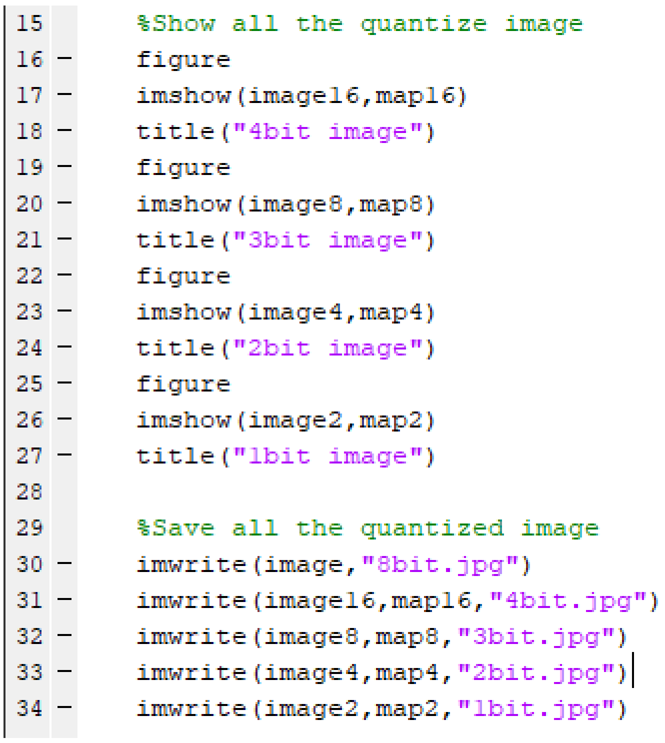
```
infol=imfinfo("hafiz_gray.jpg")
```

After that, we simply used gray2ind function to convert grayscale image to indexed image with 16 indices, 8 indices, 4 indices, and 2 indices, which is equals to 4-bit, 3-bit, 2-bit and 1-bit respectively as shown in code below.



```
%Using gray2ind function to quantize image to desired bits
[image16,map16] = gray2ind(image,16); %convert to 4bit image
[image8,map8] = gray2ind(image,8); %convert to 3bit image
[image4,map4] = gray2ind(image,4); %convert to 2bit image
[image2,map2] = gray2ind(image,2); %convert to 1bit image
```

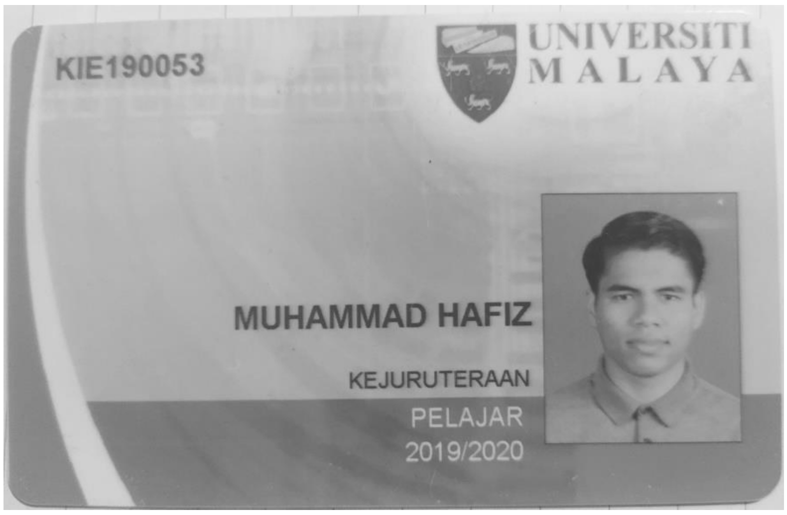
After that, the images are showed and save in jpg format.



```
%Show all the quantize image
figure
imshow(image16,map16)
title("4bit image")
figure
imshow(image8,map8)
title("3bit image")
figure
imshow(image4,map4)
title("2bit image")
figure
imshow(image2,map2)
title("1bit image")

%Save all the quantized image
imwrite(image,"8bit.jpg")
imwrite(image16,map16,"4bit.jpg")
imwrite(image8,map8,"3bit.jpg")
imwrite(image4,map4,"2bit.jpg")
imwrite(image2,map2,"1bit.jpg")
```

Question 2b: Results and discussion


KIE190053

UNIVERSITI MALAYA

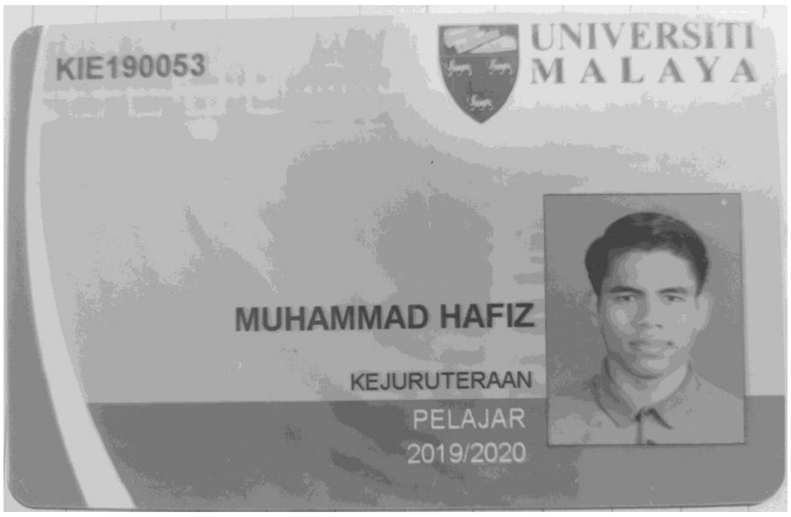
MUHAMMAD HAFIZ

KEJURUTERAAN

PELAJAR

2019/2020

This is the original grayscale image with 8-bit. The intensity of pixel is good and can see the image clearly.


KIE190053

UNIVERSITI MALAYA

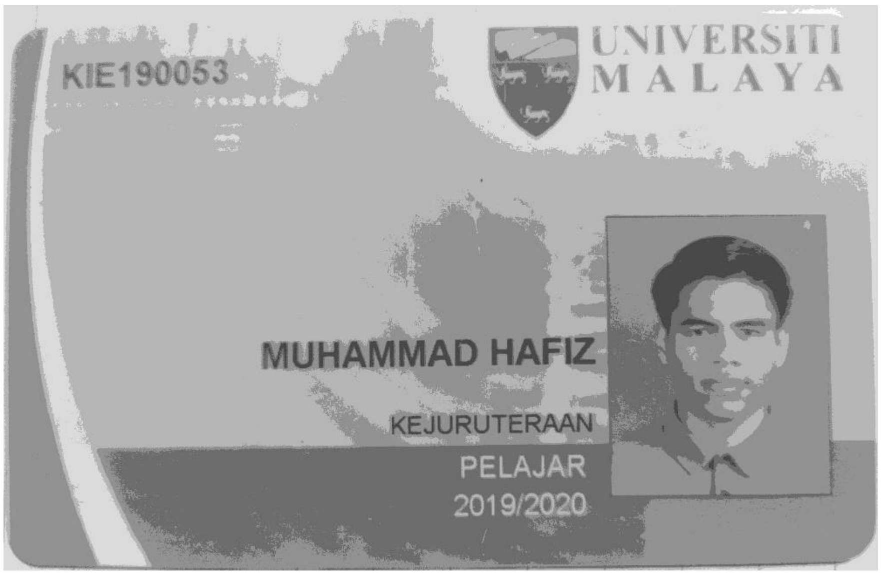
MUHAMMAD HAFIZ

KEJURUTERAAN

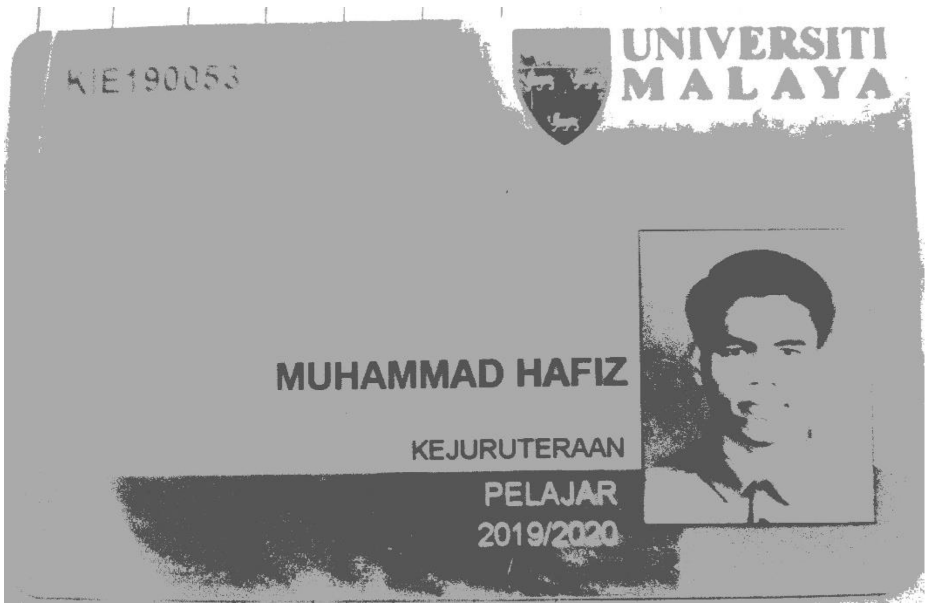
PELAJAR

2019/2020

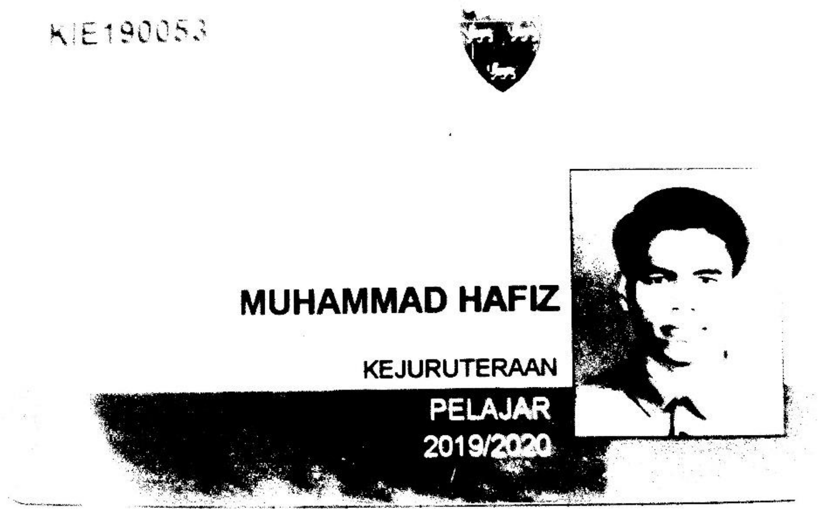
This is 4-bit grayscale image. The intensity of pixel is reduced and the grey colour is reduced. The image can see clearly.



This is 3-bit grayscale image. The intensity of pixel is reduced and the grey colour also further reduced. The image looked like ‘cartoon’.



This is 2-bit grayscale image. The intensity of pixel is reduced and the grey colour also further reduced.



KIE190053

MUHAMMAD HAFIZ

KEJURUTERAAN

PELAJAR

2019/2020

This is 1-bit grayscale image. The intensity of pixel is reduced and the grey colour are not allowed. The color has white and black only because 1-bit. The image looked like old image in 18th century.