THE UNIVERSITY OF NOTTINGHAM

SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING

H64DS2 project

Academic year 2007/08

Industrial voice control

Full report

STUDENT ID 4011551

DATE submitted 6th May 2008

Project is submitted in part fulfilment of the requirements of the H64DS2 module.

TABLE OF CONTENTS

Introduction 3

Step 1 - Data Acquisition 3

Step 2 - Signal Interface to MATLAB 4

Step 3 – Visual Analysis of the signals 6

Step 4 – Correlation Detection 12

Step 5 – Spectral Analysis 13

Step 6 – Signal enhancement by bandpass filtering 16

Step 7 – Weiner filtering 18

Step 8 – Adaptive filtering 19

Step 9 – Choice of the sufficient sampling rate 21

Step 10 – Elaboration of the system outline. 22

Step 11 - Research and reflection.. 24

Bibliography and references 27

H64DS2 – DSP for Telecommunications, multimedia and instrumentation with project

(20 Credits)

Industrial Voice Control project – FULL REPORT

Introduction

The aim of this project is to conduct a feasibility study of implementing voice controlled functionality to machinery within an industrial environment. This will involve several steps included collecting the data, simulation and estimating the development and implementation costs. This report focuses on the three steps that have already been carried out which have dealt with the acquisition and analysis of typical signals that may be encountered. In a typical system the operation of the machinery would be controlled by the user stating different control words e.g. stop or go etc. In an industrial environment this may be a difficult task as a variety of background noises and electromagnetic interference are present. Therefore the initial steps of development required simulation of these signals begin captured with these typical noises present.

Step 1 - Data Acquisition

The recorded control words (stop, check and go) were recorded via a standard omni directional microphone interfaced with a standard PC soundcard. Care was taken to avoid saturation. The sounds were recorded at a sampling frequency of 22050 Hz at 16 bits on a single channel (mono) without compression. The sounds files conformed to the standard wave format. Care was also taken to keep the recordings under 3.00 seconds and the control words were repeated four times consecutively within the 3.00 seconds.

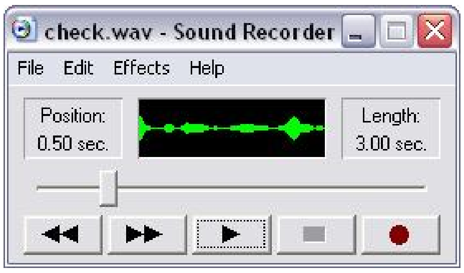


Figure 1 - Sound Recorder Waveform

The 'noise.wav' was obtained from a standard PC fan (As the fan rotates at a speed of 2600 rpm it was noted that a possible frequency of the noise signal might be a strong frequency response in the 43.33 Hz range (2600/60 = 43.33 revs per seconds of Hz). This is analogous with filtering for a 50Hz main signal from sensitive equipment powered off of mains electricity.

The tone signal used was tone7 as dictated by my student ID: 4+0+1+1+5+5+1 = 17 -> tone7.wav

The 'stopsat.wav' was recorded by deliberately causing saturation which could be seen via the display window. On playback the quality was very poor as the physical signal's amplitude exceeded the recording range of the PC's soundcard e.g. the maximum recorded signal level was outside the range and was consequently clipped. Thus in comparison to the 'stop.wav' file, 'stopsat.wav' was a partial/distorted representation of the original analogue signal.

Step 2 - Signal Interface to MATLAB

MATLAB provides the facility of being able to load the samples from a recorded wave file into an array which can then be manipulated via different operations e.g. multiplication, transposition etc. This was achieved by using the ‘wavread.m‘ function which takes a wave file and loads each of the samples into a target array. The syntax needed to do this was ‘Y = wavread(‘name_of_file.wav’. Additional options are that you can return the sample rate and the number of bits per sample to two single variables designated typically by ‘Fs’ and ‘NBits’ by specifying additional parameters: [Y,Fs,NBits] = wavread(‘name_of_file.wav’). This was done for all the files and the results were as follows:

File name	Resultant Array size	Sampling frequency (Hz)	No of Bits per sample
stop.wav	66160	22050	16
check.wav	66159	22050	16
go.wav	66150	22050	16
noise.wav	66155	22050	16
stopsat.wav	66162	22050	16
tone7.wav	66150	22050	16

Table 1

These values matched the sampling rates and number of bits specified during the recording of these files. It was noted that if a recording was to last for exactly three seconds and it was sampled at 22050 Hz then the theoretical size of the array that Matlab would produce would be:

$$Array\_size = Total\_signal\_length * Sampling\_frequency$$

Equation 1

E.g. sampling period = 1/22050 Hz, Array size = 3s*22050 = 66150 elements for the array.

The actual values varied slightly from file to file which was due to very small variations in the track length as some of them were slightly under and over 3 seconds (error from trimming of the wave file performed by sound recorder).

The ‘sound.m’ function was used to play an array through Matlab. The syntax needed to do this was “sound(Y,FS,BITS)” where ‘Y‘ is the target array to be played and ‘FS’ and ‘BITS’ are the sampling frequency and number of bits that it will be played back with. Several files were loaded into an array via ‘wavread.m’ and played back at their sampled frequencies. This obviously made no difference but after changing the sampling rate e.g. from 22050Hz to 44100 Hz the signal played back at twice the speed. E.g. when playing back tone7.wav at a higher sampling frequency it produced a tone of a higher pitch or frequency.

This is because the originally recording was sampled at a fixed frequency for the duration of the signal length. Playing back the samples at twice the originally recorded rates will obviously playback the original signal at twice the speed. Decreasing the sampling frequency obviously has the effect of playing back the signal at a slower rate. Halving the playback sampling frequency means the playback will take twice as long e.g. 6 seconds.

Altering the number of bits to play back the array from 16 to 8 did not seem to make much difference to audible quality. Theoretically a reduced number of bits will reduce the quality of the recording as more quantisation noise is introduced and less detail of the original signal’s waveform is represented [1]. This was more evident when the number of bits was reduced to 4 and 3 bits. The playback was significantly distorted and sounded as if it has been clipped. The effect of playing back with a smaller number of bits decreases the amount of discrete levels that can be represented by the signal (quantisation issue). This resulted in a significant degradation of the audible quality of the signal. Obviously the signal was recorded with 16 bits per sample and the maximum range of bits you can play back a file with is 2 to 16 bits with ‘sound.m’. If you could playback with more bits then theoretically there would be no change as there would be no additional information to reproduce [2].

The ‘soundsc.m’ function allows you to play back an array with the added ability of scaling the outputted signal. This can be done by including additional parameters along with the function e.g. soundsc(Y,…,SLIM) where SLIM are you limits for the scaling. This can be useful in playing back arrays that contain data outside of the limits of 1 to -1 by scaling the vector and playing it back through the sound card in the allowable range e.g. 1 to -1. For example you can playback an array/vector (peak to peak 2 to -2) with a scaling [10 -10] that produces a playback significantly lower in volume and is scaled to be within range. Using this function to playback ‘stopsat.wav’ produced no overall improvement in the sound quality. This was because the clipping was not a result of playing back the wave file and it being out of range. The problem was that the originally recorded of the analogue waveform (the sound -> microphone) was clipped at this stage. Therefore the waveform was a distorted representation of the original analogue waveform. Simply scaling the playback vector does not introduce the original analogue information that was omitted in the first place. Therefore the audible quality could not be improved with ‘soundsc.m’.

The next task was to generate white noise via the ‘randn.m‘ function. The ‘randn.m’ function allows you to produce an array of random variables following the normal distribution with a mean of zero and a variance and standard deviation of one. A typical normal distribution with a mean of zero means that the distribution will be centred on the value of zero on the x axis. A standard deviation of one dictates that approximation 68% of the values should fall within one deviation away from the mean [3].

The numbers were pseudo randomly generated and therefore their generation was based on an algorithm which could already be at a particular state e.g. having calculated up to a certain value. The command “RANDN('state',0)” was used to set the random number generating algorithm back to its initial state.

Using the syntax “RANDN(M,N)” produced an array of random numbers. As the task was to produce a white noise sound file a single line array was needed to last for three seconds. The array would therefore need to consist of 1x66150 elements (see equation 1). The random noise array was produced via the following “whinoise = RANDN(66150,1)”.

The next task involved writing the array to a file called ‘whinoise.wav’ using the ‘wavwrite.m’ function.
The syntax necessary to write is an array to a wave file is “wavwrite(Y,Fs,NBits,'name_of_file.wav')”.
Initially an attempt to write the noise array via the syntax “wavwrite(whinoise,22050,16,'whinoise.wav')” caused MATLAB to generate a warning stating that the data was clipped when attempting to write to the file. This was because an element in the array cannot exceed 1 and -1 as the wave file can only have a peak to peak value of 1 to -1. Using the command “Max(abs(whinoise))” returned a value of 4.211 to 3 d.p. Therefore the array values needed to be normalised to make the highest value less than or equal to one and the other should be reduced in amplitude proportionately. This was achieved by using the syntax “whinoise = whinoise/max(abs(whinoise)*1.0001)”. The max value of the array was now 0.999 (within range). The file was successfully written out as ‘whinoise.wav’. On playback the sound resembles noise that you may hear from an radio or TV set which has not be tuned into a channel as this is essentially white noise.

On comparison there was a difference between the ‘noise.wav’ and the ‘whinoise.wav’ file. As mentioned before the noise file was recorded from a pc fan outlet and is caused by a mixture of the passing air and the frequency of oscillation of the fan. These two components obviously have strong frequency components in the audible range as they can be heard. This explains why the ‘noise.wav’ contained a humming sound that was periodic in nature (fan speed + air). The whinoise.wav file produced a hissing sound on playback. Ideal white noise is equivalent to white light and contains all frequencies. The ideal power spectral density of white noise should be uniformly flat as there should be no significant gain at any particular frequency. Thus when playing back ‘whinoise.wav’ many of frequencies were present (obviously not an infinite amount) leading to a ‘hiss’ effect [1].

The next task was to create a mixed noise file that would be a mixture of the recorded noise (noise.wav), the tone and the generated white noise (whinoise.wav).

In order to multiply or mix an array they need to be of the same size e.g. all of the arrays needed to be the same length i.e. the length of the smallest array involved in the mix. A routine was created in the script file (step2.m) that would ensure that this was the case before any attempt at mixing was carried out. This was achieved by making sure all the necessary signals were read into arrays in Matlab.
Then the function ‘length.m’ was used to determine the length of each of the arrays e.g. length (noise). The values for the lengths were then stored into an array called ‘L’. The function ‘min’ was then used to determine

the smallest array. This value was then used by resizing all of the arrays to be mixed by setting the array equal to a smaller version of itself e.g.

```
L(1) = length(noise);
L(2) = length(tone);
L(3) = length(whinoise);

L = min(L);   % determine the array with the smallest length
noise = noise(1:L);
tone = tone(1:L);
whinoise = whinoise(1:L);
```

Note: A few elements from larger arrays were dropped but on average this was a very small amount (6 elements) and didn't remove a significant part of the signal.

The mix was created by following a formula with values unique for my student ID. The following formula was used:

SI: 4011551   *mixnoise=noise*6+tone*6+whinoise*2*

This array was then normalised and saved as a wave file 'mixnoise.wav'.

The Matlab script (step2.m) carries out of the above steps automatically and generates the required output. The various stages that the script is executing are indicated by messages displayed at the command window. The file 'mixnoise.wav' is then played back via 'sound.m' with a message stating "pause(duration_of_playback)". This lasts for the duration of playback by using the pause function with the parameter 'L/22050' (66150/22050 Hz = 3 seconds) which produces a pause for the duration of the 'mixnoise.wav'.

Step 3 – Visual Analysis of the signals

The 'plot.m' function in Matlab allows graphical plotting of a vector or vectors. As the signals have been loaded into arrays within Matlab, a plot of these arrays produces a plot which is essentially a time domain representation of the signal. This is verifiable by comparing the graphs to that of the sound recorder's display during playback where the features match at certain time intervals.

The x axis represents time and is generated for the plot by using the 'plot.m' function to plot two arrays e.g. x and y against each other. An array called time was generated via "time = 0:1/22050/3", which produced an array of values that could then be used as the x axis values to plot against the signal. This provided a time scale to accompany the plots and gives an idea of how the signal varies over time.
The plots were generated via "plot(time,signal)" along with an additional command "xlabel('Time (s)')" which simple provides a label for the x axis. The resultant plots of all the signals can be found in the presentation file (pages 2-7).

The interval chosen to represent the 'stop' control word was taken form the 'stop.wav' signal from sample 1 to 12000. This provided adequate samples to represent the entire word as the playback of the array 'repstop' resulted in a just one instance of the 'stop' control word. This was then saved as 'repstop.wav'. The corresponding intervals for the other signals were also determined and their plots were obtained.

The 'mixnoise' signal contained a mixture of the tone, the recorded noise and the white noise and there were characteristics of each signal present in the mix. The tone signal had a very distinctive waveform with strong rectangular intervals clearly present in the mixnoise signal. Looking initially at the whole waveforms it was difficult to see the individual contributions from the noise and white noise. Looking more closely at the representative intervals it was seen that the individual signals had affected each other and this led to a mixture of the two (indicated by point A).

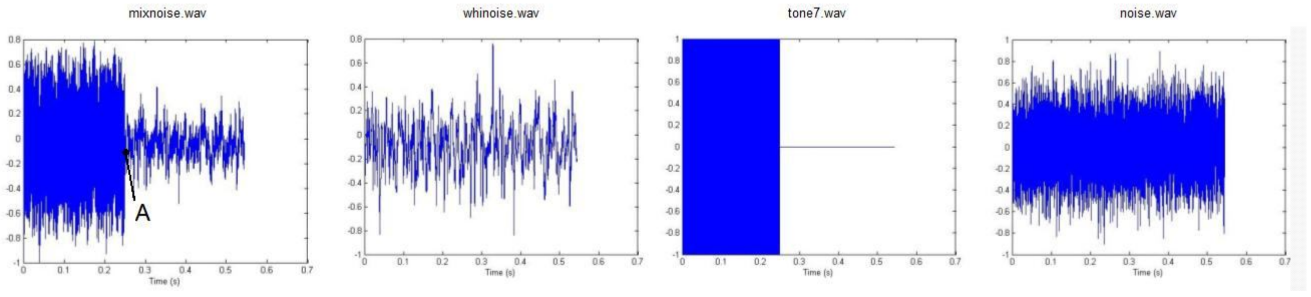


Figure 2 – mixnoise.wav and its constituent signals

At point 'A' the tone signal was almost equal to zero, so the contribution of it to the mixnoise signal at this point was very small. It could be seen that the larger amplitudes of the white noise signal were no longer present in the mixnoise. The signals standard deviation at point A had also changed and reflected the combination of the two noises.

The next task was to create three mixes of the 'stop' control signal and the mixed noise signal. This provided a good approximation to the type of signal that may be recorded by a voice controlled system in an industrial environment e.g. tones, machinery sound, various frequencies and of course the control signal. The three mixes required different weightings of the noise and the signal e.g. large amount of noise (in terms of magnitude) to a small amount of the signal. This was achieved by using the provided formulas:

spn 1=mixnoise/max(abs(mixnoise))(4)+stop/max(abs(stop))*
spn 2=mixnoise/max(abs(mixnoise))+stop/max(abs(stop))
spn 3=mixnoise/max(abs(mixnoise))/(4)+stop/max(abs(stop))

These mixtures were then also normalised.

On viewing the plots of the signals of the spn mixes the characteristics of the original signals were present to various degrees. The different mixes spn1, spn2 and spn3 involved different amounts of signal being mixed with noise (weighting dictated by the formula). The plot of 'spn1' indicated that the small amount of the stop signal added had little effect on the consequently produced waveform. The plot was not very different from the original mixnoise.
The second plot 'spn2' indicated the 'stop' signal had more of an effect as the time domain waveform of the 'stop' word could be seen to shape the signal for its duration. The combination of the 'mixnoise' and 'stop' signals can be clearly seen in the graph below (Figure 3). The presence of the stop signal waveform can clearly be seen as the waveform has changed from the original ranctangualr shape (interval for mixnoise) to the more that of the time domain representation of the word 'stop'.

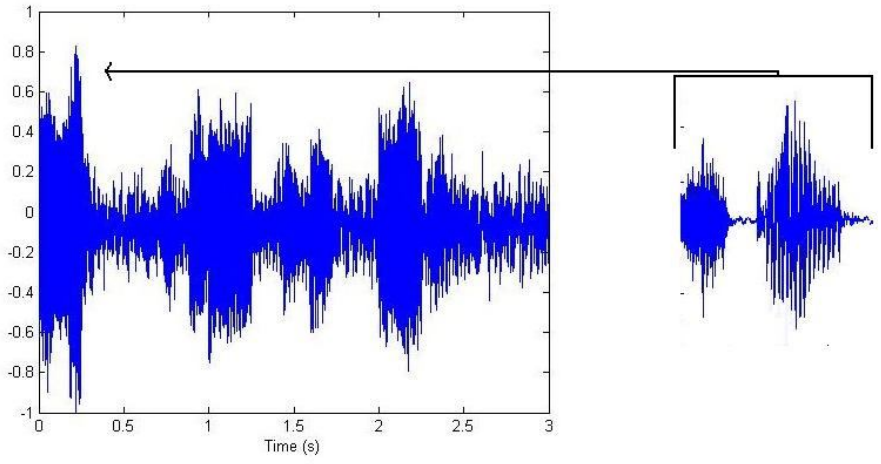


Figure 3 – Spn2.wav

Finally in ‘spn3’ a larger amount of signal was present with respect to the noise. The plot clearly showed the stop signal had a significant effect for its duration with that part of the signal very closely resembling the ‘stop’ waveform.

The plots mentioned above have all been included in the presentation file on page 4 labelled spn1, spn2 and spn3.

The probability density function of a set of data can be a useful analytical tool as it can provide you with information regarding the spread of the data and the likelihood of certain values occurring [3]. This also applies to signal analysis where characteristics from certain signals will be present in the mixes of different signals. A way of performing the analysis in Matlab is to use the ‘hist.m’ function which allows you to plot a histogram of the signal and shows the frequency (amount of occurrences) of the samples values for the entire signal. This can be converted to a probability density function by taking the relevant frequency for a particular value and dividing it by the total number of samples. This gives you a percentage value which represents the likelihood or probability of a value for a sample occurring.

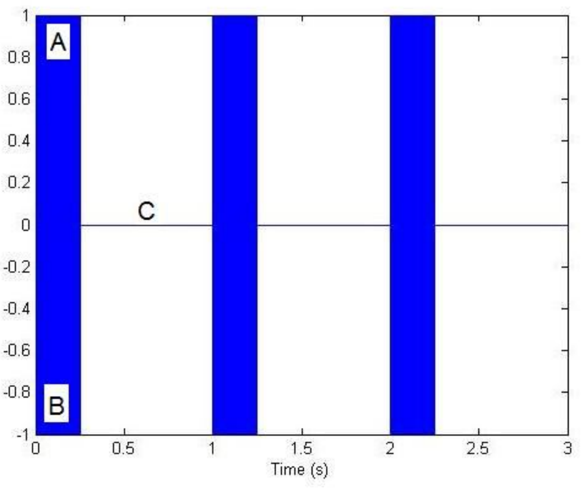
N/B: in this section the word frequency is used to indicate the number of times of occurrences of a particular value for a sample i.e. similar to that of a tally mark. This is not to be confused with physical or audible frequencies.

The analysis carried out in Matlab was with using a histogram with a 100 boxes which means that the data was grouped into 100 boxes. Choosing different numbers of boxes can indicate different features of the data set, but 100 seemed to provide a good representation of the probability density/distribution.

The ‘stop’ signal’s distribution indicated a mean signal value that was off centre from zero and that a majority of the signal’s samples were between 0.5 and -0.5. This was clearly evident when looking at the signal in the time domain as the signal on average was between 0.5 and -0.5.

The white noise’s histogram showed a spread of data which was very close to an ideal normal distribution. The curve was centred on zero (the mean of the distribution) which reflects the type of distribution that is meant to be generated by the ‘randn.m’ function. As mentioned before white noise ideally contains all frequencies and consequently would follow the normal distribution in terms of its probability density function. Therefore the obtained distribution seemed to be correct.

The tone’s histogram reflected the characteristics of the time domain signal as when the tone was present the signal was very close to 1 (point A in figure 4) and -1 (point B) and approximately zero (point C) until the next tone. This is reflected as the sample values with the highest frequencies reside around zero and the sample values with the next highest occurrences are for the tone being at approximately 1 and -1 (very small in comparison).



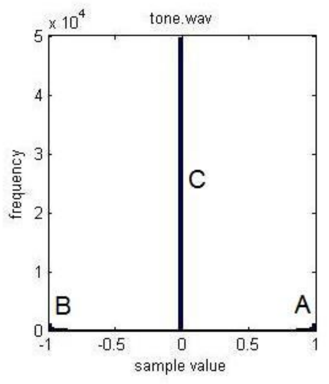


Figure 4 – Tone.wav and its histogram

The noise signal's histogram indicated a very similar distribution to that of the white noise (similar time domain appearance). The mean of the distribution was slightly off centre indicating that the distribution of the samples was not completely uniform. If the power spectral density of the signal was to be plotted then it would not be completely uniform as with white noise and may indicate power at specific frequencies.

The mixnoise's histogram reflected the mixture of the three signals as the characteristics of the distribution now contained elements of each individual signal's own distribution. The mean of the distribution had shifted away from zero (the contribution from noise.wav) and the maximum amplitudes of the some of the samples had increased from 2000 2225 to 3000 (contribution from tone.wav). The overall profile of the distribution had also changed as a result of the difference in standard deviation between 'noise.wav' and 'whinoise.wav' and the effect of many of the samples around zero due to 'tone.wav'.

'Spn2' was a mixture of the 'stop' and the 'mixnoise.wav' signal (noise+whinoise+tone) and therefore the characteristics of all of the signal's distributions made a contribution. The resultant histogram produced a distribution that was slightly narrower in width (standard deviation) and the maximum amplitude had now increased from 3000 – 3750 (due to frequencies of the sample in the 'stop' distribution). The narrow profile of the 'stop' signal caused the distribution of 'mixnoise' to narrow in width and caused an elongation of the distribution in the vertical direction. Thus the distribution of the 'spn2.wav' reflected the combination of the two signals.

The analysis was carried out for a second time for the representative intervals of the various signals. The representative intervals produced histograms that resembled the histograms for the full signals to a good to degree of accuracy. The histograms did not exactly match but they did reflect the same characteristics of the distributions of the full signals. For example the histograms below are for the full signal and representative interval of 'mixnoise':

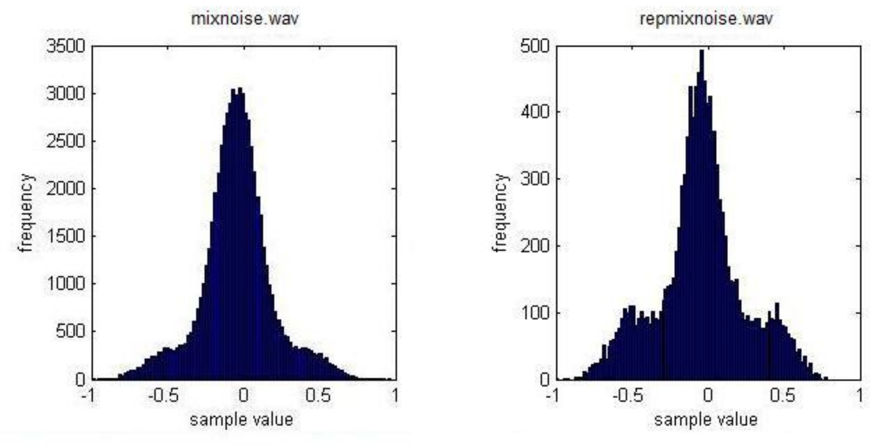


Figure 5 – Full and representative histograms of mixnoise.wav

It can be clearly seen that the representative interval has similar characteristics but is somewhat distorted in terms of its profile. When the histograms are taken for the whole signal it has the effect of averaging the result and represents more accurately the true nature of the distribution of the signal. This was the case for the entire representative intervals of the full signals analysed and the resultant histograms are included the presentation file.

A useful indication of the quality of a signal is the signal to noise ratio. The S/N ratio is a numerical ratio of the strength of the desired signal to the noise that is also present. Certain applications might require a certain value for the S/N of a signal in order to use it e.g. of a reasonable level to allow demodulation/decoding.
The signals SPN 1, 2 and 3 were mixes of a control signal 'stop' and a mixture of noise and a tone.

The mixes were created with slightly different weightings of the control signal to the noise which resulted in one mix featuring predominantly noise, one with equal weighting and one featuring mainly the control signal.

The S/N ratio was calculated for the signals via two different methods in Matlab. The first method involved taking the signal and calculating the abs (absolute value) version of it. This caused any negative values to become positive providing a signal in the range 0 to 1 instead of -1 to 1 (See figure 6). The maximum value of the signal was then taken and this was carried out for both the 'signal' and the 'noise' signal.

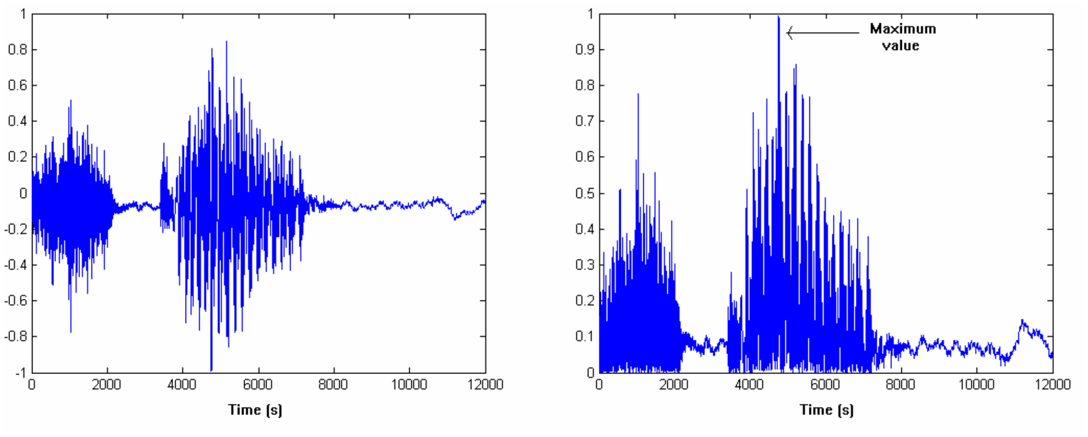


Figure 6 – repstop.wav and its ABS version

These two maximum values were used to calculate the S/N ratio with the following formula:

$$S/N = 20\log\left(\frac{Signal}{Noise}\right) = dB$$

Equation 2

This was possible as the recorded signals were essentially time varying voltages with the voltage being proportionate to power. Thus the ratio of two voltages could be calculated in decibels.

Another method for calculating the S/N ratio was made by using the standard deviations of the two signals. The standard deviation is essentially the average variance of the deviations from the mean.

The two methods mentioned obviously produced different result numerically but the overall difference between the signals were similar for both e.g. if one method produced a low S/N value for a mix then the other method indicated a similarly low S/N ratio.

The calculations for each of the mixes spn 1, 2 and 3 were calculated by using 'repstop' and the same interval from the 'mixnoise' signal (mixnoise(1:12000)).

The 'step3.m' file that was created performs the calculation for the S/N ratio via both methods automatically and presents the results at the command window. The results for the produced mixes were as follows:

ABS BASED CALCULATIONS	S/N IN DB TO 2 D.P
Spn1	-27.86
Spn2	-0.14
Spn3	27.59
Standard deviation based calculations	
Spn1	-40.81
Spn2	-13.08
Spn3	14.65

Table 2

The results were clearly different for each method but they reflect similarly overall characteristics between the mixes. The first (spn1) having a very poor S/N ratio where the actually control signal was barely audible, the second one (spn2) where there was little difference between the noise and the signal and the third (spn3) where the signal was stronger in comparison to the noise.

In practice different methods are used to calculate the S/N ratio and are all generally acceptable as performance measures. There is no formal definition for the S/N ratio and therefore no formal methods are required as long as the calculation is comparing a metric quantity of the two signals e.g. voltages. A more general requirement is that the two signals must be measured at the same or equivalent points and within the same system bandwidth [4]. The two methods used obviously meet the criteria and therefore were acceptable for producing valid S/N calculations.

Step 4 – Correlation Detection

This step was necessary as ideally the system should be able to take the current signal (noise+user voice) and compare it to known control words. This approach would allow occurrences of control words to be matched to the stored version and the appropriate action could then be taken by the machine e.g. if stop, stop machine, issue alarm signal etc. This required the creation of a matched filter and two methods were used. A key requirement is that the impulse response of the filter should correspond to the time domain representation of the representative signal. The intial impulse response of the filter was produced by taking the full range of the repstop signal (the 'stop' control word). The array was then inverted. Syntax used "ir = repstop(1:12000), ir = flipud(ir)" and its zero DC content was removed.

Initially an approach was taken to try and convolve the signal with another signal that contained the signal, plus a mixture of noise. The convolution carried out can be described via equation 3 and produced a result that was the combined length of the reference signal + the signal noise mix.

$$[f * g] = \int_0^t f(\tau)g(t-\tau)d\tau \qquad \textbf{E.q. 3}$$

The convolution was achieved in Matlab by using the conv.m function, which requires two input vectors A and B. The following syntax was used "answer = conv(ir,spn3)". This produced an array of length 78149 x1 e.g. (12000 + 66150 = 71849 one sample was omitted). A plot of the array can be seen in graph and shows that the convolution was performed correctly and occurrences of the stop control word can be clearly indicated by the spikes of large amplitude with a fairly symmetrical shape around the spike.

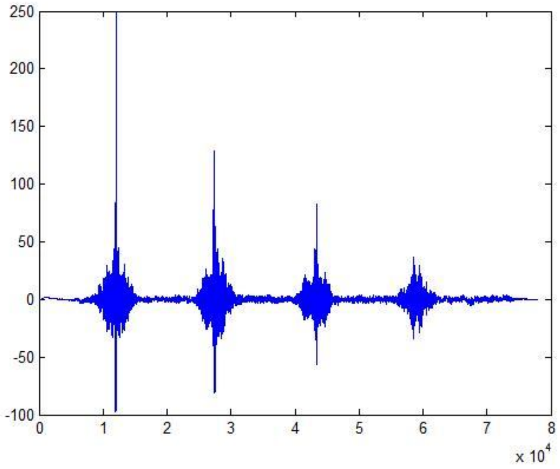


Figure 7 – Matched filter convoution

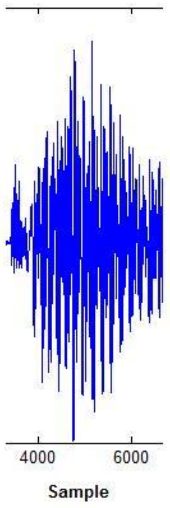


Figure 8 – 1000 samples for 'stop'

The matched filter was also achieved by using the filter.m function which allows you to specify two vectors to represent the filter A and B and apply the filter to a target array X. The two vectors that can be specified allow the creation of filters with or without feedback e.g. iir and fir. The filter necessary did not require any feedback thus the vector 'B' can be stated as 1. The syntax used was: "target_array = filter(ir,1,spn3)". This produced an array of the same size e.g. 66150. The resultant plot was nearly identical in terms of the height of the spikes and the symmetry of around the spikes. Therefore the convolution and the filter.m approaches produces similar results but with different sized arrays.

Next an attempt was made to reduce the size of the impulse response of the matched filter. It was found that a smaller amount of samples can still achieve reasonable results. Fewer samples will also be computationally more efficient and will require less hardware MAC operations. Experimentation was carried out with using different portions of the repstop array (1000 elements in length) to produce outputs. This was to see which portion of 1000 samples would produce the most useful output e.g. high spikes indication the stop control word. A satisfactory result was found by using a portion of the signal that occurred between the 4500th to the 5500th sample as this captures the most significant part of the time domain representation of the stop control word (See figure 8). This produced a filter response that had spikes of smaller amplitude but were yet sufficient to be able to identify a match and stop control word. Therefore the impulse response was successful reduced to no more than a 1000 samples in length, after careful selection in order to get the best filter response. Ideally the more samples for the filter the better but this is at a price of the amount of computer power e.g. speed (operations per second) and memory that would be needed to perform the filtering operation. A plot of the final 1000 samples used can be seen in the presentation file. Another plot was obtained by using freqz.m on 'ir' to obtain the impulse response of the filter including the magnitude and phase responses.

Now that matched filtering could be performed, a threshold limits was set, so that if the amplitude of the filters response exceeded it, then the stop control word had been detected. By looking at the filter response for spn3, the smallest spike that indicated a match had an amplitude of approximately 34. Therefore an initial threshold was set at 33. This was sufficient as it led to the correct detection of the stop signal at the correct time as this was cross checked with the time domain representation of spn3. The filter was performed with the other mixes of the noises and then it became harder to detect the occurrence of the stop control word. The original stop signal contained four repetitions of the stop control word with the last instance being significantly weaker. This can be observed on playback and seen in the time domain waveform. This meant that the last instance of stop would always be harder to detect and consequently produced very small spikes in the filter outputs. As the signal to noise ratio became smaller e.g. spn2 to spn1, less instances of stop were recorded as the threshold value was simple not being trigger (the spikes were not large enough). Therefore different threshold values were experiment with e.g. 23 which produced four counts of the word stop for the spn2 signal as the minimum spikes produced by a match was now less than the threshold. The amplitude was lower further for spn1, but and not all occurrences were successfully detected. The presentation file shows the digital output of the matched filters with applied the signal spn1, spn2 and spn3.

The digital signals that were obtained are also present and were achieved by checking each element of the array (the output) and comparing it to a threshold value. When the threshold value was triggered the array (z) current value would be set to 1 and if it was less than would be set to 0. An inherent problem with this approach is that the digital signal captured fluctuates between one and zero a number of times as does the matched filter response. This can be seen in figure 9.

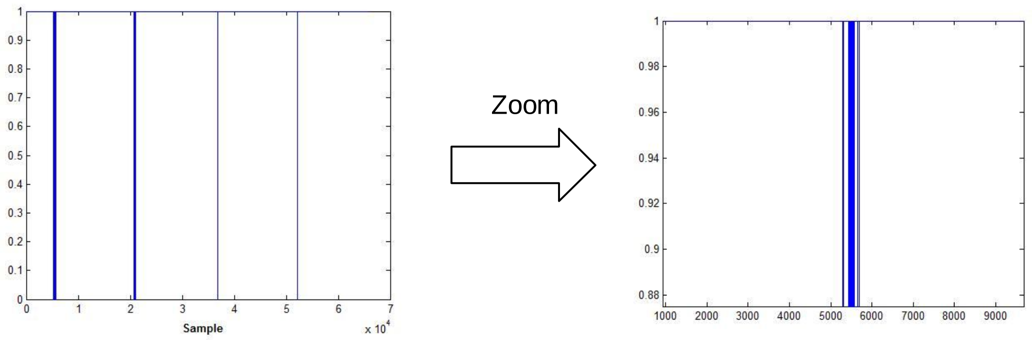


Figure 9 – Digital output signal.

In real life this could be compensated for by feeding the digital signal (z) to a circuit which performs a smoothing or envelope capture operation and then fed to a comparator. The output of the comparator could then be used to trigger an alarm signal etc.

As mentioned If the signal to noise ratio is lower such as in spn2, then the threshold needs to be revaluated as the amplitude spikes are not as large on average. A practical system may have to have a dynamic threshold level can be calibrated for various levels of background noise e.g. take a reading of the ambient noise and with a reference stop signal (e.g. operator states "stop" at an appropriate level), to determine the s/n ratio and then determine and set the threshold.

* Please note the script file for this step does take a long time to execute, but it does operate correctly

Step 5 – Spectral Analysis

Non parametric spectral analysis

Spectral analysis was performed on the signals using three subroutines fft.m, specgram.m and psd.m. The fft.m performs a fast Fourier transform on an input signal X which can be used to determine the amplitude and phase spectrum of the signal. This is a useful tool as it allows you to find out the strength of frequencies or range of frequencies for use in applications e.g. if some noise exists at a certain frequency you can filter it out if you know its location in the frequency spectrum e.g. mains hum at 50 Hz. It is used by specifying an input signal X and the number of points to use for the transform e.g. "fft('target_array',256)". As you increase the number of points, the fft becomes more comprehensive and indicates smaller amounts of detail regarding the frequency spectrum. Experimenting with different values was carrier out starting from very small values e.g. 256 samples up to 65536 samples. There is also an advantage in using a value for N which is a power of 2 e.g. 2^{10}, so only powers of 2 were used. The results obtained for using 1024 points on the signal 'tone' can be seen below.

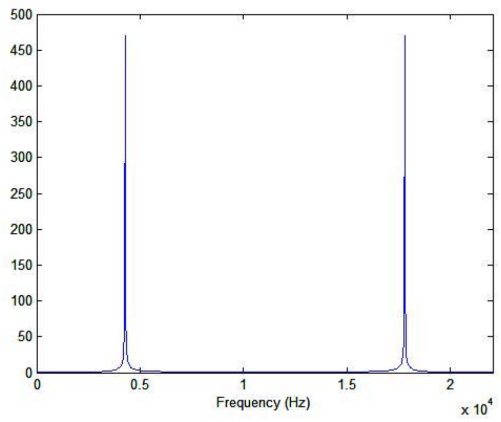


Figure 10 - Complex Spectrum

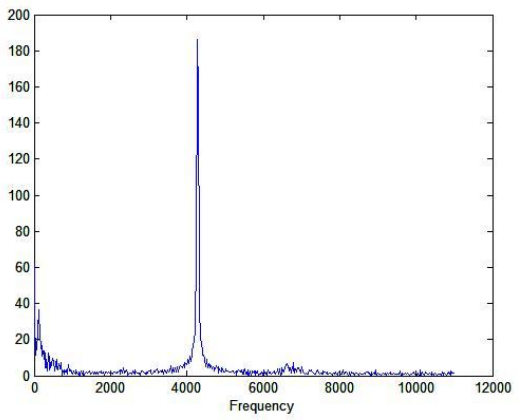


Figure 11 - Real Spectrum

Initially the function generates a result that has the full spectrum which contains an mirror image of the spectrum due the nature of the Fourier transforms e.g. negatives frequencies included. The real positive frequencies are the only ones of interest so half of the spectrum for the results was eliminated as shown in figure 11.

The corresponding x axis (frequency) was generated by the plots by use of the code f = 0:(11025/511):11020. This is because the frequency range was from 0 to the sampling frequency and the resolution can be seen to be:

$$\Delta f = \frac{f_S}{N} \text{ E.g. } 11025/512.$$

The addition of this axis was a useful aid in as it makes it possible to determine what is happening at specific frequencies.

The spectrogram routine preforms spectral analysis of the signal by applying the fft algorithm over a small section of the signal. It is a useful routine as it gives a time changing representation of the frequency spectrum. The plots produced, as outputs indicate graphically the frequency spectrum for a signal and how it varies over time. The colours used in the plot indicates the relative strength of the frequency e.g. red = high, dark blue = low. Spectrogram is used via by specifying the input vector X, The number of samples for the fft (nffft), frequency value used to scale the graph (Fs), a window e.g. hanning, Kaiser, etc and a value for overlap). Initially the following syntax was used: "specgram('target array',256,22050)". This produces valid results but only indicated vaguely where frequencies responses were strong and weak as indicated by the distribution of the regions of colour. In some of the plots the dominant frequency of the tone 4275 Hz can be seen via the sharp red section which gradually fades out into the rest of the spectrum. Plots with a higher resolution for tone.wav show more clearly the exact nature of the frequency spectrum as the red line is present, but less spreading occurs suggesting that most of the power is concentrated very narrowly around that frequency and little elsewhere.

The psd.m provides a more useful amplitude response as it indicates the power present at certain frequencies in decibels. This is useful for practical filter design as it provides you with the amount of attenuation you may need to achieve for a specific frequency if you wish to suppress it etc. Initially it was used without specifying additional parameters in order to generate basic amplitude responses. The number of points for the fft and the frequency axis were then changed e.g. "psd('target_array',1024,22050). The was then used to generate the results for the presentation file.

Parametric spectral analysis

Parametric spectral analysis can be useful when trying to determine the spectrum of a signal when the length of the signal is short e.g. a short duration. The results between the full and representative signals were very similar in terms of the shape of the responses, but the relative amplitudes were different. Although applying the non parametric method to a full signal might give a better indication of the true power at specific frequencies, pburg.m can give a general idea of the dominate frequency components for a very short version of the signal. This might be useful for some process which needs to know what frequencies to eliminate or enhance, without having to capture a lengthy signal.

In comparison the non parametric seemed to produce more accurate results than the parametric based analysis. This is because the parametric analysis makes is an approximation at certain points and makes use of averaging e.g. AR model or Auto regressive moving average. The indication of relative levels of different frequency components for both types was very similar but as mention previous the relative power amplitude tended to be lower in terms of dB in the parametric base analysis.

As mentioned all of the above methods were used over for the full and representative

Rootmusic.m was also experimented with to determine the dominant components of the signals. This was used via the following syntax: [X,power] = rootmusic('tone',4,22050), producing the typical results:

```
      X =                      power =

  1.0e+003 *
    4.2736                      0.1234
   -4.2736
    4.2184                      0.0006
   -4.2184
```

It can be seen that dominant components are returned an array with the relative power in another. This can be used to identify dominant components within a signal.

The tone signal was a noise artefact that was present in the mixnoise signal and the various signal+noise mixes. Table 3 shows the estimated frequency of the tone for estimated via the various methods.

Method	Tone frequency (Hz)		
	Spn2	Tone	Mixnoise
fft.m (65536)	4273.1254	4.273.1254	4273.1254
specgram.m	4250 - 4280	4240 - 4320	4240 - 4280
psd.m	4275	4275	4275
pburg.m (Full signal)	4274	4.2730	4.2730
pburg.m (Representative interval)	4230	4.2743	4.2434
rootmusic.m	4.2189	4.2736	4.2732

Table 3 - The tone's frequency estimation via various methods

Obviously the value for the tone is estimated slightly differently for each method but it seems that the tone's frequency actually may be around 4273 - 4725 Hz.

The methods used have different degrees of usefulness depending on their application. Pburg.m for example, is useful for signal of limited duration whereas spec graph provides a visual representation of how the signal varies over time. The psd.m tool was found to be the most useful as it automatically produces a frequency axis based on a given sampling rate and gives the power in decibels for the amplitude spectrum. This is more useful than fft.m which simple has a amplitude value of unspecified units.

Step 6 – Signal enhancement by bandpass filtering

Filtering of the recorded signal would provide the ability to remove unnecessary components e.g. the tone, in order to provide a cleaner signal for then trying to determine occurrences of control words. This would therefore increase the signal to noise ratio. A Butterworth filter can be used to create fairly decent filter approaching an approximately ideal filter response with increasing order as seen in figure 12. Analogue Butterworth filters can be used for bandpass filtering and could be cascaded to achieve high attenuation outside of the pass bands e.g. create a higher order. In this particular case an order of 15 maybe required and may make an analogue counterpart unsuitable. Digital filters are more reliable and can be used to achieve very high orders for very accurate/tight pass bands, making them a more suitable choice.

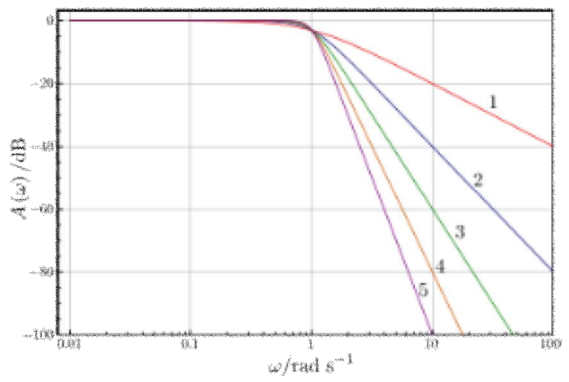


Figure 12 - Butterworth filter of increasing order[5]

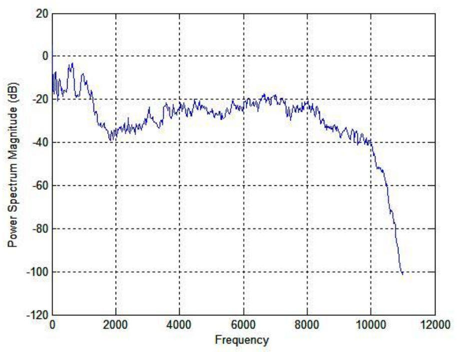
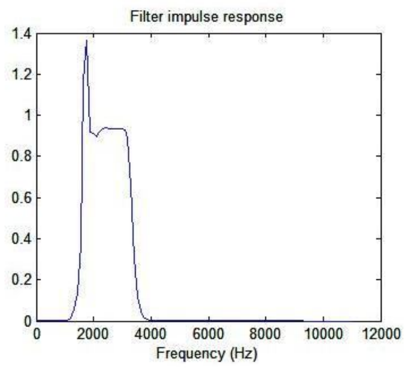


Figure 13 – Stop' signal power specral densitiy

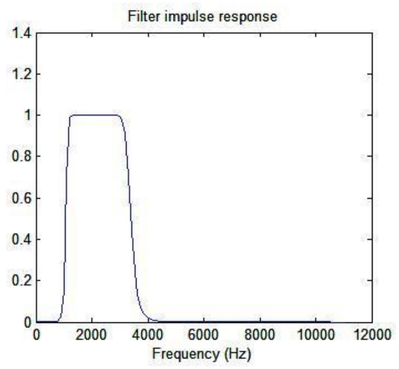
An attempt was made to use a Butterworth filter in order to try and remove undesirable frequency components. 'Butter.m' is a Matlab function that allows you to specify a cut-off frequency (or pair of frequencies for bandpass and bandstop) for the filter.

After studying the fft plot for the stop control word, a Butterworth based bandpass filter was attempted by using Butterworth with a 2 x 1 array of cut off frequencies instead of one e.g. [instead of [0.35] array of ws = [0.1,0.3]. The corresponding values can be reference to graph e.g. 0.1 ~= 1103 to 0.3 ~= 3308Hz. This was meant to be a 15th order design. Ideally, the initial filter was to cut out the tone and a portion of the noise (low frequencies). It resulted in a filter that had a response with a spike at the beginning of the pass band. The audible output was unusable for any matching process. Increasing the range from 0.1 (1103Hz) – 0.4 (4410 Hz) produced a successful bandpass filter this time obviously letting through the tone as well (tone at 4725Hz). This didn't achieve what was desired but none the less had actually performed bandpass filtering. Another bandpass filter was produced in the range 0.1-0.3 by reducing the order to 10. This was successful in reducing the level of the tone and the impulse response had no spikes. The impulse responses for the attempts can be seen in figure 14. Although this achieved suppression of the tone and a small amount of the noise, there was still room for improvement as it removed lower frequencies that relate to the stop control word (as indicated by 'stop' frequency spectrum). This was audibly obvious as the stop signal was reduced in terms of its sound level and tonal characteristics were slightly different (dulled sound). Obviously suppression of a set of frequencies means that they will be removed from both the noise and a voice signal. The key was to find a trade off between sufficient noise suppression and sufficient spectral preservation of the desired signal.

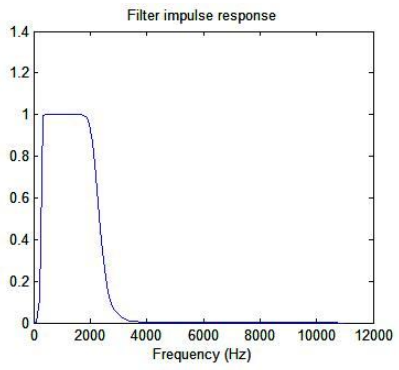
A final attempt was made to by decreasing the order to 8 with pass band frequencies (0.025 (276 Hz)– 0.2 (2205 Hz)). A slightly tighter passband was attained from at the expense of achieving a slower roll-off rate (smaller amount of attenuation) at the pass band frequencies.



Order = 15
$\omega_L = 1103Hz \quad \omega_H = 3308Hz$



Order = 10
$\omega_L = 1103Hz \quad \omega_H = 3308Hz$



Order = 8
$\omega_L = 276Hz \quad \omega_H = 2205Hz$

Figure 14

The result was that it sufficiently reduced the presence of the tone and reduced some of the lower frequencies. This has a noticeable effect of the audibility of the noise but also only slightly reduced the tonal quality of the stop control word.

Typically the human voice has a range between 80 -1200 Hz which means that anything below and beyond these frequencies maybe of little use as the spectral components are not related to the stop control word. The response for the 'stop' signal shows that a majority of the power is approximately below 1200 Hz (Figure 13). After this point the signal power is considerable small but then begins to rise again. This is due to higher frequencies captured during the recording process. The most useful range to capture to represent the human voice is 80-1200 Hz above they range may contain frequencies that contribute to the characteristics of the recorded voice signal, for example room acoustics. Thus filtering explicitly within this range would provide sufficient spectral information regarding the voice signal but the output may have impaired tonal quality. This is why the 3rd design 8th order (0.025,0.2) was chosen as this significantly suppressed as much as the noise as possible without significantly suppression of the 'stop' signal.

A clear result of experimenting with the filters is that a higher rate of attenuation can be achieved with a higher order filter e.g. after the pass bands are reached, the signal is attenuated to a smaller level at a faster rate (20 dB/ decade etc). Difficulty was experienced in trying to achieve a very small pass band with the higher order filters as the filter responses produced had spikes present in their responses indicating instabilities. Therefore the narrow pass band was achieved at the expense of the amount of attenuation that would be experienced outside the pass band.

The plots included in the presentation filter were obtained using a psd.m with 1024 points and was chosen as it clearly represents levels of attenuation in decibels. A plot of the phase response is also included and indicates non linear phase, which is a property of IIR filters.

New signal to noise values were the calculated and their there the difference determined to that before filtering. Bandpass the signal has significantly improved the signal to noise ratio and this would have the effect of the improving the matched filtering detection.

The signal to noise values were calculated in the same way as they were for step 3. they are presented at the end in the command window when the script has been executed.

An analogue equivalent of the filter was designed via the filter lab software. It was designed as a Butterworth bandpass filter of 8th order. Limitations of the software prevented the same cut off frequencies achieved above but one design was completed with the lower cut off at 280 Hz, and the higher at 1629 Hz (compared to above 180 – 2205 Hz). This would still be satisfactory and could be achieved with the circuitry shown in the schematic in the appendix (Appendix 1). The circuit uses operational amplifiers and are difficult to set up practically and there response can tend to vary, e.g. Drift. This was simple to highlight that an approximate analogue equivalent is possible, but a slightly more selective filter was created in Matlab.

Step 7 – Weiner filtering

An attempt at Wiener filtering was carried out to improve the audibility of the desired signal content. Essentially the Wiener filter requires determining the power spectral density of the two signals e.g. the desired signal and the signal plus the noise. The transfer function of the filter is governed by equation:

$$H(\omega) = \frac{W_s(\omega)}{W_s(\omega) + W_n(\omega)}$$

The flexibility of using Wiener means that you can selectively filter out regions based on a designed specification e.g. multiple pass bands. Unlike the bandpass filtering achieved in section 6, this meant that certain higher frequency components relating to the tonal quality of the stop word could be included in a second pass band. For example the fft of ‘stop’ is shown below indicates a strong frequency responses at points A and B:

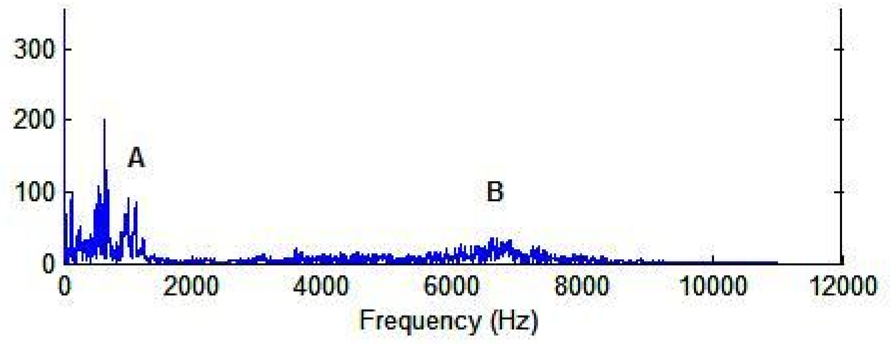


Figure 15 – 'Stop' signals fft

There it would be ideal to be able to selective bandpass filter these two ranges to make sure the filtered signal contained frequencies that are related to the stop ‘control’ word’. Obviously the tone occurs 4275 Hz so it should be included in a stop band along with the rest of the noise frequencies.

The signals used where ‘repstop’ for Ws and the representative interval for spn2 ‘repspn2’ for Ws+Wn. The power spectral densities were determined by using the psd function completing a fft of 1024 points via psd(repstop,1024,22050) etc. The ratio of the psd’s were stored into two arrays X and Y.

The ratio was then calculated for H = X./Y included the full stop to make sure all the elements of the vectors/arrays undergo the division. The resultant plot of H (include in presentation file) indicated the above ratio and the how the two psds varied with respect to one another. This was then used to develop a specification for the Wiener filter by selecting four regions which showed the ratio of the power spectral densities being fairly uniform to one another. This was used to indicate the ranges that would attempt to be filtered out whilst maintaining uniform gain for the other required ranges. The plot of H(w) in the presentation file indicates the region that were to be selectively passed and stopped by the filters. The filter specification can be seen in the presentation file indicating four regions. Essentially this formed two pass bands and stop bands with the transitions in between. Suitable values for the transition regions e.g. From a stop to pass were designed via normalising them e.g. Putting them into the range 0 – 1. The regions were decided as 198Hz – 496.125Hz for a gradual transition, 496 – 1653.75 Hz (Pass band),1653.75 – 1984.5 Hz (Transition band), 5733.1 – 6065.75 Hz (transition band), 7497 – 7717.5 Hz (Second pass region). These values can be related to the specification figure in the report and were carefully selected to make sure that the transition regions were not too small, resulting in an unstable design.

After deciding from the graph of H(w) the specification for the filter the function remez.m was used to produce the coefficients via the following syntax.

B = remez(60,[0 0.018 0.045 0.15 0.18 0.52 0.55 0.68 0.70 1],[0 0 1 1 0 0 0.5 0.5 0 0],[10,1,10,1,10]);

Weighting also had to be assigned for the different regions e.g. 1. It was found that too much of the tone signal was still coming though when the second pass region was 1, so it was changed to 0.5 as shown above. The signal was applied to the signal plus noise mix spn2.wav via the following:

filterout = filter(B,1,spn2);

After playing back the result using sound.m the 'stop' control word was clearly more prominent. The tone had been suppressed greatly and the higher and very low frequencies of the noise had been removed.
The impulse response for the filter was then obtained using freqz.m and it indicated the required response that was aimed to be designed. Also it was noted that no spikes had been encountered, indicating it was a stable design. Experimentation was carried out with the order of the created filter, and it was found that an order 50 produced fairly good results but the suppression of the tone was not as good. The order of the designed filter dictates the ripple that will be experienced in the stop and pass bands, becoming smaller in amplitude with an increase in the order. After further experimentation the order of 60 was kept as it produced a result that was satisfactory. The signal to noise ratios were calculated and are shown in the command window. The results have also been included in the presentation file.

Signal	Audibility	S/N ratio
Spn1	Significantly improved	increased
Spn2	Significantly improved	increased
Spn3	Improved	increased

Table 4

Step 8 – Adaptive filtering

The next step required the used of an adaptive filter to cancel the noise that would be present during typical signal acquisition. An adaptive filter works by dynamically changing the coefficients that dictate the transform function of the filter that otherwise would be static. This has the effect of being able to improve the effect that the filter has on the output signal by use of feedback. Noise reduction or cancellation can be achieved by using a signal that contains the noise and employing it as feedback for determining the new coefficients.

The low pass filter requirement was that it should have a pass band at 3000 Hz and an order of 2. The value needed for the filters stop band filter based on a normalised frequency scheme with 22050 Hz as the sampling period was determined as 0.272. e.g. Fs = 22050/2, then normalised value = 3000/11025 = 0.272. This was then implanted via the following syntax: `[B,A] = butter(2,0.2721,'low')`and then applied to the signal 'mixnoise' via: `adanoise = filter(B,A,mixnoise);`. Finally this was written out as the wave file adanoise.wav.

The output was then played back using sound.m and the tone (4275 Hz) had been slightly reduced in terms of its level. The psd.m routine was also used and verified that adequate attenuation had taken place.

The impulse response of the filter was determined using freqz(B,1) and can be seen in figure 16 illustrating successful implementation.

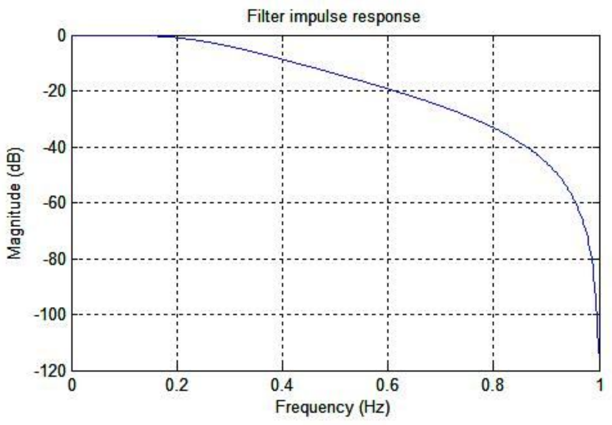


Figure 16 – Impulse response of the lowpass filter

The filtering at 3 kHz, was to ensure that the noise employed as feedback remained correlated to the noise signal that may would be captured by both microphones e.g. if spaced 10 cm, wavelengths greater than 10cm e.g 3 KHz or lower. The filtered mix noise signal then had an additional 10% of the white noise added to it. This was then used in conjunction with an adaptive filtering routine [6] with an order of 20 and the choice of μ was determined via the equations shown below:

$$0 < \mu < \frac{1}{(L+1)RMS^2}$$

In this case the peak of the signal is 1, so the RMS value is:

$$RMS = \left(\frac{1}{\sqrt{2}}\right) \quad \therefore \quad \mu < \frac{1}{(21)\left(\frac{1}{\sqrt{2}}\right)^2} = 0.095$$

Successful attempts were made by using 0.095 and smaller values. The value 0.02 was also used which gave very good results. Audibly, the voice signal was much clearer and most of the noise had been significantly reduced. The tone could not be heard anymore and only small amounts of the noise could be heard. Listening closely, the small amount of noise could be heard to vary slightly as the signal progresses. Obviously due to the adaptive filtering there is slight variation in the noise reduction over time.

Further investigation was carried out to see if the filtered signal would improve the detection by the matched filter and the results can be seen in figure 17. It can be seen that the noise mix spn1 which was the noisiest mix, produced much betters matches after the adaptive filtering.

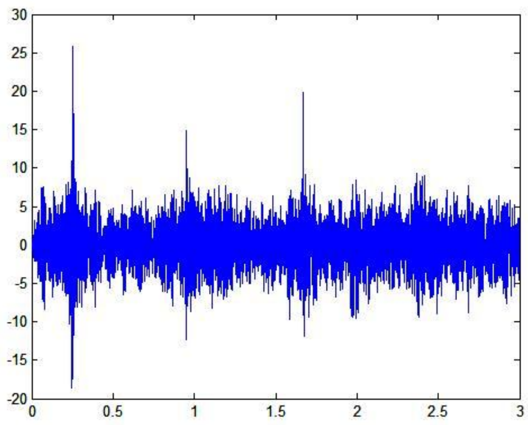
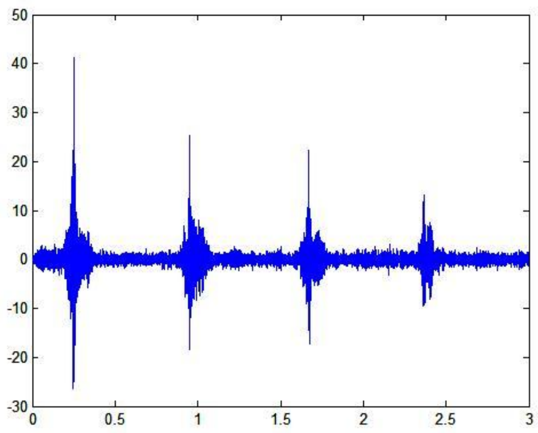


Figure 17 - Matched filter response for spn1 before adaptive filtering



Matched filter response for spn1 after adaptive filtering

This method was found to be very effective method for reducing the noise content and produced the best results. The results were as follows:

Signal	Audibility of 'stop' control word
Spn1	Significantly improved
Spn2	Significantly improved
Spn3	Improved

Table 5

Step 9 – Choice of the sufficient sampling rate

The signals were re-sampled by using the resample.m function with the syntax:

```
output_array = resample(target,1,6)
```

The re-sampling rate is expressed as a fraction of the current sample rate e.g. .current rate was 22050, resample at 1,6 re-samples it at a 1/6th of the rate or 3675 Hz.

The new resampled versions were then played back via sound.m with the new sampling frequency as the playback frequency parameter.

After experimenting with various fractions, it was found that a good trade off between preservation of the tonal quality and achieving the lowest sampling frequency was provided by a fraction of 1/6. As mentioned previously the human voice typically takes the range of 100 to 1120 Hz and therefore the minimum possible frequency should be: be 2 * 1120 Hz or 2240 Hz. A fraction of 1/8 was also used but it was felt the quality of the signal suffered to much.

Another benefit of being able to achieve a much lower sampling rate means that signal processing requires hardware that does not have to operate at as a higher speed. E.g. a cheaper DSP might be available that requires a sampling rate less than 22050 Hz. Therefore resampling at a lower rate could mean a cheaper alternative could alternative could be used.

Mp3 compression allows compression of audio files e.g. wave files, by removing parts of the signal's spectrum, thus leading to a reduction in bandwidth and ultimately the size of the file. The reduction in file size and bit rate makes the transmission of distribution files easier via the Internet and for storage on portable devices. The frequencies that are removed are deemed unimportant or produce little contribution to the audibility of the signal e.g. they may be that they are outside the audible range. Mp3 compression was carried out via use of the Lame encoder with a fixed bit rate. It significantly reduced the size of the original wave file e.g. stop.wav 129 KB -> stop.mp3 13 KB or 1/10th of the size. This is comparable to the resample method, but little audio quality is but significant file size is achieved.

The files produced for both steps were then compressed via the Win Rar compression algorithm e.g. file.rar. This allows redundant information in files to be removed and therefore the size of the file can be decreased. The results were that compressing the mp3 files resulted in no further reduction in size as they had already undergone compression. The resampled files could be compressed slightly but still were not as small as the mp3 files. The full results are included in the presentation file.

A conclusion from the mp3 compression is that it may be possible to achieve smaller sizes for transmission, but in terms of its application to DSP, it requires additional components e.g. encoders/decoders. This would increase the cost and complexity of the system but may for suitable for certain applications [7].

Step 10 – Elaboration of the system outline.

In order to form a working voice recognition system there needs to be the ability to match the signal to a known signal e.g. a control word. It was decided that the matched filter provided a suitable method for achieving this. It was also found that filtering can greatly improve the success of matching and detection so therefore a filtering stage should be included. Analogue solutions may able to provide the above two steps but once designed, are fairly inflexible and it is hard to then modify the system for a different user. Different users may need to be accommodated for, so therefore the system should provide that sort of flexibility. The proposed system is based around a DSP core which provides the filtering and matching capabilities. A DSP can perform both of these functions and can be reprogrammed at anytime in order to change the configuration. Digital signal processing can also provided superior filtering which means that a designed system can be much more selective than its analogue counterpart.

Capturing the signal requires a microphone element, some sort of preamplifier to boost the signal. The choice of DSP which will be discussed further on can provide a set of inputs with ADC capabilities. Therefore a separate ADC was not necessary. The filtering method selected was an adaptive filtering scheme which would use a secondary microphone to capture the noise signal. The two microphone could use to same preamplifier and then be fed into two input on the chosen DSP. The microphones chosen were FS43W omni directional sub miniature microphones, which are the type typically found in pc desktop microphones. These would then be used in conjunction with a suitable preamplifier e.g. a Low noise stereo preamplifier. The two microphones signal will be kept separate, but a single chip solution can be used to amplifier both signals simultaneously. The two separate signals would then be fed into a low pass anti aliasing filter. This would have a cut off that was at least the necessary sampling period (step 9) and near a value which would allow as much correlation between the other noise signal based on the distances between the microphones e.g. 10 cm e.g. 3 kHz (see section 8). This would be implemented via an analogue filter with an appropriate order e.g. 8th order Cheyshev filter. The two signals would then be fed into two input channels on the chosen DSP.

Choice of DSP

The rating of a DSP in terms of its processing capability can be hard to determine as there different ways of quantifying it [8]. These include MIPS (Millions of instructions per Second), indicating the time it takes to execute the simplest instruction. Amore useful indication is the amount of multiply and accumulates operations (MACs) that can be performed per second as this gives more of an idea of its signal processing capabilities. A suitable DSP was found that could be used as the core which was able to perform at 40 MIPS and perform multiply and accumulate operations in parallel in a single clock (80 MHz). This was taken to mean that the DSP could perform 80 MMACs. This was more than sufficient for the matched filtering and should be able to accommodate the adaptive filtering as well. If not this was actually not possible a second DSP could be used to provide the matched filtering capability. Additional features make the DSP a good choice as it has two input channels with ADC capability eliminating the need for additional analogue to digital converters at the input. The DSP also supports wait and stop modes allowing reduction in the power consumption for when the system is not in use. This is a highly likely scenario considering its application. Additionally the programming of the device can be performed relatively easily as much of the code written in Matlab could be converted into the C language for implementation. The DSP chosen was a Freescale semiconductor – 16 bit DSP 56F800.

The DSP would have four roles primarily: loading the operator's profile, adaptively filtering the input signal with the noise signal, applying this signal to the matched filter, determining the response e.g. a threshold value and taking the appropriate action output. As shown in figure 18 there would by different impulse response for the various control words and each one would generate a specific output event e.g. sound alarm, start operation etc.

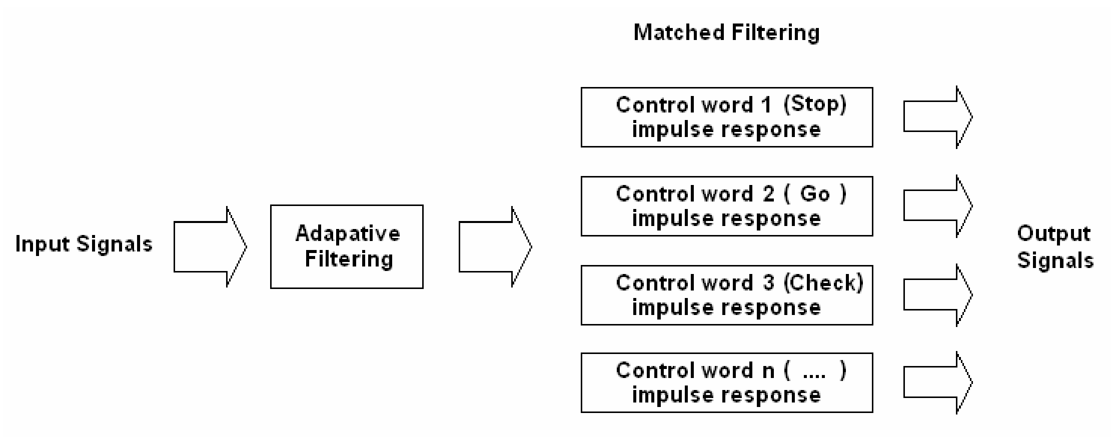

Matched Filtering
Input Signals
Adapative Filtering
Control word 1 (Stop) impulse response
Control word 2 (Go) impulse response
Control word 3 (Check) impulse response
Control word n (....) impulse response
Output Signals

Figure 18 – System diagram

Different operators could be accommodated by each operator having there own profile with their recordings which could be selected via the keypad e.g. id number. Each profile would contain an impulse response for all of the control words specifically recorded by that particular operator. Obviously this means that more memory might be needed to support large amounts of operators or many control words. External memory could be interfaced with the DSP which could be accessed when a profile is selected and then loaded into the memory of the DSP. A crude estimate for the amount of external memory necessary can be derived by consider a system calibrated for four operators, with three possible control words and the impulse response for each word having a length of 1000 samples:

1000 samples at 16 bit = 16000 bits per impulse response
3 control words = 3 filter impulse responses = 48000 bits
*4 operators = 4 * 48000 = 192000 bits =* ***192 kB*** *of extra memory.*

A final note is that a DSP development kit maybe needed for the initially development stage which would allow the testing of code etc. The price for this could not be found, but can be obtained from the manufacture. A possible estimate is an additional £150.

Cost and Development Time

When it comes to estimating a time scale for the development there a re many factors to take into consideration. In terms of a time scale, Initially the development of the system would require a prototyping stage which would allow a working system to produce for a given specification. Apart from the cost of the components a significant additional cost is the labour time spent coding and debugging the system. This can often be a lengthy process and is hard to predict how long it will take. The choice of the DSP can directly affect this as the quality of the development tools that the device supports can affect how easy it is to program and how quickly it can be brought to market. A very cheap DSP may save money in terms in the component costs, but may require a longer developing period, counteracting the savings that are made [9]. For example if the developed by a single engineer over three months, based on the average salary of a graduate engineer e.g. £23,000, then the labour cost might work out to be approximately £5750. The validity of this estimation is open to interpretation but it might be able to be used to estimate the typical labour costs expected at the development stage.

It would seem that assigning more people to a task would ensure that it would completed more quickly, but with software engineering related projects, this is not always the case. Having many people can simply mean that there are more people that need to take part in the communication process to ensure that people are doing things correctly and as a team which can lead to time being wasted simply giving explanations [10]. This obviously increases with the team size, so therefore the size should be kept as small as possible e.g. 2 – 3 people. Estimations of the length of the time it may take to develop software also may not turn out to be the case. There are a variety of different methods to estimate development time. A five step evaluation [11] routine was applied to the initial 3 months of development time and it generated the following results:

E = Best original time (3 months)
2(E) = Best case scenario (6 months)
4(E) = Most likely time in reality (12 months)
8(E) = Worst case scenario (2 Years)

It can be seen that the time needed for development could increase very quickly, substantially changing the initial labour costs needed for development. Based on this and assuming everything went well, the initial labour costs might expected to be approximately (£11500 e.g. 6 months).

The raw component cost for one time purchases might be expected to be as seen in table 6. Obviously when large amounts of the finished design are ready to be produced, components can be purchased in bulk. This normally means components can be purchased at a reduce rate, thus lowering the unit cost further. The preamplifier used for the development stage may also be able to be produced more cheaply by the use of simple transistor circuits.

Component	Source	Price
DSP Freescale semiconductor – 16 bit DSP 56F800 (Quantity 10 +)	Farnell	£ 5.33
Low noise Pre Amplifier (N47FL) – If necessary	Maplin	£4.99
Low pass filter (Built out of basic components)	Non specific	£1.00
Microphone	Maplin	£2.49 x 2
LCD - EVERBOUQUET MC0802A-SGR (Optional) (Quantity 10 +)	Farnell	£5.30
Keypad – Keypad 12 Key (Optional)	RS components	£4.21
Miscellaneous cost (Unknown extra cost)	Non specific	+/- £2.00
Total unit cost - Without Options		£18.33
Total unit cost - With Options		£27.81
Development labour cost (3 months)		£5750
Total		**£5777.81**

Table 6 - Prices for 2008 [12]

Once the development stage had been completed and rigorous testing had been completed, the code can simple be loaded onto bulk purchased DSPs. The DSPs would then obviously be mounted onto boards and connected to the rest of the components. The typical unit price then might be expected to be as follows:

DSP Freescale semiconductor – 16 bit DSP 56F800 (Quantity 10 +)	£ 4.44
Pre Amplifier – If necessary	£1.00
Low pass filter	£1.00
Microphone	£2.49 x 2
Miscellaneous cost	+/- £1.00
LCD - EVERBOUQUET MC0802A-SGR (Optional) (Quantity 10 +)	£4.51
Keypad – Keypad 12 Key (Optional)	£4.21
Total - Without Options	**£13.42**
Total - With Options	**£22.14**

Table 7 – Final unit cost

This shows that the majority of the expense would be incurred at the development stage and this will hopefully be accounted for by the sales of the final units. This would be for a basic system without external memory for multiple operators, where the price will obviously vary accordingly.

Step 11 - Research and reflection

A change in operator is a very real possibility and a good system should be able to accommodate for this. The ‘stop’ control signal recording was replaced with two other recordings by two other people one male (Figure 19) and the other female (Figure 20) (from information pack). Initial examination of the signals in time domain showed different sound levels in terms of amplitudes. Obviously the occurrences of the ‘stop’ word were at slightly different times compared to the previous operators ‘stop’ signal. Representative intervals were also determined for these new ‘stop’ signals e.g. repstop = stop(1:12000) etc. This needed to be altered for the new recordings. Also the choice of the 1000 samples for the impulse response of the matched filter also had to be changed to make sure the portion of repstop was suitable chosen. After this was performed the filtering was carried out and a result of matched filtering of the spn3 mix can be seen in graphs. The result was slightly poorer matches with the maximum amplitude being comparatively smaller. The response for the female voice was even worse possibly due to a combination of the sound level and the different spectral properties as a female’s voice has a different frequency response. Therefore a practical consideration might be to have a system that allows various operators to calibrate the system by stating the control words and having them stored in memory or a dynamic threshold could be set every time to allow proper determination of an instance of a control word being stated.

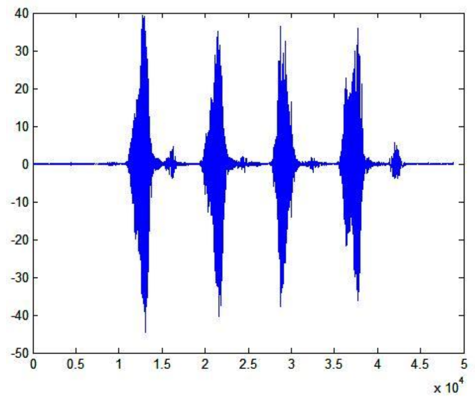


Figure 19 - Matched filter response (Male operator)

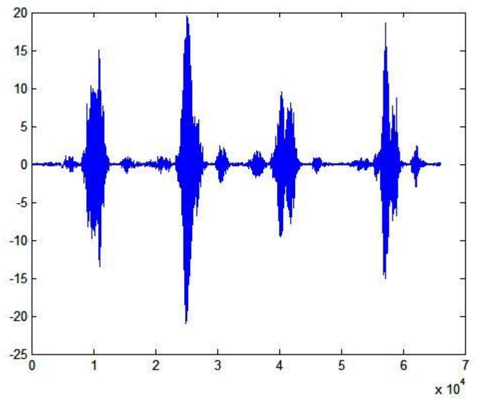


Figure 20 - Matched filter response (Female operator)

Further experimentation was carried out to check the response of the matched filter given other control words e.g. go and check. It can be seen from figure 21 and 22 that the response are not very strong for clear loud recordings of the other words with the maximum amplitudes never exceeded 40. This therefore could be used as a threshold setting as clear loud instances of stop will cause spikes that will exceed this value. Therefore discrimination against other words was to an extent achievable via the current system. A recommendation for better discrimination could be to use very different words for the appropriate control words to guarantee very different time and frequency domain representations to that of stop e.g. instead of go, commence or start could be used as the phonetics and sibilance caused by pronunciation produces a more unique time domain waveform.

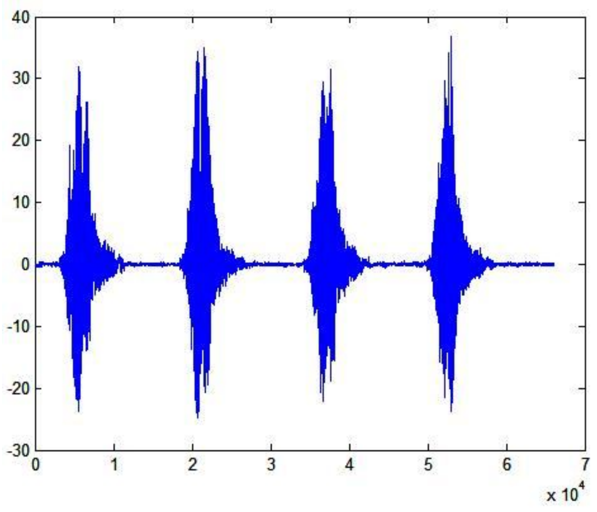


Figure 21 - Go Response

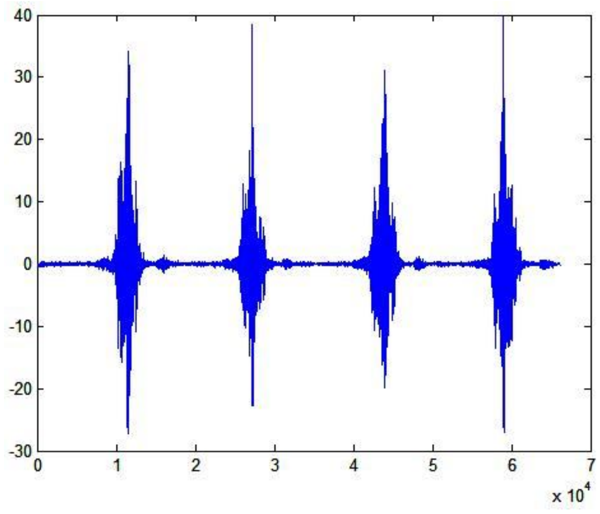


Figure 22 - Check Response

A practical system would need to have recordings of the various matched filters for each control word and these would need to be applied to the input signal in turn, thus determining which control word was stated.

Once investment has been made into the development of a product or technology it makes good business sense to find as many different applications it can be used for. Voice recognition can be used to improve the safety of certain task and to general makes tasks easier. Voice controlled GPS is a current application, where the driver can control their GPS system via spoken commands rather than having to use their hands for operation. This makes the process a lot safer as the driver can remain fully in control. Environmental control of a building might employ voice recognition in order to eliminate the need to physically go to a location to throw a switch e.g. lightning systems, temperature control. Another application of voice recognition is for people with severe disabilities and mobility issues. Being able to carry out tasks without physically having to become involved can make life easier for people in this situation and is a potentially large market for products featuring this kind of technology. A final use could be within children's toys to mimic a type of real life interaction with a toy, for example it being able to respond to commands issued by the child. Higher performance/more complex systems might be needed to achieve the above alternative uses though.

A final note is that filtering and basic signal processing can also be handled by microcontrollers as well as DSPs[13]. It was found however that many DSP were available that were similar in price to their equivalent microcontroller but more suited to audio processing applications e.g. built in ADCs. Therefore it was deemed that a DSP might be more suitable choice.

Bibliography and references

[1] - Electronic and electrical engineering – Principles and practice – Chapter 25 Digital communications - Third edition – Lionel Warnes – 2003 – ISBN: 0-333-99040-4

[2] - Digital Signal processing – DSP & Applications – Chapter 2 – The analogue–digital interface– Dag Stranneby – 2001 – ISBN: 0-7506-4811-2

[3] - Engineering mathematics Chapter 30 – 37 - Sampling and estimation theories/Normal distribution – John Bird – Third edition – ISBN: 0-7506-4110-X

[4] – Signal-to-noise ratio – wikipedia – – Revised version - February 2008 http://en.wikipedia.org/w/index.php?title=Signal-to-noise_ratio&oldid=193269470 (permanant link)

[5] – Image taken from - The Butterworth filter – WikiPedia - http://en.wikipedia.org/w/index.php?title=Butterworth_filter&oldid=208387544

[6] – Lecture 7 – Adaptive signal processing – Page 24 - H64SP2 – Dr Kalashnikov

[7] – Cambridge Silicon Radio – DSP MP3 player with wireless headphones – http://www.csr.com/bluecoreplayer/technology.htm

[8] – DSP Controllers Bring New Level of Functionality to HVAC and Security Systems – by Richardhttp://www.hometoys.com/htinews/dec01/articles/motorola/mensik.htm Mensik -

[9] – Chapter 3 – Page 44 - DSP Software Development Techniques for Embedded and Real-time Systems - Robert Oshana - ISBN-10: 0750677597

[10] – The Mythical Man-Month: Essays on Software Engineering, 2nd Edition – Frederick P. Brooks.

[11] – Quickly estimate software development time in 5 steps.

http://www.jasonhanley.com/blog/2007/02/09/howto-quickly-estimate-software-development-time-in-5-steps/

[12] – Prices for components were for May 2008 from the following websites:

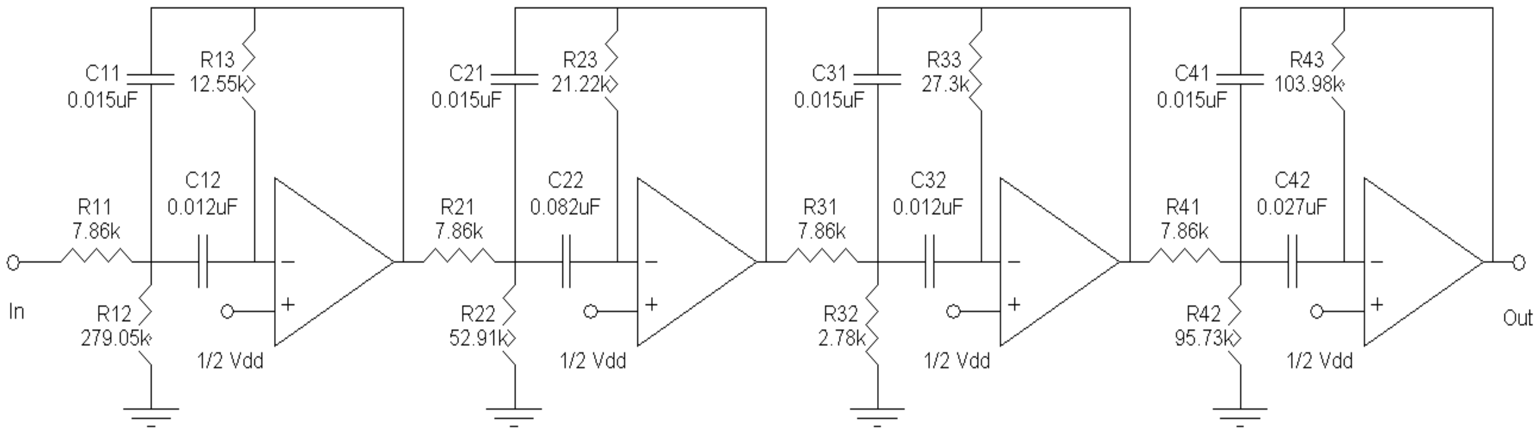
- Farnell - www.farnell.com
- RS Components - rs.www.com
- Maplins - www.maplin.com

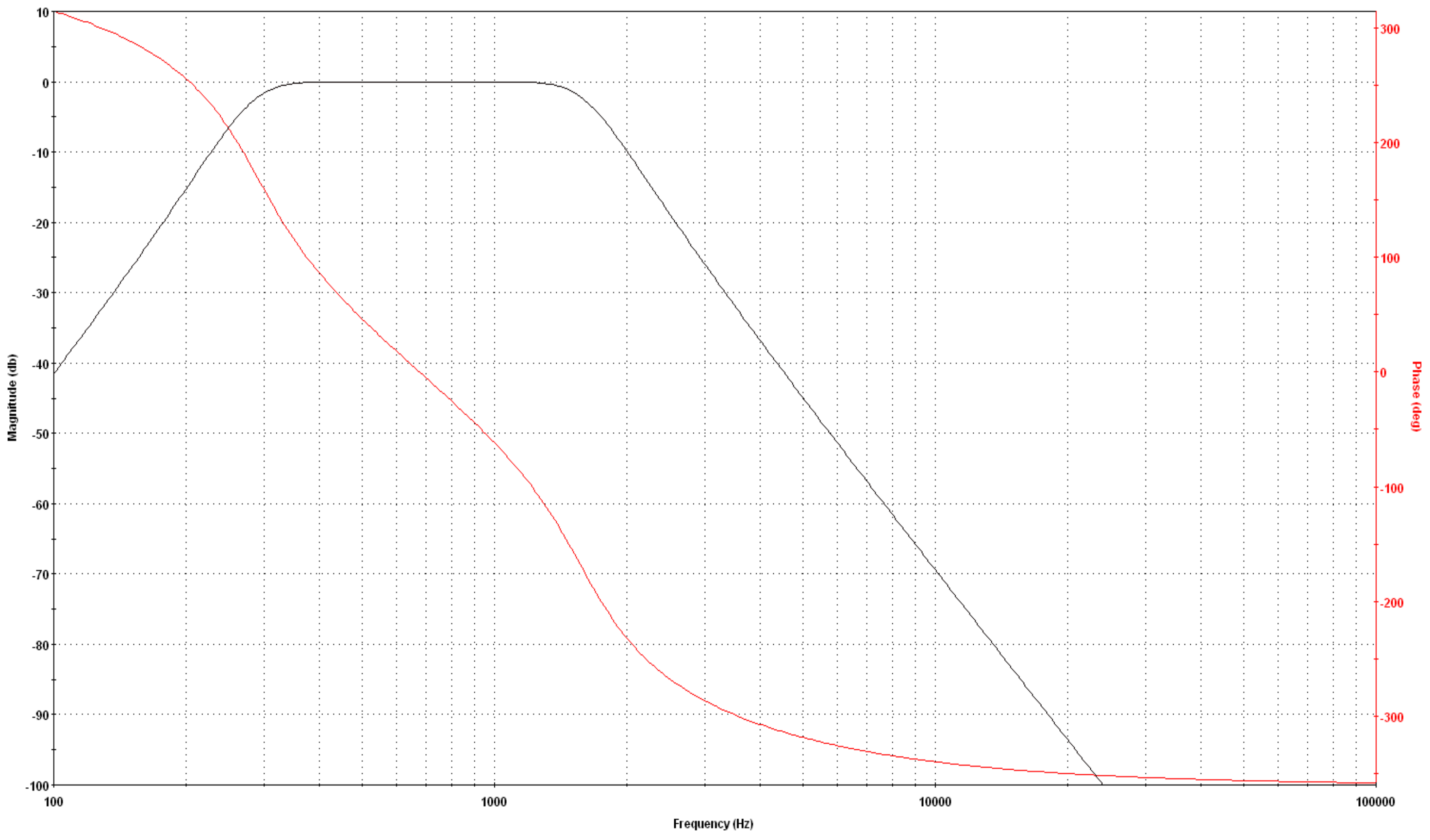
[13] – VOICE RECOGNITION SECURITY SYSTEM - ECE 476 Spring 2006 - Final Project: Voice Recognition Security System by Xiaowen Lu (xl76) and Shihjia Lee (sl362) http://instruct1.cit.cornell.edu/courses/ee476/FinalProjects/s2006/XL76_SL362/XL76%20SL362/index.html

APPENDIX

Appendix 1 – Analogue filter equivalent designed using Filter lab

8th order – Bandpass, Lower cut off = 280 Hz, Upper 1629 Hz.


C11
0.015uF
R13
12.55k
C12
0.012uF
R11
7.86k
R12
279.05k
1/2 Vdd
In
C21
0.015uF
R23
21.22k
C22
0.082uF
R21
7.86k
R22
52.91k
1/2 Vdd
C31
0.015uF
R33
27.3k
C32
0.012uF
R31
7.86k
R32
2.78k
1/2 Vdd
C41
0.015uF
R43
103.98k
C42
0.027uF
R41
7.86k
R42
95.73k
1/2 Vdd
Out


Magnitude (db)
Phase (deg)
Frequency (Hz)