ELEC 490 Capstone Report

Safety-enabled Active Noise Cancellation Headphones

Group 07

Brenda Nkuah (10170215), Riley Wells (10185808), Eduardo Gasca Cervantes (20085115), Gavin Hugh (10177524)

Supervisor: Dr. Wai-yip Geoffrey Chan

Executive Summary

This document is a detailed report outlining the design, development and testing of safety-enabled Active Noise Cancellation (ANC) headphones. The objective of this project was to implement a conventional ANC solution that responds to critical sound cues in the user's environment, such as the user's spoken name. In such an event, the system emits the ambient noise through the headphone speakers in order for the user to safely react to their environment when necessary. The methodology used for the system design is discussed including operation principles, adaptive algorithms considered for the ANC function, and Google's speech recognition engine used to model the sound recognition program The implementation of the ANC process on a Texas Instruments TMS320C6748 DSP evaluation board is explained as well as an overview of the hardware used in the project. Both the ANC and sound recognition subsystems were tested individually against performance specifications set by the group. Testing for the ANC adaptive filter algorithm proved successful as the ANC filter was able to digitally attenuate simulated airplane cabin noise up to 1 kHz by -20.1 dB. However, the system failed to integrate with the microphone sensors required to cancel ambient noise acoustically. The sound recognition program was tested in various ambient sound environments at varying speech level intensities. The system identified the correct speech pattern with an accuracy of 65% in airplane noise at a sound level of 75 dB and a speech level of 85 dB. Future efforts will include defining required microphone specifications for both subsystems and modifying the speech recognition program to also identify less distinct critical sounds such as backup alarm sounds.

Table of Contents

1. Motivation and Background

Active Noise Control (ANC) technology is becoming increasingly more effective and prevalent in the consumer market. Existing product solutions now cancel noise to such a high degree, that important environmental cues are filtered out from the user's hearing. This lack of auditory information poses a safety risk to the user, as well as an inconvenience in situations in which their attention is otherwise required. The development of a software based smart ANC actuator which interfaces with existing hardware would effectively combat this issue.

The ability to remove unwanted noise from a user's environment, fosters better focus and relaxation which consequently increases one's productivity [1]. In optimal settings, an individual should be able to isolate themselves at will from external ambient noise to capitalize on this added productivity. Isolation can come in the form of the physical removal of one's self from the noisy environment. However, this is often not possible in urban environments, public spaces, busy offices, or industrial settings giving rise to the vast popularity of ANC headphone solutions, to help combat these situations. Globally, the ANC headphone market is expected to reach beyond $9 Billion in 2024 [2]. With such a large growth in the market, companies are striving to improve on existing ANC technologies, and add new features.

In a growingly distracted age the added usage of these technologies can cancel important environmental cues from the user's perception, leading to potential injuries. A 2011 American study found 116 cases of pedestrian injury or death, where the victim was wearing headphones at the time of the event. Furthermore, 29% of all cases mentioned a warning sound being issued before the event, such as a horn or alarm [3].

Additionally, companies such as 3M, Bose, and Sony have identified this gap in the market, and are adding manual ANC overrides to their products. However, these manual overrides require the user to be inherently aware of their situation, and thus can only perform as intended in a limited set of use cases. This further validates the market for this technology, and only further displays the need for automatic ANC disabling technologies such as the project in this report will detail.

2. Design

2.1 Requirements

The goal of this project is to provide a solution that effectively cancels external ambient noise when the user is listening to music, and quickly reacts to critical sound cues in order for the user to clearly hear what is around them when necessary. As such, the design requirements for this project were decided on to reflect these functions.

The project requirements are divided into three categories: *functional* requirements, *interface* requirements and *performance* requirements.

Functional requirements:

1.1 *Actively cancel unwanted ambient noise*: The principal function of an ANC system is to produce an inverse noise signal that, when summed with acoustic ambient noise results in zero as perceived by the human ear. The group decided to focus on providing this functionality in only one ear in order to decrease hardware costs and reduce implementation complexity.

1.2 *Recognize backup alarm sounds in a noisy environment:* One situation in which such a product would be useful is in an environment where users are in proximity with large machinery. In these situations, awareness of one’s surroundings is critical. Therefore, the system will be designed to recognize backup alarm sounds produced by large vehicles such as a forklift.

1.3 *Recognize the user’s name in a noisy environment*: In order to warn the user of their surroundings, it is essential for the system to trigger a switch from ANC mode to a mode in which the user can hear what is around them. The system must be able to recognize distinct sounds that will trigger this switch. As a proof of concept, the project aims to perform this operation when the user’s name is detected.

Interface requirements:

2.1 *Send command signals from the PC sound recognition program to the DSP evaluation board to toggle between ANC and audio feed-through mode*: In order to fulfill functional requirement 2, the sound recognition process must interface with the ANC process in order to activate and deactivate filtering. Furthermore, it must notify the ANC process to allow the feedthrough of audio from the microphones to the speakers when a user’s name is detected.

2.2 *Interface microphones and headphone speakers with the DSP*: The DSP board chosen to perform all filtering operations must handle input samples from external microphones and be able to send output samples to headphone speakers to detect and produce real-world signals.

Performance requirements:

3.1 *Attenuate machinery hum noise by -10 dB +/- 1 dB below 1 kHz*: These specifications were determined by examining the performance of a typical ANC pair of headphones. The attenuation specification is set lower than average to a more achievable objective [4].

3.2 *Recognize backup alarm sounds with 80% accuracy at an ambient airplane noise level of 80 dB*: In order for the sound recognition feature to be usable, the team determined that it would be desirable for the program to be able to recognize machine backup alarm sounds with 80% accuracy in a typical airplane noise level of 80 dB [4].

3.3 *Recognize user's name with 80% accuracy at an ambient airplane noise level of 80 dB*: In order for the sound recognition feature to be usable, the team determined that it would be desirable for the program to be able to recognize the user's name with a 80% accuracy in a typical airplane noise level of 80 dB [4].

Additionally, some major constraints were placed upon these requirements based on the budget given, and the hardware and software tools available to us. These constraints include:

- Must be inexpensive, all component costs needed to be under $400.
- Must interface with an existing pair of headphones.
- All filtering operations must occur digitally in software.

2.2 Methodology

The project has two main functions; thus, the project was broken down into two major subsystems: the active noise cancellation subsystem and the sound recognition subsystem. The active noise cancellation (ANC) subsystem includes all parts of the project involving the generation of anti-noise used to attenuate sound inside the headphones. These include the adaptive filter and an adaptive algorithm used to update the filter, both of which will be discussed in the following section. The sound recognition subsystem consists of the program used to recognize and interpret speech. The program includes a speech-to-text model and a python script that interprets results and sends flags to the ANC subsystem to toggle between modes.

2.2.1 Active Noise Cancellation Subsystem

The basis of the Active Noise Cancellation system stems from pre-existing ANC system topologies. As the goal of the project was to implement a functional ANC system onto a hardware platform and tune it to fit requirements, it was not necessary or ideal to design the entire system from scratch. Therefore, the design emphasis was placed upon choosing the ANC implementation, the type of adaptive filter at its core, and tuning parameters to achieve desired performance. The following section will describe the fundamental operation of an ANC system that serves as the basis of the final subsystem design.

The operation of an ANC system is based upon destructive interference between two acoustic signals: the incoming external noise signal desired to be cancelled and an identical noise signal of equal amplitude, 180 degrees out of phase generated by headphone speakers. Assuming this interference occurs before the unwanted noise signal reaches the user's ear, the user will perceive the resulting attenuated noise signal. An adaptive filter is responsible for replicating and inverting incoming noise signals in real-time to produce the speaker output. All ANC systems require such a filter since the noise signal undergoes an acoustic transformation from the outside to the inside of the headphone earcup in the form of passive attenuation.

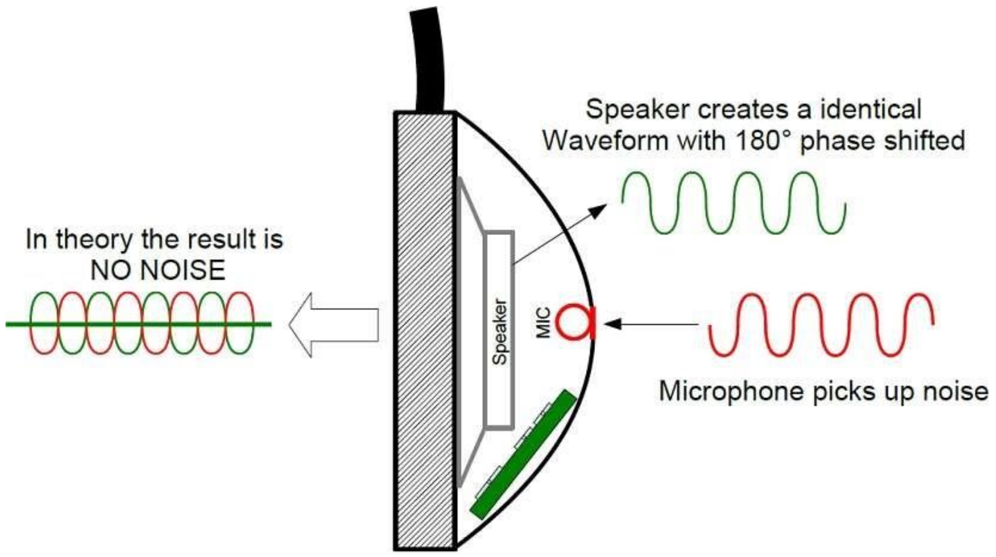

Speaker creates a identical
Waveform with 180° phase shifted
In theory the result is
NO NOISE
Speaker
MIC
Microphone picks up noise

Figure 1: Basic overview of a typical feedforward ANC system [16].

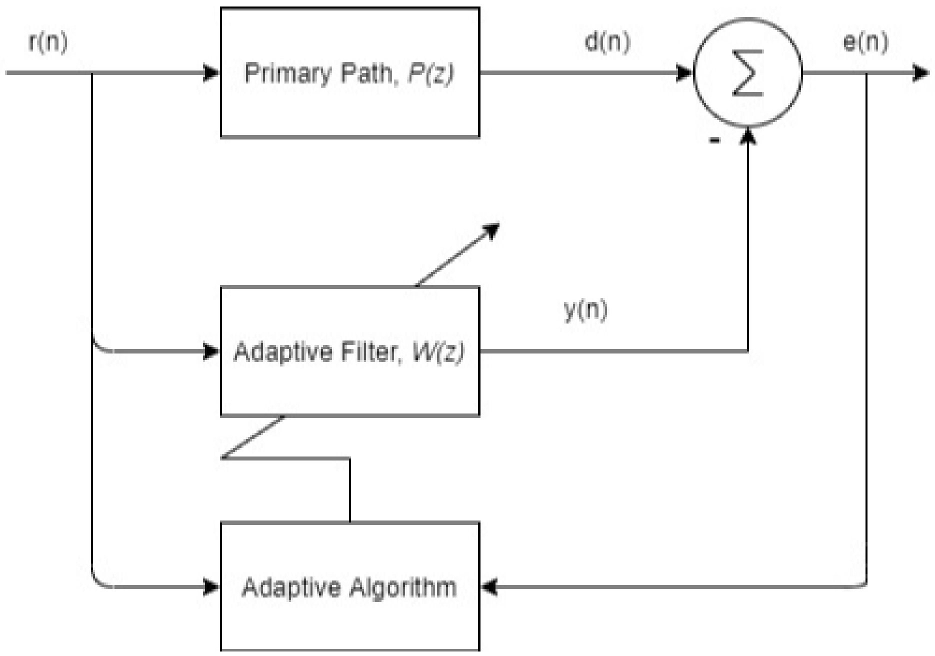


Figure 2: Block diagram of an ANC system.

Figure 2 above is the basic principle of an adaptive filter. r(n), as it pertains to the ANC system, is the digitized ambient noise signal captured by a microphone on the outside of the earcup. d(n) is the acoustic ambient noise signal resulting from the filtering of r(n) as it passes through the earcup to the user's ear. This unknown path is referred to as the primary path, P(z).The goal of an adaptive filter, W(z), is to minimize the signal that reaches the user's ear, $e(n) = d(n) - y(n)$, where $y(n) = r(n) * w(n)$, by choice of w(n). Clearly, when r(n) is filtered by w(n) such that the resulting signal $y(n) = d(n)$, e(n) is zero. An adaptive algorithm is used to update the weights of the filter, w(n), given the past samples of e(n) such that this matching occurs. While there are many variants of adaptive algorithms that could be employed, the Least-Mean-Square (LMS) algorithm was chosen to begin the design process due to its simplicity. Choosing a simple filter algorithm initially allowed the team to learn and observe how key parameters affected the overall performance of the system. The algorithm uses the gradient descent method to update filter weights such that the mean-square error between d(n) and y(n) is minimized given previous input and output samples. Each coefficient update is a step towards the minimum point of the gradient function.

The following equations explain how the coefficients are updated and how the output of the filter is calculated [5]:

$$w_i(n+1) = w_i(n) + \beta e(n) r(n-i) \tag{1}$$

$$y(n) = \sum_{i=0}^{M-1} w_i(n) r(n-i) \tag{2}$$

The algorithm depends on two key parameters which determine speed of convergence and stability:

1. ***β***, the step size of the gradient descent. Its value is dependent on the power spectral density of the reference signal, r(n).
2. **M-1**, the length of the adaptive filter or the number of coefficients characterizing w(n).

Both parameters have a noticeable impact on the performance of the system. Increasing the step size of the algorithm can drastically increase the speed at which the unknown system, P(z), is identified. However, doing so will tend to drive the system unstable. The length of the adaptive filter determines how accurately the unknown system can be estimated. A large filter can utilize a larger set of coefficients than a smaller filter in its estimation. However, a larger filter adds more computational complexity to the system, reducing the time that the filtering program has to perform other operations.

2.2.1.1 MATLAB Simulation

To begin development and the design of the ANC algorithm, a model of the algorithm was developed in MATLAB. The MATLAB simulation consisted of generating the reference microphone and error microphone signals and applying the ANC algorithm described in Figure 2: Block diagram of an ANC system.. Conducting simulations simplified the design process of the algorithm by allowing the team to focus on tuning a few key parameters to achieve desired system outputs. These parameters included the ideal number of filter taps used to estimate the primary path and the learning rate of the adaptive filter.

The reference microphone signal used was a collection of harmonic sine waves with a fundamental frequency of 60 Hz that mimicked the industrial sound of a motor. The transfer function of the headphone-to-ear primary path was modeled as a lowpass filter with a cutoff frequency of 2 kHz [6]. The reference signal was filtered by the primary path estimate to create d(n) as shown in Figure 2. The adaptive filter w(n) was created with two tunable parameters: the length of the adaptive filter M and the rate of convergence β.

The length of the filter was increased to 200 until the attenuation of the error signal stopped improving. This signal is shown in Figure 3. The learning rate was then varied to find its optimal value. Figure 4 shows that a β value of 10^{-4} enabled the LMS algorithm to converge in around 200 samples. These parameter values provided the starting point of the ANC subsystem implementation in hardware.

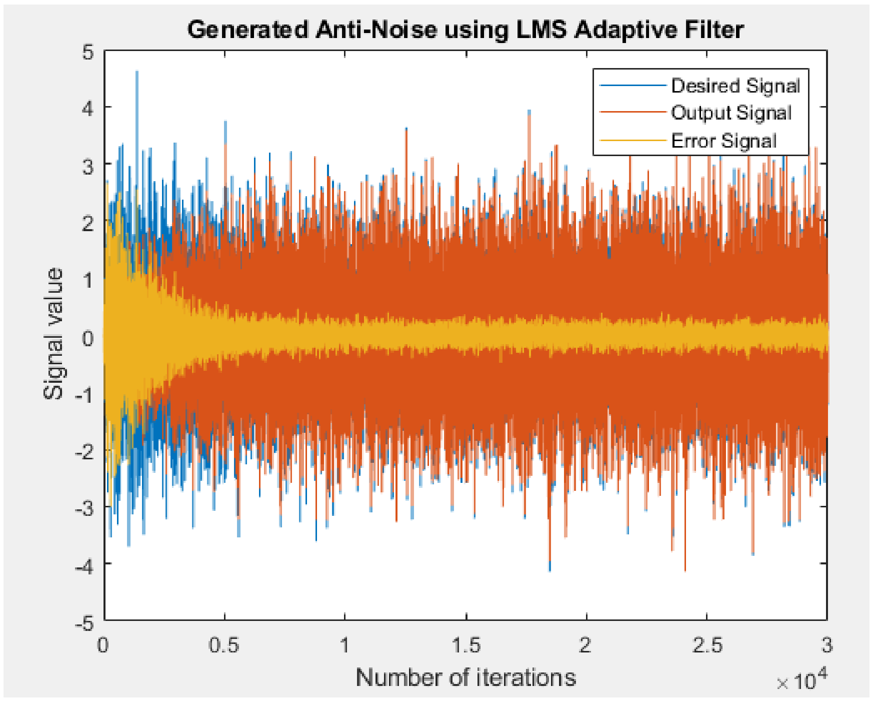


Figure 3: ANC output and error signal.

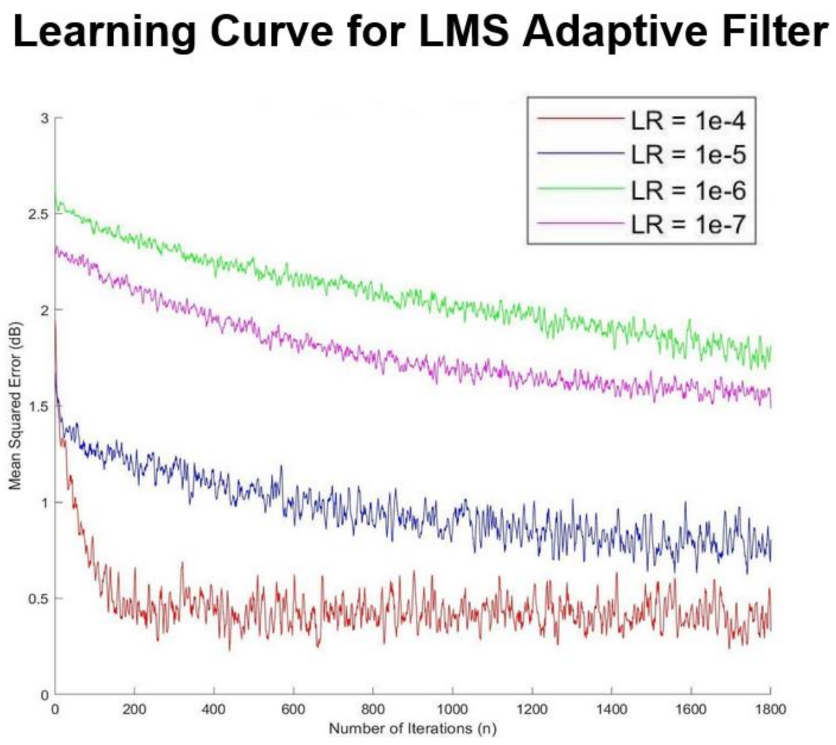


Figure 4: Various learning curves for the LMS adaptive filter.

2.2.1.2 LMS vs. NLMS

Choosing an optimal fixed step size for the LMS filter is essential to guarantee system stability. However, because the ideal value is dependent on input signal power, a fixed step size will be suboptimal if the ambient noise signal power tends to fluctuate. This is a problem encountered when operating upon random noise. An algorithm that handles these fluctuations by using a time-varying step size is the Normalized LMS (NLMS) algorithm. With the NLMS algorithm, the weights are updated using the same gradient method as in the LMS algorithm. However, the NLMS step size is normalized with respect to the squared Euclidean norm of the previous input signal sample.

$$\beta = \frac{1}{\sum_{n=0}^{M-1} \|r(n)\|^2} \tag{3}$$

Using the NLMS algorithm over the LMS algorithm results in a system that converges faster and is less sensitive to signal power fluctuations. Therefore, after testing both options, the final system is based on the NLMS algorithm.

2.2.1.3 Tradeoff between filter taps and sampling rate

In order to reduce the amount of latency in the ANC subsystem, it was decided that the signals would be processed on a sample by sample basis. This means that all computation has to be completed in under one sampling period. Therefore, all operations, including signal filtering, are constrained. Due to this fact, the number of filter taps used to estimate the primary path is limited by the input sampling rate. The sampling rate of the system dictates the bandwidth of the signal that can be obtained according to the Nyquist-Shannon sampling theorem. In order to most reliably estimate the primary path, it is desired to obtain as much frequency content as possible. However, increasing the sampling rate greatly reduces the computation time available. The number of filter taps, or length of the adaptive filter, determines the precision of the filter estimate. Computational complexity scales proportionally with the filter length. Although greater precision is desired in order to best estimate the unknown system, the degree of precision can be relaxed slightly due to the perception of human hearing.

2.2.2 Sound Recognition Subsystem

The sound recognition subsystem's purpose is to first, continuously listen for speech patterns in the surrounding environment and determine whether a keyword has been spoken. If a keyword is detected, such as the user's name, the recognition script will send a flag to the ANC subsystem to turn off ANC mode, turn on passthrough mode and allow the user to listen to their environment. Speech recognition using deep neural networks and hidden Markov models has been in use for several years now and pretrained models and APIs are available in the Python programming language [7]. These APIs include online solutions such as Google Cloud Speech and offline ones such as CMU Sphinx. A critical design decision early in development of the speech recognition subsystem was choosing an online or an offline model.

2.2.2.1 Offline vs. Online ML Speech-to-Text Model

The speech recognition subsystem was decided to be developed in python, a general-purpose programming language that allows the designer to create user interfaces, perform speech recognition, and interface the program with other devices. A key factor in the design of the system is choosing the speech recognition API that was most effective in satisfying the functional requirements of the project. Two APIs were identified as potential candidates for use in the project: CMU Sphinx and Google Cloud Speech. These speech recognition models were developed independently and offered different features, so a comparison was made to determine the optimal choice.

CMU Sphinx is an open source API that has several different versions and packages for use in mobile and desktop applications [8]. One of the packages, PocketSphinx, was chosen for consideration because it is a lightweight offline speech recognition engine. This is attractive because the speech recognition subsystem would ideally run on a mobile processor integrated into the noise cancelling headphones. The fact that it is an offline model means that the recorded speech is processed and converted to text locally without the use of an internet connection.

Google Cloud Speech is an industry leading speech recognition platform that is available for use in the SpeechRecognition Python library. This API offers an online speech recognition model that is meant for use in desktop applications [9]. The recorded speech is preprocessed on the device running the program and then sent to Google servers. The speech is converted to text and sent back to the original device. Although this method would not be viable in the final design of the noise cancelling headphones, it was a viable option for use in a prototype with online connectivity.

The performance of the two models was compared to make this initial design decision. The offline PocketSphinx model was found to be 28% less accurate than the online Google Cloud Speech model [9]. This means that using the offline model would increase the word error rate of the speech recognition subsystem, thereby increasing the chance that its performance requirements would not be met. This result led to the decision to use Google Cloud Speech to build a prototype on a PC as a prototype, focusing on optimizing the system for the highest accuracy possible.

3. Implementation

A successful realization of the proposed system needed both hardware and software that met the requirements determined in the design. Different DSP hardware solutions were considered for the ANC subsystem, but the TI TMS320C6748 was chosen because it met the budgetary constraints and specifications of the design. The Google Speech Recognition Python library was chosen to implement the speech recognition subsystem because of its availability, ease of use in development, and reliability. The entire system block diagram is shown in Figure 5.

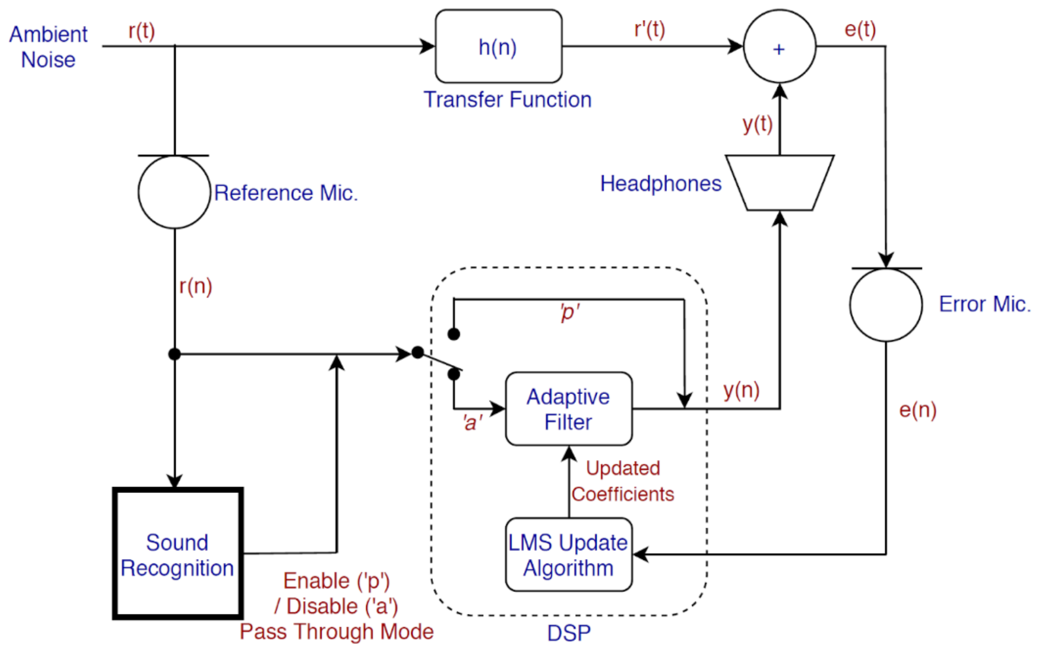
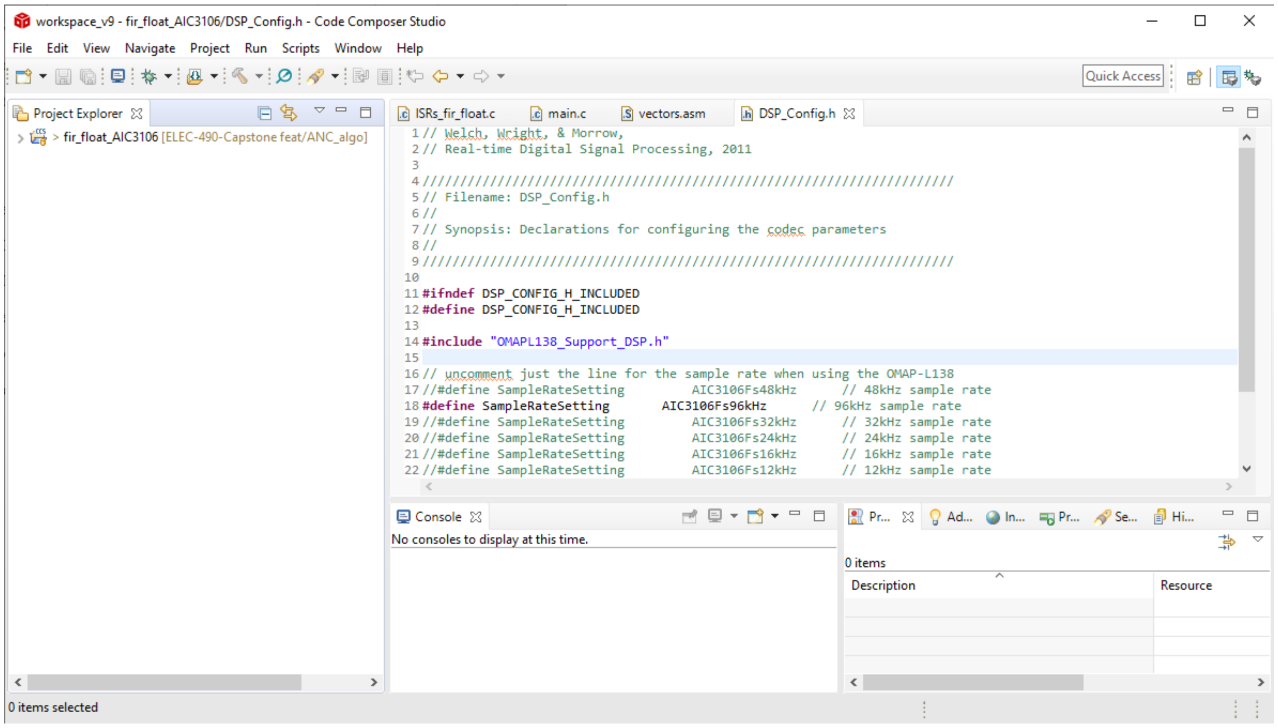


Figure 5: Overall System Block Diagram

The Texas Instruments Code Composer Studio IDE was used to develop and debug every aspect of the DSP program. The IDE allowed the use of breakpoints and stepping through code to diagnose issues in the software and hardware. It also contains an invaluable register viewer that allows the monitoring of important control and data registers in real time as the program runs. Code Composer Studio integrated with the project GIT repository containing several different feature branches, allowing different team members to develop parts of the software concurrently.

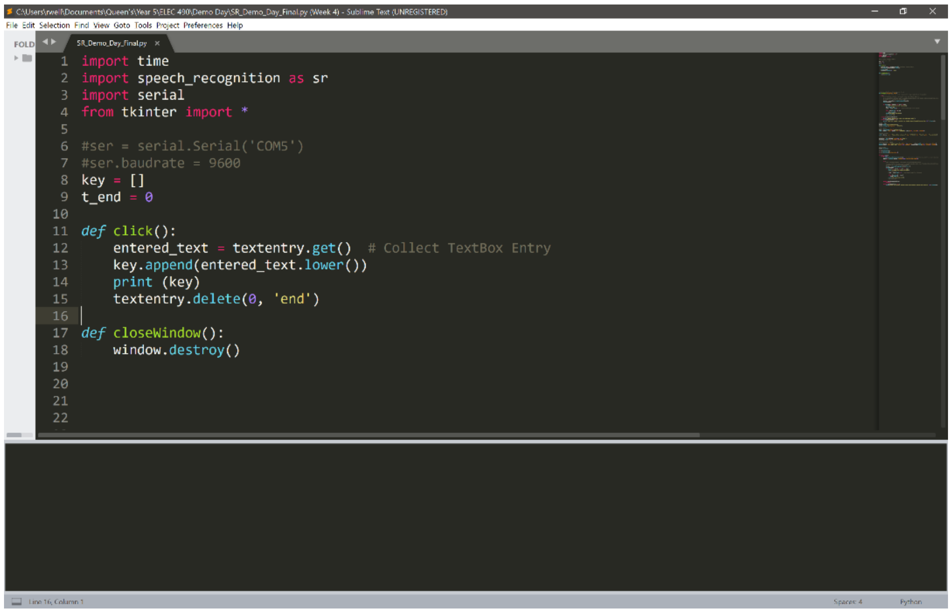

workspace_v9 - fir_float_AIC3106/DSP_Config.h - Code Composer Studio
File Edit View Navigate Project Run Scripts Window Help
Quick Access
Project Explorer
> fir_float_AIC3106 [ELEC-490-Capstone feat/ANC_algo]
ISRs_fir_float.c
main.c
vectors.asm
DSP_Config.h
Console
Pr... Ad... In... Pr... Se... Hi...
0 items
Description
Resource
0 items selected

```
// Welch, Wright, & Morrow,
// Real-time Digital Signal Processing, 2011

///////////////////////////////////////////////////////////////////////
// Filename: DSP_Config.h
//
// Synopsis: Declarations for configuring the codec parameters
//
///////////////////////////////////////////////////////////////////////

#ifndef DSP_CONFIG_H_INCLUDED
#define DSP_CONFIG_H_INCLUDED

#include "OMAPL138_Support_DSP.h"

// uncomment just the line for the sample rate when using the OMAP-L138
//#define SampleRateSetting         AIC3106Fs48kHz      // 48kHz sample rate
#define SampleRateSetting       AIC3106Fs96kHz      // 96kHz sample rate
//#define SampleRateSetting         AIC3106Fs32kHz      // 32kHz sample rate
//#define SampleRateSetting         AIC3106Fs24kHz      // 24kHz sample rate
//#define SampleRateSetting         AIC3106Fs16kHz      // 16kHz sample rate
//#define SampleRateSetting         AIC3106Fs12kHz      // 12kHz sample rate
```

No consoles to display at this time.

Figure 6: Code Composer Studio IDE showing integration with GIT (left hand tab under Project Explorer)

For the entire production of the speech recognition program, a free text editor – "*Sublime Text"* was used as an IDE for Python development. This lightweight IDE allowed for streamlined functionality while running and testing code as well as displaying console outputs.


C:\Users\rwell\Documents\Queen's\Year 5\ELEC 490\Demo Day\SR_Demo_Day_Final.py (Week 4) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
FOLD
SR_Demo_Day_Final.py
Line 16, Column 1
Spaces: 4
Python

```python
import time
import speech_recognition as sr
import serial
from tkinter import *

#ser = serial.Serial('COM5')
#ser.baudrate = 9600
key = []
t_end = 0

def click():
    entered_text = textentry.get()  # Collect TextBox Entry
    key.append(entered_text.lower())
    print (key)
    textentry.delete(0, 'end')

def closeWindow():
    window.destroy()
```

Figure 7: Sublime Text IDE

3.1 ANC Subsystem Implementation

The ANC subsystem implementation consists of a digital signal processor (DSP), two microphones, and a pair of over-the-ear headphones. The ANC algorithm shown in the DSP section of Figure 5 was implemented on a TI TMS320C6748 DSP. The audio reference and error signals were captured by two Pro JK MIC-J 044 lapel microphones fastened to a pair of HD 4.50BTNC Sennheiser headphones. The two single-channel microphone signals were combined into a two-channel signal using a Hosa stereo breakout cable. The new signal was then fed into the line-in port of the DSP. The anti-noise generated by the output of the ANC algorithm was outputted to the line-out port of the DSP to the headphones. The reference and error microphones were attached to the outside and inside of the left headphone earcup, respectively. Only the left headphone earcup was used for ANC since the DSP only had one line-in port. A laptop running the sound recognition program was connected to the DSP via a mini USB cable. The sound recognition program sends a simple control signal to the DSP in order to activate or deactivate the generation of anti-noise.

The DSP was programmed in C using Code Composer Studio. A TI XDS110 debug probe was used to load and debug programs on the DSP. This setup is shown in

Figure 8. The boilerplate code that initialized the audio codec, UART peripheral, I^{2}C peripheral, and interrupt capability was adapted from an excerpt of *Real-Time Digital Signal Processing from MATLAB to C with the TMS320C6x DSPs* by Thad B. Welch [10]. The ANC algorithm was implemented in real time inside an interrupt service routine (ISR) that was called once per received audio sample. A sampling rate of 24 kHz was chosen to maximize the processing time available inside the ISR, as all the ANC computations had to be done within one sampling period.

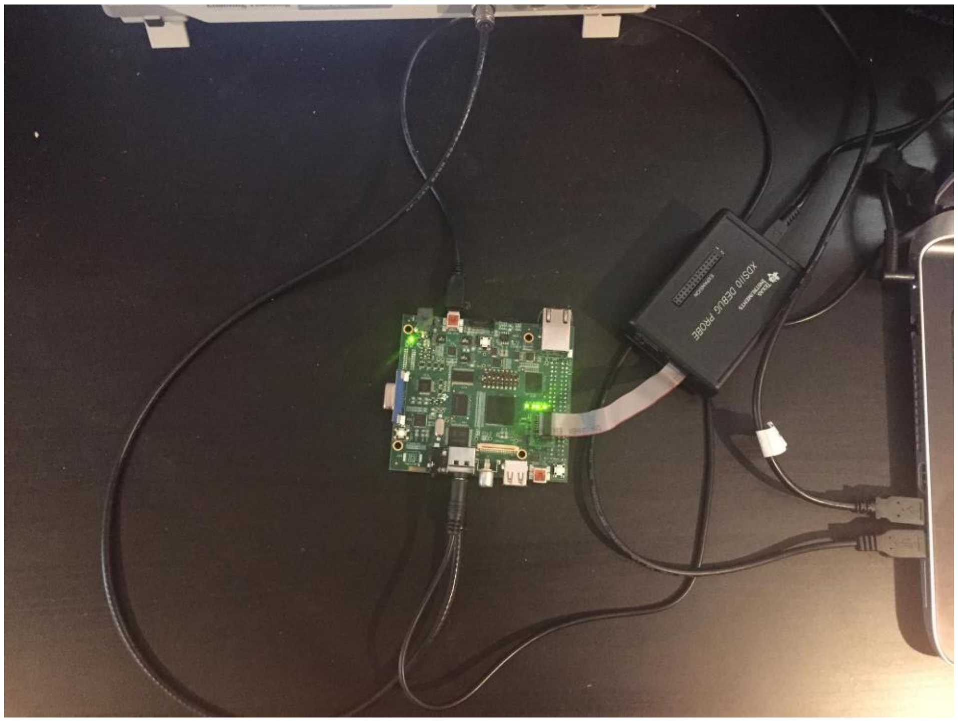


Figure 8: Photograph of DSP and Debugger

The ISR implementation of the ANC algorithm is shown in Figure 9. The data from both microphones is received by the DSP as two 16-bit words packed into one 32-bit word; the upper 16 bits are the form the reference microphone and the lower 16 bits are from the error microphone. The adaptive filter ‘w’ is an array with 80 elements initialized to zero at the start of the program. These represent the filter coefficients that are going to be adapted in the ISR.

Before any processing takes place, the ISR must determine if the system is in passthrough mode or ANC mode. Passthrough mode transfers the reference microphone signal directly to the DAC with no processing to allow the user to hear their surroundings. ANC mode is the default mode that performs adaptive filtering on the reference microphone signal to generate and output an anti-noise signal to the DAC.

The ISR first polls the UART interface between the sound recognition program running on a laptop and the DSP and reads in data if it is available. The data sent from the sound recognition program is either the character ‘a’ or ‘p’, representing ANC mode or passthrough mode respectively. If the UART is polled and a ‘p’ has been sent to the DSP indicating that a keyword has been detected, the ISR begins by reading the current ADC data into the 0’th element of a reference signal array. It then simply writes this element directly to the DAC, outputting the microphone signal to the headphones sample by sample. The DSP will continue to do this until a ‘p’ character is read from the UART, indicating an instruction to change to ANC mode.

In ANC mode, the ISR begins by resetting an output accumulator called ‘result’ to zero. The result accumulator will be the final filtered output sent to the DAC at the end of the ISR. It then reads in the reference microphone data into the 0’th element of the previously discussed reference signal array called *x_R_buffer*. This array is the same length as the adaptive filter w because the filtering process consists of multiplying every element of *x_R_buffer* with the respective element of w and accumulating each of these multiplications into the result variable. Once this is done and the output of the filter is computed, the learning rate *beta* is updated by summing the square of each element in *x_R_buffer* and taking the reciprocal. This ensures that the learning rate adapts to changes in ambient noise characteristics. Finally, the filter weights are updated using Equation 1 and the elements of *x_R_buffer* are right shifted to make way for the next sample. The result of the filtering operation is outputted to the DAC and the ISR returns to be called again once the next sample is received.

```
// Data is received as 2 16-bit words (left/right) packed into one
// 32-bit word.  The union allows the data to be accessed as a single
// entity when transferring to and from the serial port, but still be
// able to manipulate the left and right channels independently.



///////////////////////////////////////////////////////////////////////
// Filename: ANC_ISR.c
//
// Summary: Interrupt service routine for ANC adaptive filtering
//
///////////////////////////////////////////////////////////////////////

#include "DSP_Config.h"
#include "AIC3106.h"
#include <stdio.h>


#define LEFT  0
#define RIGHT 1

volatile union {
    Uint32 UINT;
    Int16 Channel[2];
} CodecDataIn, CodecDataOut;

// LMS filter parameters
#define N 80
float beta = 1e-10;        // learning rate start point, chosen empirically

float x_R_buffer[N];       // buffer for reference signal delay samples
float x_E;             // Error signal

int data_flag = 0;         //UART data received flag
char data = 'a';            //keyboard data (a = ANC mode) or (p = passthrough mode)

float w[N];             //buffer weights of adapt filter
```

```c
interrupt void Codec_ISR()
///////////////////////////////////////////////////////////////////////
// Purpose:   Codec interface interrupt service routine
//
// Input:     None
//
// Returns:   Nothing
//
// Calls:     CheckForOverrun, ReadCodecData, WriteCodecData
//
// Notes:     None
///////////////////////////////////////////////////////////////////////
{

    int i;
    data_flag = IsDataReady_UART2(); //polls status of UART receive flag
    if(data_flag != 0){
        data = Read_UART2();        // reads UART data (either 'a' or 'p')
    }

    float result = 0; //initialize the accumulator

    if(CheckForOverrun())   // overrun error occurred (i.e. halted DSP)
        return;             // so serial port is reset to recover

    CodecDataIn.UINT = ReadCodecData();     // get input data samples

    x_R_buffer[0] = CodecDataIn.Channel[LEFT];  // send left channel data to ref buffer
    x_E = CodecDataIn.Channel[RIGHT];   // send right channel data to error signal

    // ANC mode: use adaptive filter to produce anti-noise
    if (data == 'a'){
        for (i = 0; i< N; i++)
        {
            result += (w[i] * x_R_buffer[i]);   // y(n) = w(n) * r(n); calculate output to send to speaker
        beta += (x_R_buffer[i]*x_R_buffer[i]);  // calculate learning rate normalizing factor
        }

        beta = 1/beta;

        for (i = N-1; i> 0; i--)       // updates adaptive filter weights and delays
        {
            w[i] = w[i] + beta * x_R_buffer[i] * x_E;   //update weights with normalized lms learning rate
            x_R_buffer[i] = x_R_buffer[i-1];             //update reference delay samples
        }
        w[0] = w[0] + beta * x_R_buffer[0] * x_E;
    }

    // passthrough mode: pass input sample directly to output
    else if(data == 'p')
    {
        result = x_R_buffer[0];
    }

    //Return 16-bit sample to DAC
    // outputs y(n) (ANC mode) or r(n) (passthrough mode)
    CodecDataOut.Channel[LEFT] = (short) result;

    // Copy Right input directly to Right output with no filtering
    CodecDataOut.Channel[RIGHT] = (short) result;

    WriteCodecData(CodecDataOut.UINT);      // send output data to audio port

}
```

Figure 9: ANC Real-time Implementation inside ISR

3.2 Speech Recognition Subsystem Implementation

To produce the working model, Python 3.7 was used, along with four main Python packages: *time*, *speech_recognition*, *serial*, and *tkinter*. The *time* package was used to set the duration of the passthrough mode signal in conjunction with *serial,* a USB communications package which allowed information to be sent via a USB serial connection to the DSP. *Tkinter,* a lightweight GUI package, was used to create a simple interface as seen in Figure 10, allowing user input to set the alert keywords for the session. Finally, the bulk of operations used the *speech_recognition* package from Google which used online Google speech recognition servers to identify words in each audio file recorded from an onboard microphone.

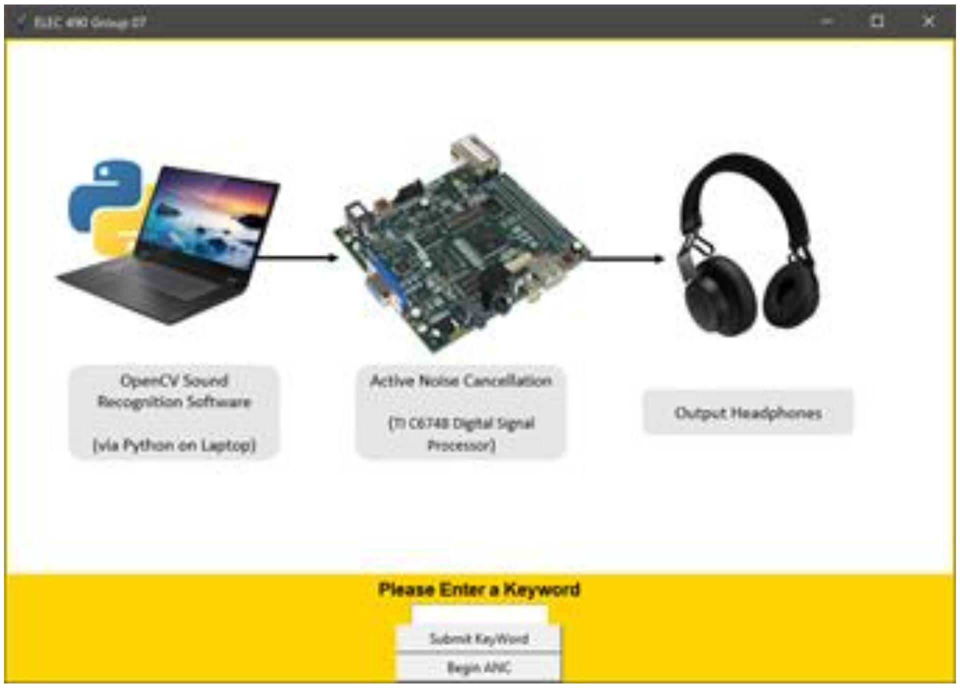
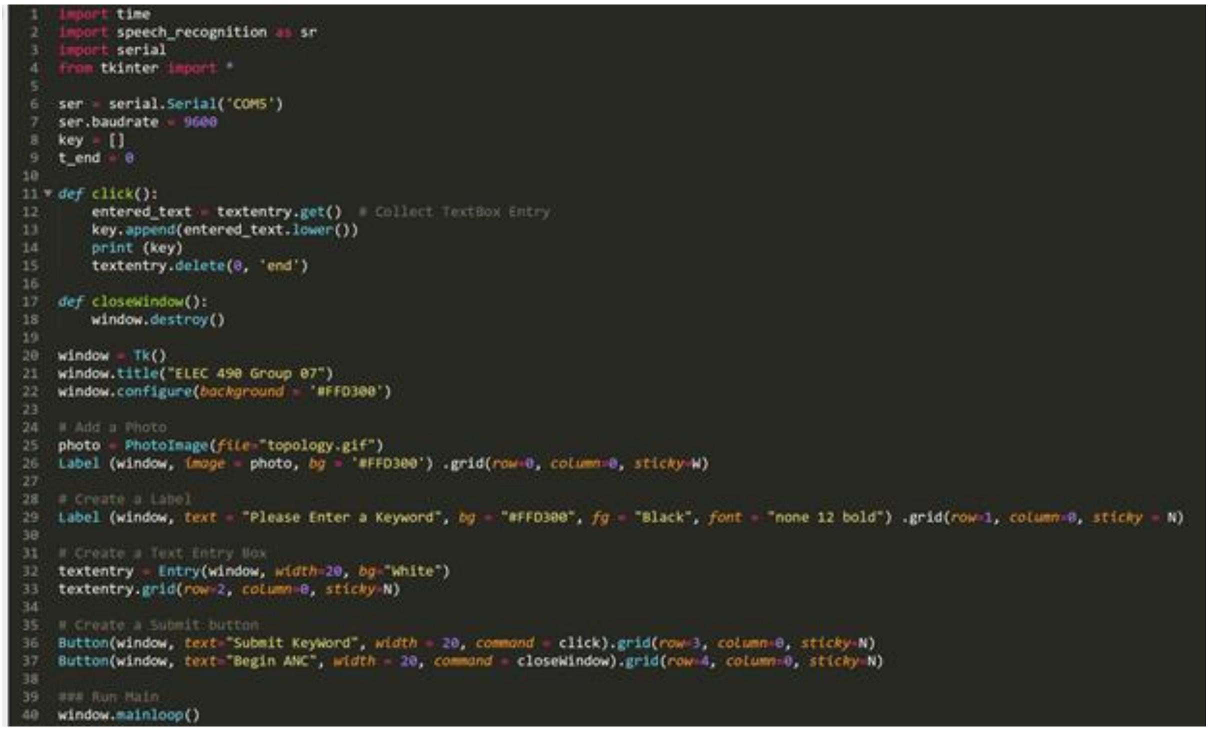

ELEC 490 Group 07
OpenCV Sound Recognition Software (via Python on Laptop)
Active Noise Cancellation (TI C6748 Digital Signal Processor)
Output Headphones
Please Enter a Keyword
Submit KeyWord
Begin ANC

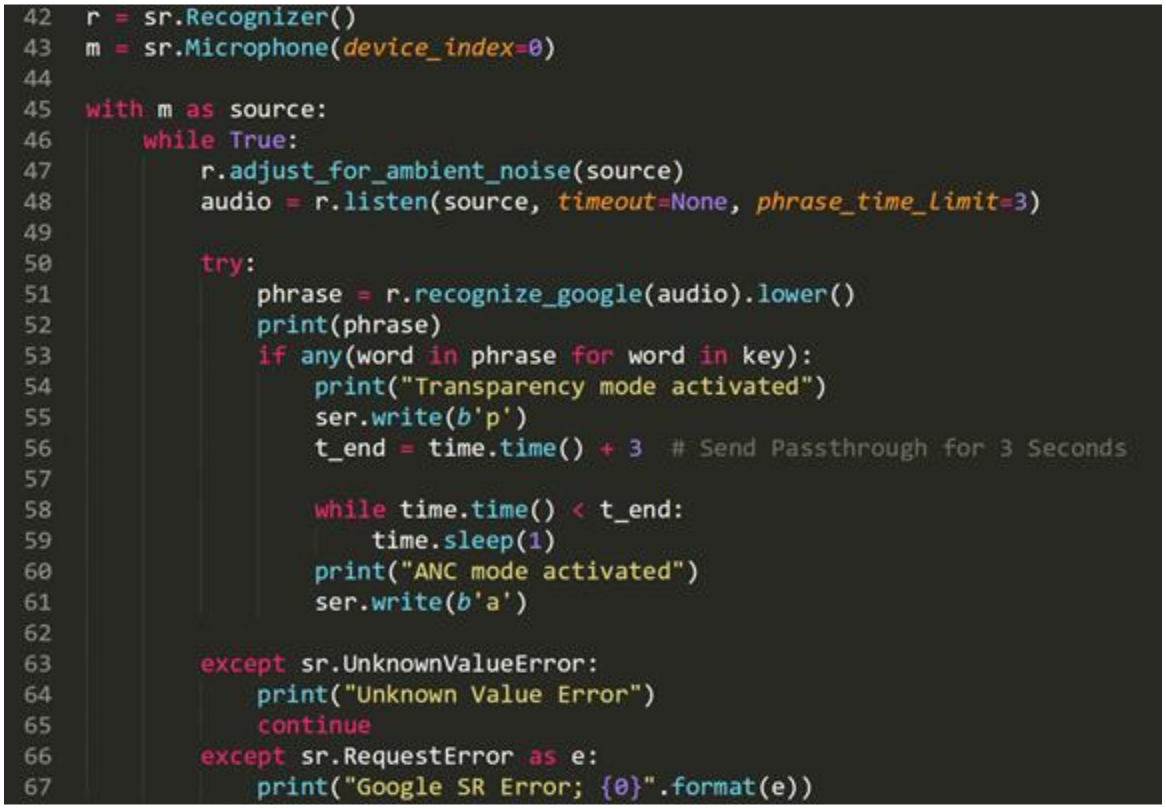
Figure 10: GUI used to accept keyword inputs and start the system.



```python
import time
import speech_recognition as sr
import serial
from tkinter import *

ser = serial.Serial('COM5')
ser.baudrate = 9600
key = []
t_end = 0

def click():
    entered_text = textentry.get()  # Collect TextBox Entry
    key.append(entered_text.lower())
    print (key)
    textentry.delete(0, 'end')

def closeWindow():
    window.destroy()

window = Tk()
window.title("ELEC 490 Group 07")
window.configure(background = '#FFD300')

# Add a Photo
photo = PhotoImage(file="topology.gif")
Label (window, image = photo, bg = '#FFD300') .grid(row=0, column=0, sticky=W)

# Create a Label
Label (window, text = "Please Enter a Keyword", bg = "#FFD300", fg = "Black", font = "none 12 bold") .grid(row=1, column=0, sticky = N)

# Create a Text Entry Box
textentry = Entry(window, width=20, bg="White")
textentry.grid(row=2, column=0, sticky=N)

# Create a Submit button
Button(window, text="Submit KeyWord", width = 20, command = click).grid(row=3, column=0, sticky=N)
Button(window, text="Begin ANC", width = 20, command = closeWindow).grid(row=4, column=0, sticky=N)

### Run Main
window.mainloop()
```

Figure 11: Serial communication and GUI initialization code.

As seen in Figure 11, the code begins with variable initializations, notably *key*, a Python list to accept keywords from user input, as well as setting the serial port ID, *ser*, and the corresponding baud rate for USB communications. Next *tkinter* is set up with window size, colours, photos, a text entry box, and buttons. Two functions are created as helpers to the GUI, *click()* and *closeWindow()*. The c*lick()* function is called on the press of the "Submit KeyWord" button shown in Figure 10, takes the user entered data from the text box, and appends it as a lowercase string entry to the *key* array. It then clears the text box in pre-emption for another input. The user may input as many keywords as desired in this manner before clicking "Begin ANC" which then calls *closeWindow()*, which simply closes the GUI and begins the main loop of code.



```python
r = sr.Recognizer()
m = sr.Microphone(device_index=0)

with m as source:
    while True:
        r.adjust_for_ambient_noise(source)
        audio = r.listen(source, timeout=None, phrase_time_limit=3)

        try:
            phrase = r.recognize_google(audio).lower()
            print(phrase)
            if any(word in phrase for word in key):
                print("Transparency mode activated")
                ser.write(b'p')
                t_end = time.time() + 3  # Send Passthrough for 3 Seconds

                while time.time() < t_end:
                    time.sleep(1)
                print("ANC mode activated")
                ser.write(b'a')

        except sr.UnknownValueError:
            print("Unknown Value Error")
            continue
        except sr.RequestError as e:
            print("Google SR Error; {0}".format(e))
```

Figure 12: Implemented sound recognition routine.

As shown in Figure 12, Once the GUI closes, the main loop of code begins by creating a Recognizer object from the *speech_recognition* package as *r* for use throughout the loop. Additionally, a Microphone object is created as *m* to be used as the audio source for the recognizer. The microphone used is indicated by *device_index=0,* which is an onboard laptop microphone.

Using the microphone object *m* as *source* for all consequential operations, the while loop is entered. In each iteration, the recognizer object *r* adjusts for ambient noise to clean the source signal. Then the packaged *audio object* is created by using the *r.listen* method of the recognizer. The arguments passed into this method are the audio source from the microphone, *source*, and a *phrase_time_limit* of 3 seconds, indicating the maximum duration without perceived speech, before saving the audio file.

Once the packaged sound file is created, a try-catch loop is entered that handles two general exceptions. UnknownValueError, which notifies via console and continues, as well as a RequestError which sends back error *e* which displays to the console and ends the program. In the try loop, the *r.recognize_google* method accepts the packaged sound file *audio* and sends it to Google Speech Recognition servers, which then return a string of the recognized words in the file which is stored in *phrase.* The string *phrase* is printed to the console for feedback, then checked for containing words within the specified *key*. If a word from *key* is found, the program writes the ASCII character 'p' to the USB, signalling passthrough mode to be activated on the DSP. The program then sleeps for a total of 3 seconds before sending the ASCII character 'a' via the same method, signalling the DSP to re-activate the ANC. Conversely, if a keyword is not detected, the loop is exited, and new microphone data is captured again. This loop runs until a user interrupt is provided, thus exiting the program.

3.3 Bill of Materials

The full bill of materials is shown in Table 1. The final budget of the project was $400 CAD provided by the electrical engineering department.

Table 1: Bill of Materials

Description	**Supplier**	**Price Per Unit ($CAD)**	**Quantity**	**Subtotal**
Pro JK MIC-J 044 Lavalier Lapel External Microphone	Amazon.ca	42.00	2	84.00
Native Instruments Traktor 8-Inch DJ Cable for iPad/iPhone	Amazon.ca	12.00	1	12.00
Hosa YMM-261 3.5 mm TRS to Dual 3.5 mm TSF Stereo Breakout Cable	Amazon.ca	13.90	1	13.90
HD 4.50BTNC Sennheiser Headphones	Sennheiser (borrowed)	0.00	1	0.00
TI TMS320C6748 DSP	Texas Instruments (borrowed)	0.00	1	0.00
TI XDS110 Debug Probe	Texas Instruments (borrowed)	0.00	1	0.00
Total Expenses				109.90

4. Testing and Evaluation

4.1 Energy Attenuation in the Desired Frequency Range

In order to test the effectiveness of the ANC algorithm, first a simulation test program was created. The simulation test was primarily concerned with the algorithm's ability to consistently and accurately estimate a predetermined transfer function and produce anti-noise that would cancel out a pre-filtered noise signal. Since the microphone sensors at this point had not been characterized and would introduce unknown effects, no measurement of real signals occurred. The reference signal was sourced from a PC soundcard using a two-way stereo aux cable, and the error signal was generated within the DSP software. As per design requirements, the test reference signal is a pre-recorded interior airplane cabin noise sound file. To simulate the transformation of the reference signal as it passes through the earcup, the signal was prefiltered with a lowpass filter. The lowpass filter was designed to resemble the acoustic transfer function of a typical high-performance passive earmuff. [4]

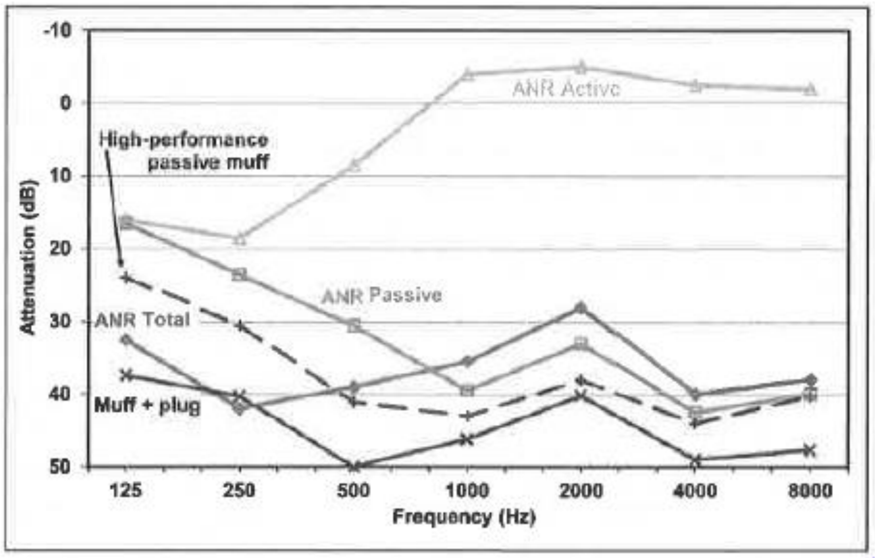
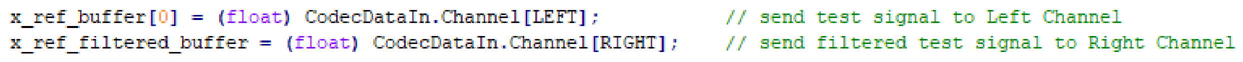


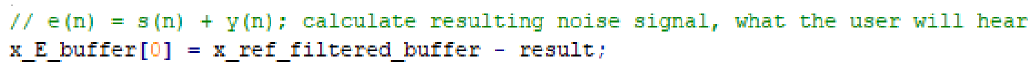
Figure 13: Frequency responses for various noise control devices.

In the simulation code, the error signal is calculated by subtracting the output of the filter estimate from the simulated ambient noise signal. The error signal is then sent to the left headphone speaker as an output. This output is effectively what the user would hear after acoustic interference of the two sound waves. Full code can be found in Appendix A.



```
x_ref_buffer[0] = (float) CodecDataIn.Channel[LEFT];           // send test signal to Left Channel
x_ref_filtered_buffer = (float) CodecDataIn.Channel[RIGHT];    // send filtered test signal to Right Channel
```

Figure 14: Left and Right channel signals, simulation test.



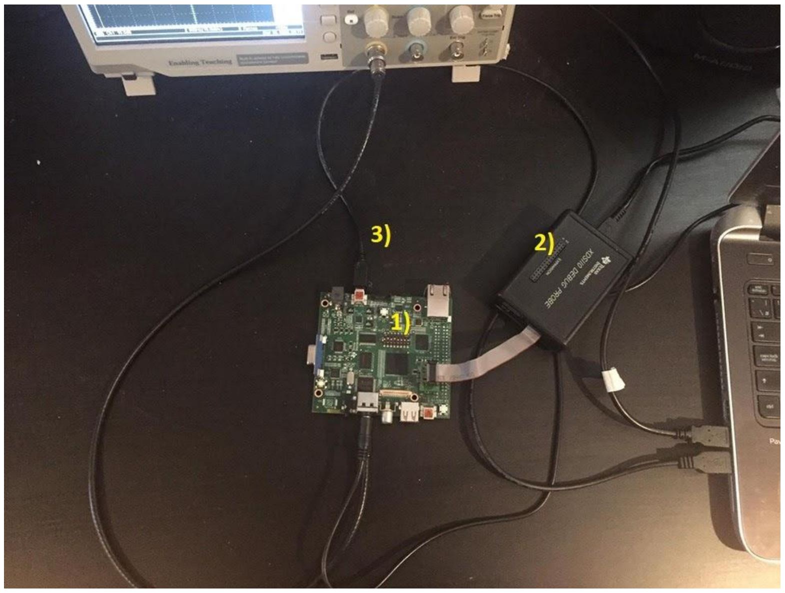
```
// e(n) = s(n) + y(n); calculate resulting noise signal, what the user will hear
x_E_buffer[0] = x_ref_filtered_buffer - result;
```

Figure 15: Error signal calculation, simulation test.

Figure 16 below details the physical connections between devices during this test. The devices and connections depicted are as follows:

1) TI TMS320C6748 DSP Evaluation Board
2) TI XDS110 Debug Probe (uploads and emulates programs from the PC to the DSP)
3) USB-to-Go 5V power
4) Stereo Line In (L - airplane noise, R - filtered airplane noise)
5) Mono Line Out (error signal)
6) Tektronix TBS1052B-EDU Oscilloscope

a)



b)

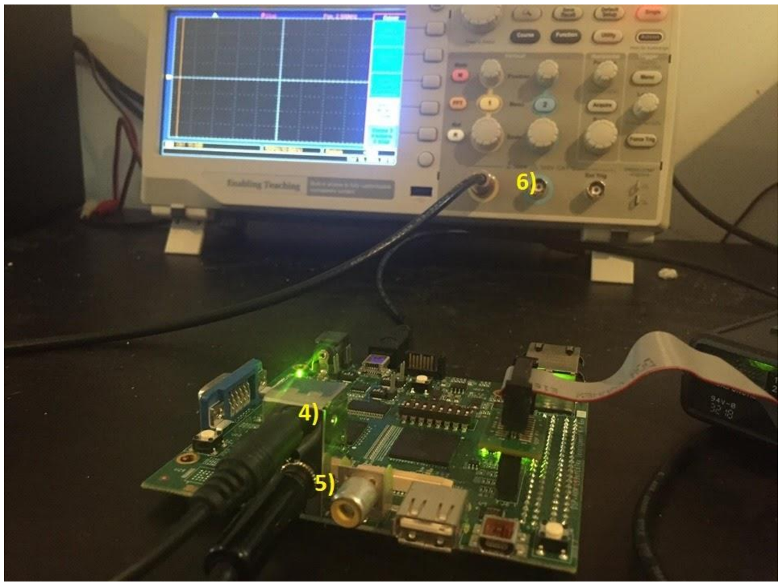


Figure 16: ANC simulation test setup, a) top-down view, b) profile view

4.1.1 Verification

The oscilloscope plots in Figure 17 and Figure 18 represent the spectral energy of the simulated airplane cabin noise as filtered by a typical headphone earcup and the spectral energy of that same noise after ANC filtering.

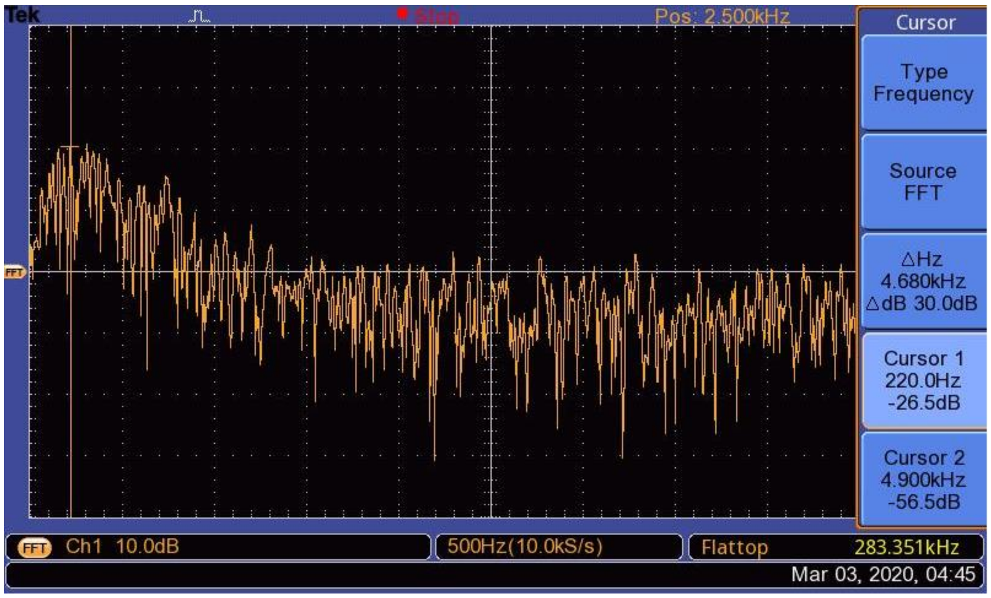


Figure 17: Oscilloscope plot of the unattenuated airplane cabin noise.

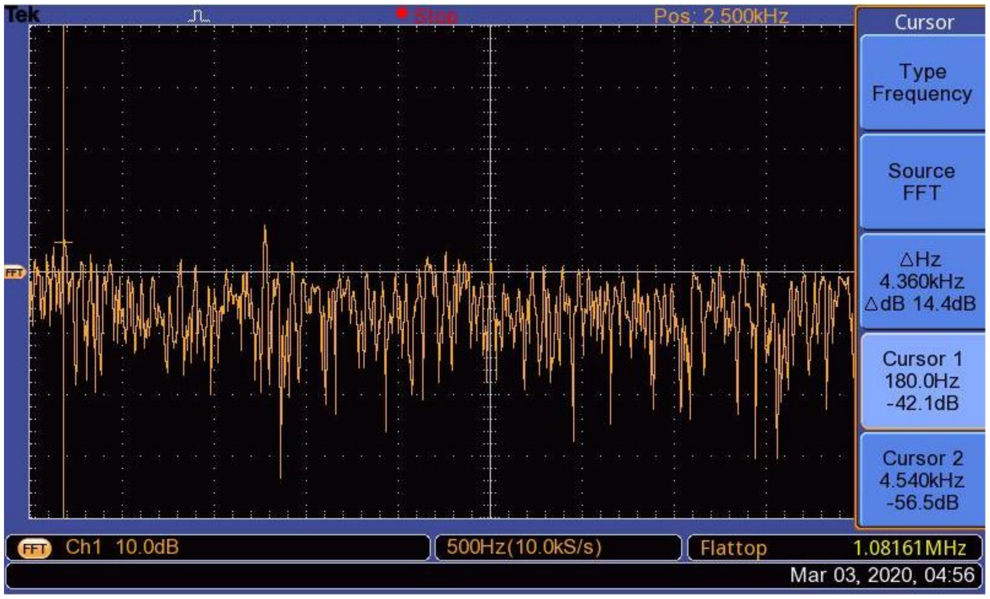


Figure 18: Airplane cabin noise spectrum after ANC filtering.

In order to verify the system and check that it fulfilled performance requirements, multiple plot points were recorded using an oscilloscope from 120 Hz to 1120 Hz, the target frequency range. In this range, there is a clear area in the first plot from 120 Hz to 720 Hz where significant spectral energy was observed. By calculating the average magnitude across this frequency range for each plot, the difference in magnitude between the noise before and after filtering was found to be -20.1 dB. This is approximately 10 dB below the required performance specification, showing that the adaptive filter works as desired.

No ANC

Table 2: Spectral data points, No ANC.

Frequency (Hz)	**Magnitude, $\Delta y = y - y_o$ (dB) (y_o @ 4.9 kHz, -56.5 dB)**
120	27.2
220	30.0
320	27.6
420	28.0
520	16.8
620	23.6
720	**20.4**
820	16.4
920	10.4
1020	11.2
1120	12.8
1220	1.2

Δy_{avg} (No ANC) = (27.2 + 30.0 + 27.6 + 28.0 + 16.8 + 23.6 + 20.4) /7 = **24.8 dB**

ANC

Table 3: Spectral data points, ANC activated.

Frequency (Hz)	**Magnitude, $\Delta y = y - y_o$ (dB) (y_o @ 4.9 kHz, -56.5 dB)**
120	10.8
220	8.0
320	3.2
420	6.4
520	0.8
620	2.8
720	**1.2**
820	11.6
920	2.8
1020	4.0
1120	5.6
1220	1.2

Δy_{avg} (ANC) = (10.8 + 8.0 + 3.2 + 6.4 + 0.8 + 2.8 + 1.2) / 7 = **4.74 dB**

$\Delta = \Delta y_{avg}$ (ANC) - Δy_{avg} (No ANC) = 24.8 - 4.7 = 20.1 dB

4.1.2 Validation

Although the test above obtained results that meet performance requirements, it fails to test the system against random ambient noise recorded by microphone. Furthermore, this test did not record acoustic feedback and use it in the system. Therefore, the system has no way of knowing whether the noise is attenuated acoustically before it reaches the user's ear. Since there is no acoustic feedback and no real-time recording of noise, the simulated system in fact sends noise to the user's ear instead of cancelling existing noise.

4.1.3 Evaluation

It can be seen that the final simulated system obtains a random noise signal that is sent into the system and reduces the spectral energy of the signal before it is played into the speakers. The system duplicates the acoustic noise that reaches the user's ear, reduces the energy of the digital copy of the noise and plays the digitally attenuated signal. However, the system does not perform any useful operations on the existing acoustic signal. Therefore, the tested system is incomplete. While the filter program meets performance specifications, it fails to meet the functional requirement of actively cancelling unwanted ambient noise in one ear.

4.2 Microphone Integration Test

Once the ANC simulation had been tested and it was confirmed that the adaptive filter was indeed performing up to specifications, it was desired to integrate microphone sensors to measure reference and error signals in the acoustic domain. The ANC program used to test the full system with microphone inputs is discussed in section 3 (implementation). Initially, each of the two microphones were plugged into the Mic In audio input of the TMS320C6748. It was then confirmed that both microphones could stream audio samples to the DSP by running the program in passthrough mode and observing the output through an oscilloscope. From there, the microphones were connected to the same Mic In audio input via a dual mono to stereo cable. Connecting the microphones in this way enabled access to two audio streams simultaneously from one physical audio input, with the reference mic streaming to the Left stereo channel and the error mic streaming to the Right stereo channel. Once connected, each mic was tested in passthrough mode again by streaming the Left and Right inputs to the Left output. It was confirmed that both mics could be streamed simultaneously by listening to the audio output recorded by the microphones.

The ANC program was then tested with the microphones. The reference mic was attached to the outside of the Left earcup and the error mic was taped to the inside of the Left headphone. The error microphone was placed approximately 2 cm adjacent to the centre of the headphone speaker in order to reduce the delay between when audio is output from the headphone speaker and received by the mic.

The code was then run in ANC mode. Immediately, positive feedback was observed in the headphone output indicating an instability within the system. In an attempt to diagnose the issue, the ANC code was run with the microphones disconnected. In a similar way, the output sample values seemed to gradually increase indicating positive feedback. Unfortunately, the team was unable to concretely define the source of the issue and resolve it within the set time constraints. Although it was never defined, the issue encountered could be likely attributed to a combination of the noise floor observed in the ANC simulation test and insufficient microphone sensitivity. It is possible that even if the microphones could sense an impulse (tapping the mic), they might not have had the sensitivity to measure incoming noise signals with enough definition to be of use to the adaptive algorithm. Hence, the microphones may not have contributed to the adaptation of the filter at all. The positive feedback observed must have been created as a result of the DSP’s electrical noise floor acting as the effective reference and error mic input signals.

4.3 Sound Recognition Test

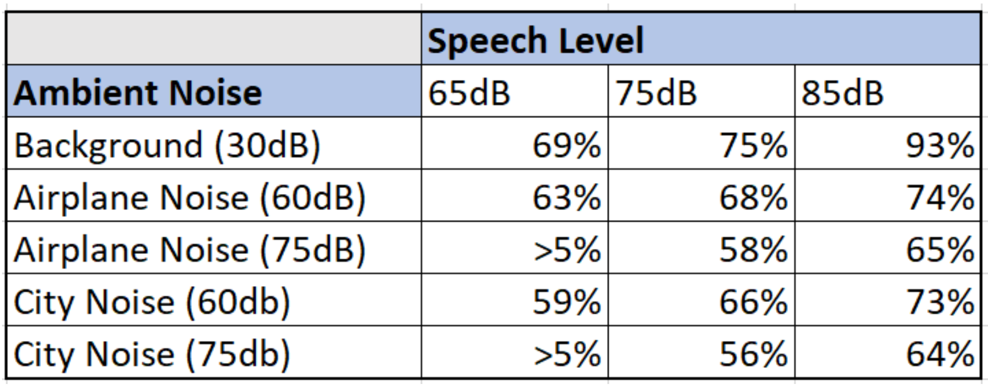
In order to perform adequate physical testing of the Machine Learning speech recognition platform, a test was carefully crafted to emulate potential use case settings of the solution. Three common noise environments were selected – Background indoor ambient noise at 30dB, Airplane cabin noise at 60dB / 75dB and City noise at 60dB / 75dB. The testing word bank consisted of 50 common English first names [11], as well as 40 of the most common English words [12]. Additionally, the following 10 potential use case words were selected: "Hello, Hey, Help, Danger, Stop, Attention, Go, No, Yes, Good". These words were sufficiently diverse to create a rich testing profile over many phonetics and sounds, to better predict the realistic accuracy of the model.

To ensure consistent speech input throughout testing, the 100 words within the word bank, were transcribed from text to speech via "Google Cloud Text-To-Speech". An American male voice was used, separating each word by 3 seconds of silence, with no emphasis or inflection on the speech output. This audio file was then downloaded and played back for usage in all tests.

To better control the noise environment and reduce variance, a setup was developed and kept constant throughout all testing. The reference listening microphone was positioned 1m away from a set of two surround sound speakers equidistantly, along with a central subwoofer to ensure a proper noise frequency spectrum. These speakers emitted the ambient noise for all testing purposes. The speech was emitted through a secondary speaker also placed at 1m from the microphone. To calibrate the decibel levels of both speakers, an iPhone 11 was placed beside the reference microphone, while running "DecibelX" – a frequency analyzing software to determine the average noise levels over a minute. The volume levels of each speaker were tuned to match the specified testing criteria as below in Table 4.

The speech recognition program was run in the above setup, without any functional alterations. Once the program identified a word, the prediction would be compared to the expected value and recorded as correct or erroneous. These results were then compiled and represented as a percent of the program's accurate predictions, as seen in Table 4: Sound Recognition Accuracy w.r.t Speech Levels vs. Ambient Noise Levels..

Table 4: Sound Recognition Accuracy w.r.t Speech Levels vs. Ambient Noise Levels.



	Speech Level		
Ambient Noise	65dB	75dB	85dB
Background (30dB)	69%	75%	93%
Airplane Noise (60dB)	63%	68%	74%
Airplane Noise (75dB)	>5%	58%	65%
City Noise (60db)	59%	66%	73%
City Noise (75db)	>5%	56%	64%

The highest accuracy rating received was 93% in an optimal setting, when the speech was emitted at a level of 85dB, over a 30dB ambient background. This compared competitively to the 95% accuracy Google received in 2017 [13]. As predicted, when the ambient noise level was greater than the level of speech, the results were insignificant (<5%). However, with speech levels and noise levels equal, accuracies of 56% - 58% were recorded. With speech levels of 85dB against 60dB Airplane and City noise, accuracies of 74% and 73% were found respectively.

It follows that Performance Requirement 3.3, "Recognize user's name with 80% accuracy at an ambient noise level of 80dB" – was not satisfied as per the above experimentation. It was found that accuracies peaked at 65% and 64% at a speech level of 85dB over an ambient noise level of 75dB for Airplane noise and City noise respectfully. Further tests were not conducted at higher noise levels, as the level of speech required for tangible results became impractical for the application of the solution, since noise would begin to transfer through the headphones at these levels.

4.4 Integration Test

A simple test was created to ensure that the ANC ISR could handle serial input data while simultaneously filtering the reference noise signal. This test consisted of adding a conditional if else statement within the ISR to switch between passthrough mode, a direct transfer of the input sample to the audio output, and ANC mode, depending on the serial byte received. After this modification, the DSP board was connected to a PC USB serial port. Using the terminal program, 'Tera Term' keyboard inputs were sent to the serial connection at an arbitrary baud rate of 9600 bps.

By sending either the characters 'p' or 'a' to the serial port, the mode of operation would toggle between passthrough ('p') mode or ANC ('a') mode. The mode switch would be confirmed by another teammate wearing headphones while the program was running. A noticeable volume change of the output airplane noise signified this switch.

5. Stakeholders

The development of the project design was framed from an industry standpoint. The two main stakeholders considered for this project were future manufacturers and users of the product. In order to ensure easy adoption of our product for manufacturers, the design was chosen to utilize off the shelf materials, as seen in Table 1. This minimizes the number of custom manufacturing processes used in the creation of the product. In addition, the platform used to run the programs can be used to run a variety of open source portable code, allowing future updates for the software. To ensure that our design is optimal for consumer use, steps were taken to ensure the safety and privacy of the user. Since the product implements a passthrough feature that amplifies the sounds around the user, it never records or transmits any of this audio data. Additionally, the passthrough mode is programmed to limit the volume level of the amplified sounds below 85 dB. This is the maximum daily sound level recommended by the World Health Organization [14].

As stated in section 3.2, implementation of the sound recognition software utilized google-speech-to-text for speech processing. The selection of this reputable 3rd party software provided transparency and reliability regarding user speech data. Data logging can be enabled and disabled on the Google Cloud platform and no user data is collected otherwise [15]. In addition, Google Cloud utilizes end-to-end encryption, negating the detrimental effects of data leaks if data logging is enabled for any reason in the future. Finally, the ECE department acted as an investor for this project. The budget for the project was $400, so a DSP was borrowed from the ECE lab to reduce costs. This reduced the final cost of the implementation and kept the project under budget.

6. Compliance with Specifications

Table 5: Compliance with project specifications.

	Specification	**Specification met? (Yes/No)**
1	**Functional requirements**	
1.1	Actively cancel unwanted ambient noise	Yes
1.2	Recognize backup alarm sounds in a noisy environment	No
1.3	Recognize user's name in a noisy environment	Yes
2	**Interface requirements**	
2.1	Send command signals from the PC listener program to the DSP board to activate/deactivate audio feed-through	Yes
2.2	Interface microphones and speakers to the DSP	Yes

3	**Performance requirements**	
3.1	Attenuate machinery hum noise by -10 dB +/- 1 dB	Yes
3.2	Recognize backup alarm sounds with 80% accuracy at an ambient noise level of 80 dBm	No
3.3	Recognize user's name with 80% accuracy at an ambient noise level of 80 dBm	No

Note: Performance specifications regarding speech recognition occurred at 3 speech levels, requirement applies to a speech level of 85 dB.

6.1 Discrepancies

Recognize backup alarm sounds in a noisy environment: Due to time constraints, development of a general sound recognition model proved to have complexity beyond the scope of the project. A pivot was made to ensure that the software integration was more robust, so general sound recognition was shifted to encompass only speech recognition. Therefore, specifications 1.2 and 3.2 were not met.

Recognize user's name with 80% accuracy at an ambient noise level of 80 dBm: In order to further clarify the performance specification, it should be noted that the speech level of the user was tested at 85 dB. The highest accuracy achieved for speech recognition at speech level of 85 dB was 93% at the lowest ambient noise level 30 dB. At an ambient noise level of 75 dB speech detection accuracy was 53%. The main factor contributing to these shortcomings was the sensitivity and dynamic range of the microphone used in the design. In the future it would be beneficial to invest in a microphone capable of higher quality audio detection.

7. Conclusions and Recommendations

Performing digital signal processing on a real-time platform requires multiple domains of knowledge: An understanding of adaptive filtering techniques and embedded hardware engineering. Creating a speech recognition model that is robust against ambient noise levels is a necessary step to creating a more functional product. Designing a custom solution that optimizes the sensors (microphones), actuators (speakers), and DSP platform together would avoid many integration problems experienced in this project. Recommendations for future iterations of this project begin with obtaining a DSP with a faster input ADC to output DAC conversion rate. Additionally, purchasing microphones with greater sensitivity to pick up more ambient noise would have enabled the ANC algorithm to process higher resolution audio.

The effectiveness of the product would have greatly improved if more time had been spent improving the accuracy of the acoustic model of the ANC system. If more research is done in order to accurately model the propagation of a sound wave through headphones and into an ear canal in two or even three dimensions, an ANC algorithm could be simulated and tuned to a much higher degree. This would increase the effectiveness of the modeling and design stage of development, improving the hardware solution by decreasing uncertainty and quickening implementation.

Another way to improve the product would be to re-envision the two major systems completely. For example, a different type of adaptive filter could be designed for the ANC subsystem, such as a recursive least squares filter. The voice recognition system could be prototyped on a portable, low-power processor and used an offline speech-to-text engine.

Currently, the market for ANC headphones has been growing at an incredible rate. The overall market growth for the over-ear, on-ear, and in-ear segments is expected to reach a valuation of $9 billion USD with an CAGR of 14% during 2018-2024 [2]. In the past year there has been a jump in the number of ANC solutions that offer a passthrough mode similar to the one explored in this report, making it quite apparent that there is demand for this product.

8. Team Efforts

Table 6: Efforts expended by each team member

Name	**Overall Effort Expended (%)**
Brenda Nkuah	100
Riley Wells	100
Eduardo Gasca	100
Gavin Hugh	100

References

[1] G. Cassidy, "The effect of background music and background noise on the task performance of introverts and extraverts," July 2007. [Online]. Available: https://www.researchgate.net/publication/238333188_The_effect_of_background_music_and_background_noise_on_the_task_performance_of_introverts_and_extraverts.

[2] Arizton, "Active Noise Cancelling (ANC) Headphones Market - Global Outlook and Forecast 2019-2024," August 2019. [Online]. Available: https://www.reportlinker.com/p05806365/Active-Noise-Cancelling-ANC-Headphones-Market-Global-Outlook-and-Forecast.html?utm_source=PRN.

[3] R. Lichenstein, "Headphone use and pedestrian injury and death in the United States: 2004-2011," January 2012. [Online]. Available: https://www.researchgate.net/publication/221751067_Headphone_use_and_pedestrian_injury_and_death_in_the_United_States_2004-2011?fbclid=IwAR2oCfKs_kMy1ImzxqYsGiVpeyGatuDrj8zeISST59IfFcWOhVf4Kudnm24.

[4] E. H. B. a. J. Voix, "Chapter 11: Hearing Protection Devices," in *The Noise Manual, 6th Edition*, American Industrial Hygiene Association, 2018, p. 54.

[5] D. N. Shubhra Dixit, "LMS Adaptive Filters for Noise Cancellation: A Review," 11 April 2017. [Online]. Available: https://www.researchgate.net/publication/320248881_LMS_Adaptive_Filters_for_Noise_Cancellation_A_Review. [Accessed October 2019].

[6] H. D. P. J. a. J. Y. Jinlin Liu, "Headphone-To-Ear Transfer Function Estimation Using Measured Acoustic Parameters," 3 June 2018.

[7] R. S. a. D. Giuliani, "Deep-neural network approaches for speech recognition with heterogeneous groups of speakers including children," 12 April 2016. [Online]. Available: https://www.cambridge.org/core/journals/natural-language-engineering/article/deepneural-network-approaches-for-speech-recognition-with-heterogeneous-groups-of-speakers-including-children/49315B363FEC168B838B820DB6D6AC9F?fbclid=IwAR2BJ0X6sEw7_QyMosmqOcFDiT.

[8] Alpha Cephai, "CMUSphinx," Alpha Cephai, 23 October 2019. [Online]. Available: https://cmusphinx.github.io/.

[9] G. B. Veton Kepuska, "Comparing Speech Recognition Systems (Microsoft API, Google API and CMU Sphinx)," March 2017. [Online]. Available: https://www.researchgate.net/publication/314938892_Comparing_Speech_Recognition_Systems_Microsoft_API_Google_API_And_CMU_Sphinx.

[10] C. H. W. M. G. M. Thad B. Welch, Real-Time Digital Signal Processing from MATLAB to C with the TMS320C6x DSPs, Boca Raton, FL: CRC Press, 2016.

[11] The United States Social Security Administration, "Top Five Names for Births in 1919-2018," 2019. [Online]. Available: https://www.ssa.gov/oact/babynames/top5names.html.

[12] 1000MostCommonWords, "1000 Most Common English Words," 2014. [Online]. Available: https://1000mostcommonwords.com/1000-most-common-english-words/.

[13] N. v. d. Velde, "Speech Recognition Technology Overview," 8 July 2019. [Online]. Available: https://www.globalme.net/blog/the-present-future-of-speech-recognition.

[14] World Health Organization, "Make Listening Safe," 2015. [Online]. Available: https://www.who.int/pbd/deafness/activities/MLS_Brochure_English_lowres_for_web.pdf.

[15] Google Cloud, "Data-logging," 2019. [Online]. Available: https://cloud.google.com/speech-to-text/docs/data-logging.

[16] H. Gether, "Active noise cancellation: trends, concepts, and technical challenges," 8 October 2013. [Online]. Available: https://www.edn.com/active-noise-cancellation-trends-concepts-and-technical-challenges/. [Accessed January 2020].

Appendix A: Additional Code

```c
////////////////////////////////////////////////////////////////////////
// Filename: ANC_ISR.c
//
// Summary: Interrupt service routine for ANC adaptive filtering
//                simulation.
//
////////////////////////////////////////////////////////////////////////

#include "DSP_Config.h"
#include "AIC3106.h"
#include <stdio.h>

// Data is received as 2 16-bit words (left/right) packed into one
// 32-bit word.  The union allows the data to be accessed as a single
// entity when transferring to and from the serial port, but still be
// able to manipulate the left and right channels independently.

#define LEFT  0
#define RIGHT 1

volatile union {
      Uint32 UINT;
      Int16 Channel[2];
} CodecDataIn, CodecDataOut;

// LMS filter parameters
#define N 80
float beta = 1e-10;        // learning rate start point, chosen empirically

/* add any global variables here */
float x_E_buffer[N];       //buffer for error signal delay samples
float x_ref_buffer[N];       //buffer for test signal delay samples

float x_ref_filtered_buffer;       // buffer for the filtered test signal
delay samples

int demo_num = DEMO;

int data_flag = 0;         //UART data received flag
char data = 'a';            //keyboard data (a = ANC mode) or (p =
passthrough mode)

float w[N];             //buffer weights of adapt filter

interrupt void Codec_ISR()
////////////////////////////////////////////////////////////////////////
// Purpose:   Codec interface interrupt service routine
//
// Input:     None
//
// Returns:   Nothing
//
// Calls:     CheckForOverrun, ReadCodecData, WriteCodecData
//
// Notes:     None
////////////////////////////////////////////////////////////////////////
{
```

```
int i;
data_flag = IsDataReady_UART2(); //polls status of UART receive flag
if(data_flag != 0){
    data = Read_UART2();        // reads UART data (either 'a' or 'p')
}
float result = 0; //initialize the accumulator

if(CheckForOverrun())   // overrun error occurred (i.e. halted DSP)
      return;                 // so serial port is reset to recover

CodecDataIn.UINT = ReadCodecData();       // get input data samples

// send test signal to Left Channel
x_ref_buffer[0] = (float) CodecDataIn.Channel[LEFT];
// send filtered test signal to Right Channel
x_ref_filtered_buffer = (float) CodecDataIn.Channel[RIGHT];

// ANC mode: use adaptive filter to produce anti-noise
if (data == 'a' && demo_num == 0){
    for (i = 0; i< N; i++)
        result += (w[i] * x_ref_buffer[i]);    // y(n) = w(n) * r(n)

    // e(n) = s(n) + y(n)
      x_E_buffer[0] = x_ref_filtered_buffer - result;

    // calculate learning rate normalizing factor
    for (i = 0; i < N; i++)
    {
        beta += (x_ref_buffer[i]*x_ref_buffer[i]);
    }

    beta = 1/beta;

    // updates adaptive filter weights and delays
    for (i = N-1; i> 0; i--)
    {
        w[i] = w[i] + beta * x_ref_buffer[i] * x_E_buffer[0];
        x_ref_buffer[i] = x_ref_buffer[i-1];
    }
    w[0] = alpha * w[0] + beta * x_ref_buffer[0] * x_E_buffer[0];

      result = x_E_buffer[0];       // output resulting error signal
}

// passthrough mode: pass input sample directly to output
else if(data == 'p')
{
    result = x_ref_buffer[0];
}
//Return 16-bit sample to DAC
// outputs e(n) (ANC mode) or r(n) (passthrough mode)
CodecDataOut.Channel[LEFT] = (short) result;

// Copy Right input directly to Right output with no filtering
CodecDataOut.Channel[RIGHT] = (short) result;

WriteCodecData(CodecDataOut.UINT);  // send output data to audio port
```