Creating a Convolution Reverberation Effect from Impulse Responses in Physical Spaces

Digital Audio Systems, DESC9115, 2020
Master of Architectural Science (Audio and Acoustics)
Sydney School of Architecture, Design and Planning, The University of Sydney

ABSTRACT

This paper gives detail to the implementation of convolution of an input signal with an impulse response recorded in a physical space in both the time and frequency domain in order to model a reverberation effect. The Impulse Responses used in this project were recorded in physical spaces using sinusoidal sweeps and recorded into one channel. The Input Signals used were recorded using various instruments into one channel. This paper describes the differences between time domain convolution and FFT convolution with a focus on the uses of FFT convolution in reverberation effects.

1. INTRODUCTION

Convolution reverberation effects have changed the way audio engineers and musicians think about reverberation. According to Zölzer (2011), artificial reverberation was initially theorized by Schroeder in the early 60s and used an extensive series of recursive comb filters, tapped delay lines and all-pass filters. Convolving an input of choice (IP) with an impulse response (IR) taken in a physical room is far more accurate and can be thought of the digital era's version of a reverberation chamber.

Due to issues like background noise, accessibility and legalities, convolution reverberation effects are far more convenient and are theoretically just as accurate as recordings taken directly in a physical space. As an impulse response can be thought of as accurately describing a system, taking IR's of a space could be thought of as an attempt at preservation- saving the sound of these spaces for generations to come. Many interesting sounding spaces are not accessible to those living with mobility disabilities or simply outside of a geographic area, so using convolution reverberation effects can aid in making the world seem smaller and sound better.

1.1 Convolution in the Time Domain vs FFT Convolution

While it is true that a signal can be convolved in both time and frequency domains, it is far more efficient to use the FFT Convolution when working with longer signals (Smith, 1999). For convolution in the time domain, each discrete sample of the input signal $x[n]$ is multiplied with each discrete sample of the system $h[n]$. These values are summed together to create an output $y[n]$. Time domain convolution is only more efficient when the time domain convolution is less complex than the Fast Fourier Transform equivalent - and vice verca. The mathematical equation for convolution in the time domain is shown below in Figure 1.

$$y[i] = \sum_{j=0}^{M-1} h\,[j]x[i-j]$$

Figure 1 –Convolution in the Time Domain where i is the sample in the output system, j is each discrete IR sample running from 0:M – 1, h is the Impulse Response and x is the Input Signal (Smith, 1999). This has to occur for every sample point.

The main principle behind FFT Convolution is that convolution in the time domain gives the same result as multiplication in the frequency domain. As multiplication calculations are far more efficient than scores of convolution calculations, FFT Convolution has become the norm for any kind of convolution with a system or filter larger than around 64 samples (Smith, 1999).

To begin, the frequency response of both the IP and IR is found using DFT via FFT. Using the FFT to compute the DFT is significantly more efficient and faster; if DFT computations are $O(n^2)$ calculations, it can be seen that the FFT implementation is $O(n\,log_{(n)})$ (Heckbert, 1995). When $n = 2^r$ $where\ r\ is\ an\ interger\ of\ some\ value\ of\ interest,$ the FFT exploits symmetries in the DFT matrix and allows for significant reduction of computations by efficiently halving the size of matrixes until they have been reduced to a 2:2 matrix (Heckbert, 1995). If n isn't an integer of 2, the FFT can make use of zero padding until it is (Brunton, 2020).

The FFT algorithm uses overlap, add and multiply butterfly calculations where each butterfly takes the two complex numbers p and q and calculates other two numbers, $p + \alpha q$ and $p - \alpha q$ where α is a complex number (Heckbert, 1995). Subsequently, these complex numbers the real and imaginary parts of the frequency spectrum and the equation used to multiply them can be seen below in Figure 2 (Smith, 1999). More detailed information on the mathematics behind the FFT can be found in Heckbert's 1995 paper *'Fourier Transforms and the FFT Algorithm'* and Bekele's 2016 paper *'Cooley-Tukey FFT Algorithms'.*

$$ReY[f] = ReX\,[f]ReH[f] - ImX[f]ImH[f]$$
$$ImY[f] = ImX[f]ReH[f] + ReX[f]ImH[f]$$

$$Where\ Y\ is\ the\ output, f\ is\ the\ frequency\ domain\ sample\ X\ is\ the\ IP,$$
$$H\ is\ the\ IR, Re\ is\ the\ Real\ component\ and\ Im\ is\ the\ Imaginary\ component.$$

Figure 2 – Frequency Domain Convolution (Smith, 1999)

The real and imaginary elements of the impulse response are multiplied with windows of the input signal using the overlap and add method (Smith, 1999). The signal is decomposed into smaller segments in order for the computations to be less complex. In this case, both the input signal and impulse response are padded with zeros to ensure they are of the same length in order to prevent circular convolution (Heckbert, 1995). The IFFT is then used to produce an output signal and the FFT Convolution process is complete.

2. LAB WORK

To begin the project, I recorded a series of input signals in Pro Tools using the instruments listed in Table 1 below. Further information on the Input Signals can be found in Appendix A.

Input Signals	**Instrument**
IP1	Acoustic Guitar
IP2	Electric Guitar
IP3	Electric Organ
IP4	Trumpet
IP5	Voice
IP6	Snare Drum
IP7	Anechoic Voice

Table 1 – Input Signal Choices

The Impulse Responses chosen for this project can be seen in Table 2 below and were all sourced from the Open Acoustic Impulse Response library from the University of York. Further description of the IR's can be found in the Appendix B.

Impulse Responses	**Description**
IR1	Nuclear Reactor Hall
IR2	Factory Warehouse
IR3	A gulley in Northern Ireland
IR4	Indoor Tennis Court
IR5	A slinky

Table 2 –Impulse Responses Choices

Figure 3 below shows the signal flow for this project.

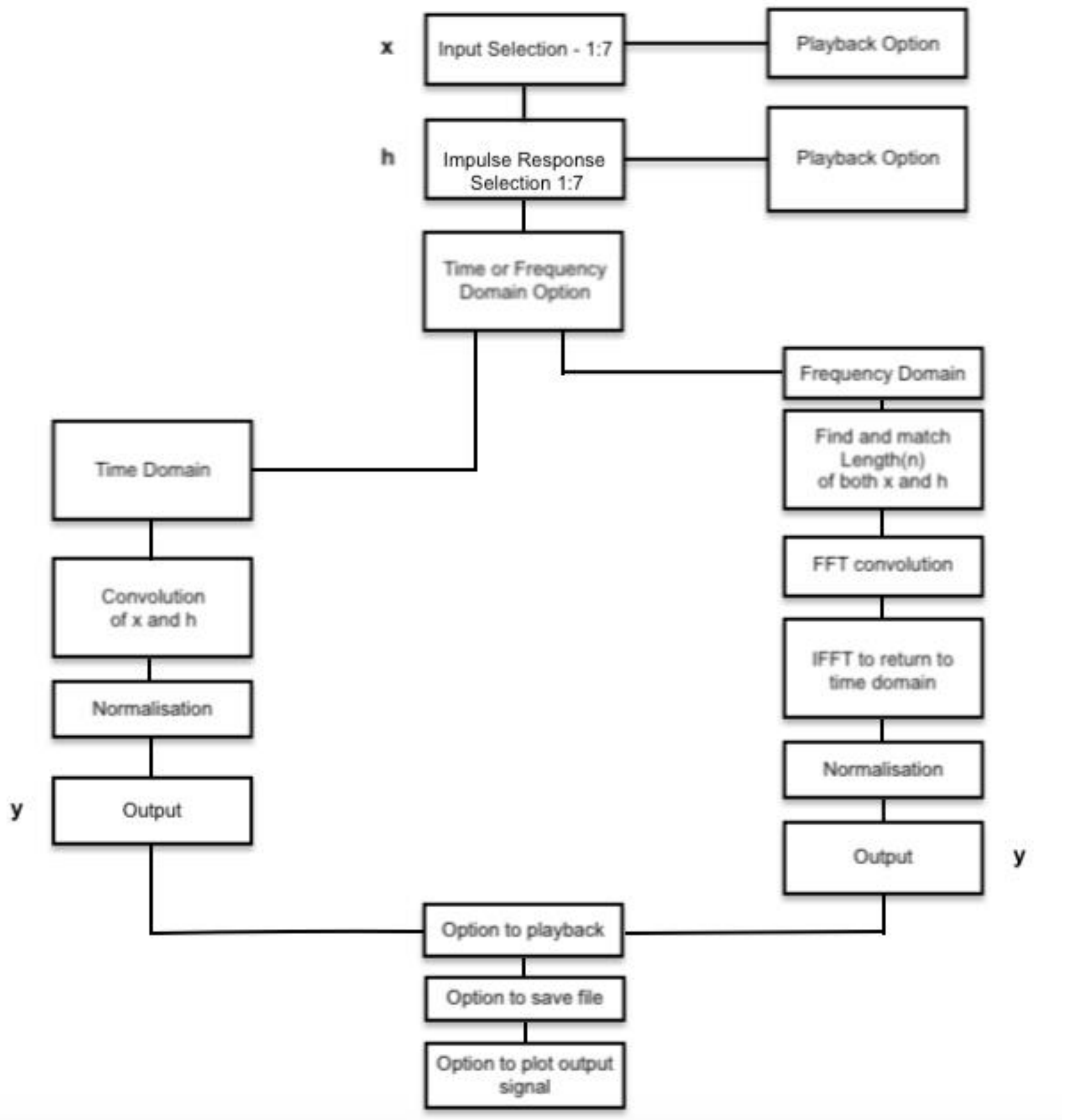
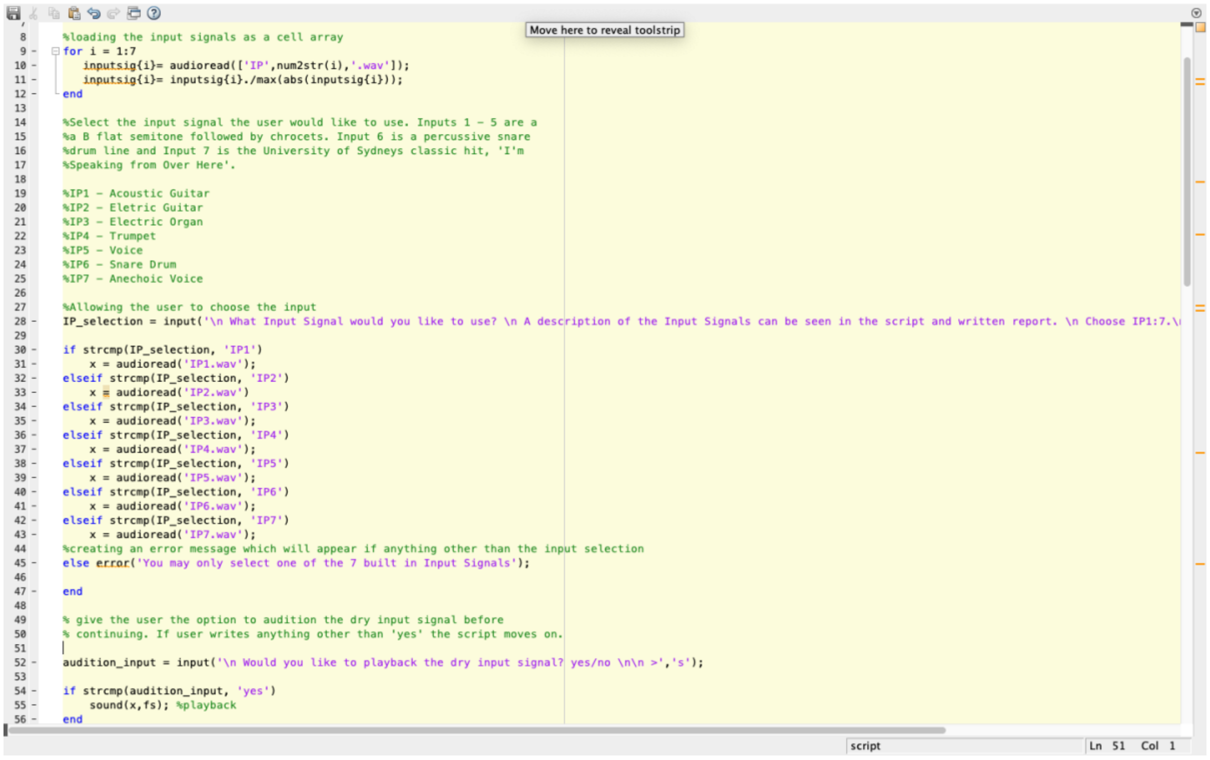


Figure 3 – Signal Flow for the convolution reverberation effect

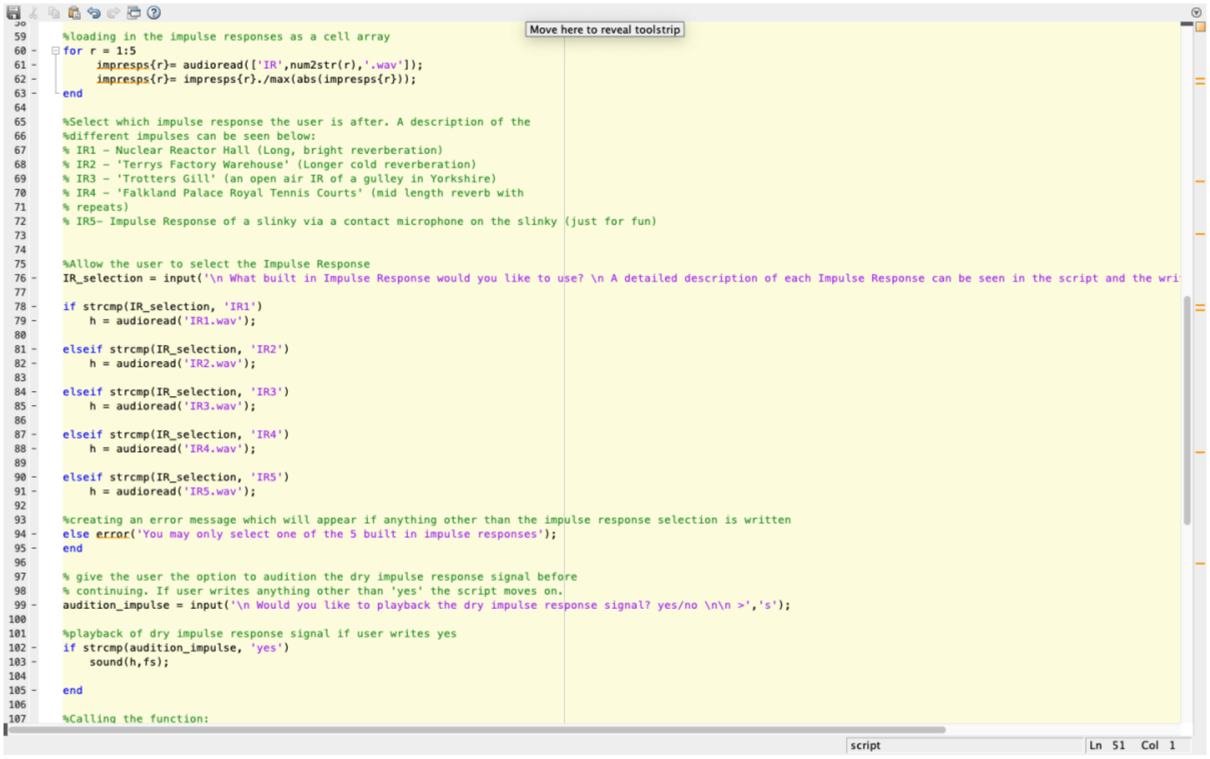
2.1 Preparing the Input Signals and Impulse Responses

To begin the script, the sampling frequency input variable (fs) is loaded as 48kHz. This was chosen for audio quality - both the IP's and IR's chosen were recorded at this sampling frequency. Both the IP's (x) and IR's (h) are loaded into a two separate cell arrays using a series of 'for' loops. The user is given the option for both the IP and IR and also has the option to play back the dry signals using the built in '$input()$' function and '$if/else$' statements. This can be seen below in figures 4 and 5.



```
%loading the input signals as a cell array
for i = 1:7
   inputsig{i}= audioread(['IP',num2str(i),'.wav']);
   inputsig{i}= inputsig{i}./max(abs(inputsig{i}));
end

%Select the input signal the user would like to use. Inputs 1 - 5 are a
%a B flat semitone followed by chrocets. Input 6 is a percussive snare
%drum line and Input 7 is the University of Sydneys classic hit, 'I'm
%Speaking from Over Here'.

%IP1 - Acoustic Guitar
%IP2 - Eletric Guitar
%IP3 - Electric Organ
%IP4 - Trumpet
%IP5 - Voice
%IP6 - Snare Drum
%IP7 - Anechoic Voice

%Allowing the user to choose the input
IP_selection = input('\n What Input Signal would you like to use? \n A description of the Input Signals can be seen in the script and written report. \n Choose IP1:7.\

if strcmp(IP_selection, 'IP1')
    x = audioread('IP1.wav');
elseif strcmp(IP_selection, 'IP2')
    x = audioread('IP2.wav')
elseif strcmp(IP_selection, 'IP3')
    x = audioread('IP3.wav');
elseif strcmp(IP_selection, 'IP4')
    x = audioread('IP4.wav');
elseif strcmp(IP_selection, 'IP5')
    x = audioread('IP5.wav');
elseif strcmp(IP_selection, 'IP6')
    x = audioread('IP6.wav');
elseif strcmp(IP_selection, 'IP7')
    x = audioread('IP7.wav');
%creating an error message which will appear if anything other than the input selection
else error('You may only select one of the 7 built in Input Signals');

end

% give the user the option to audition the dry input signal before
% continuing. If user writes anything other than 'yes' the script moves on.

audition_input = input('\n Would you like to playback the dry input signal? yes/no \n\n >','s');

if strcmp(audition_input, 'yes')
    sound(x,fs); %playback
end
```

Figure 4 - Loading in IP’s as a cell array for input variable x



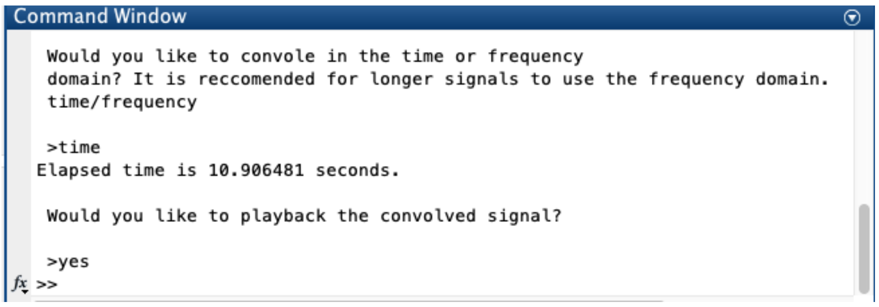
```
%loading in the impulse responses as a cell array
for r = 1:5
     impresps{r}= audioread(['IR',num2str(r),'.wav']);
     impresps{r}= impresps{r}./max(abs(impresps{r}));
end

%Select which impulse response the user is after. A description of the
%different impulses can be seen below:
% IR1 - Nuclear Reactor Hall (Long, bright reverberation)
% IR2 - 'Terrys Factory Warehouse' (Longer cold reverberation)
% IR3 - 'Trotters Gill' (an open air IR of a gulley in Yorkshire)
% IR4 - 'Falkland Palace Royal Tennis Courts' (mid length reverb with
% repeats)
% IR5- Impulse Response of a slinky via a contact microphone on the slinky (just for fun)


%Allow the user to select the Impulse Response
IR_selection = input('\n What built in Impulse Response would you like to use? \n A detailed description of each Impulse Response can be seen in the script and the wri

if strcmp(IR_selection, 'IR1')
    h = audioread('IR1.wav');

elseif strcmp(IR_selection, 'IR2')
    h = audioread('IR2.wav');

elseif strcmp(IR_selection, 'IR3')
    h = audioread('IR3.wav');

elseif strcmp(IR_selection, 'IR4')
    h = audioread('IR4.wav');

elseif strcmp(IR_selection, 'IR5')
    h = audioread('IR5.wav');

%creating an error message which will appear if anything other than the impulse response selection is written
else error('You may only select one of the 5 built in impulse responses');
end

% give the user the option to audition the dry impulse response signal before
% continuing. If user writes anything other than 'yes' the script moves on.
audition_impulse = input('\n Would you like to playback the dry impulse response signal? yes/no \n\n >','s');

%playback of dry impulse response signal if user writes yes
if strcmp(audition_impulse, 'yes')
    sound(h,fs);

end

%Calling the function:
```

Figure 5 - Loading in IR’s as a cell array for input variable h

2.2 Convolution in the Time and Frequency domain

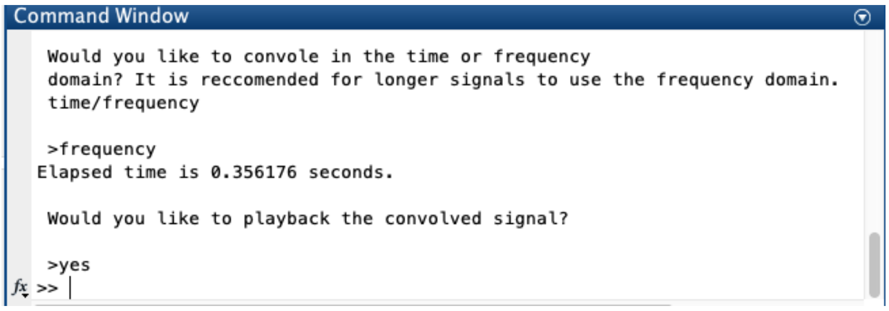
After loading the input variables, the script calls the function $conv\_verb(x, h)$. The basis of this code was taken from Nathanial Sjarif's 'FFT Convolution' function written in 2012 and found in the Universities e-repository and heavily edited.

Via an 'if' statement, the user is given a choice of convolving in the time domain or using FFT Convolution. Both options have a stopwatch timer imbedded within the function, so out of interest the user can see how much faster FFT Convolution is. This can be seen below in figure 6 and figure 7.



```
Command Window
  Would you like to convole in the time or frequency
  domain? It is reccomended for longer signals to use the frequency domain.
  time/frequency

  >time
Elapsed time is 10.906481 seconds.

  Would you like to playback the convolved signal?

  >yes
fx >>
```

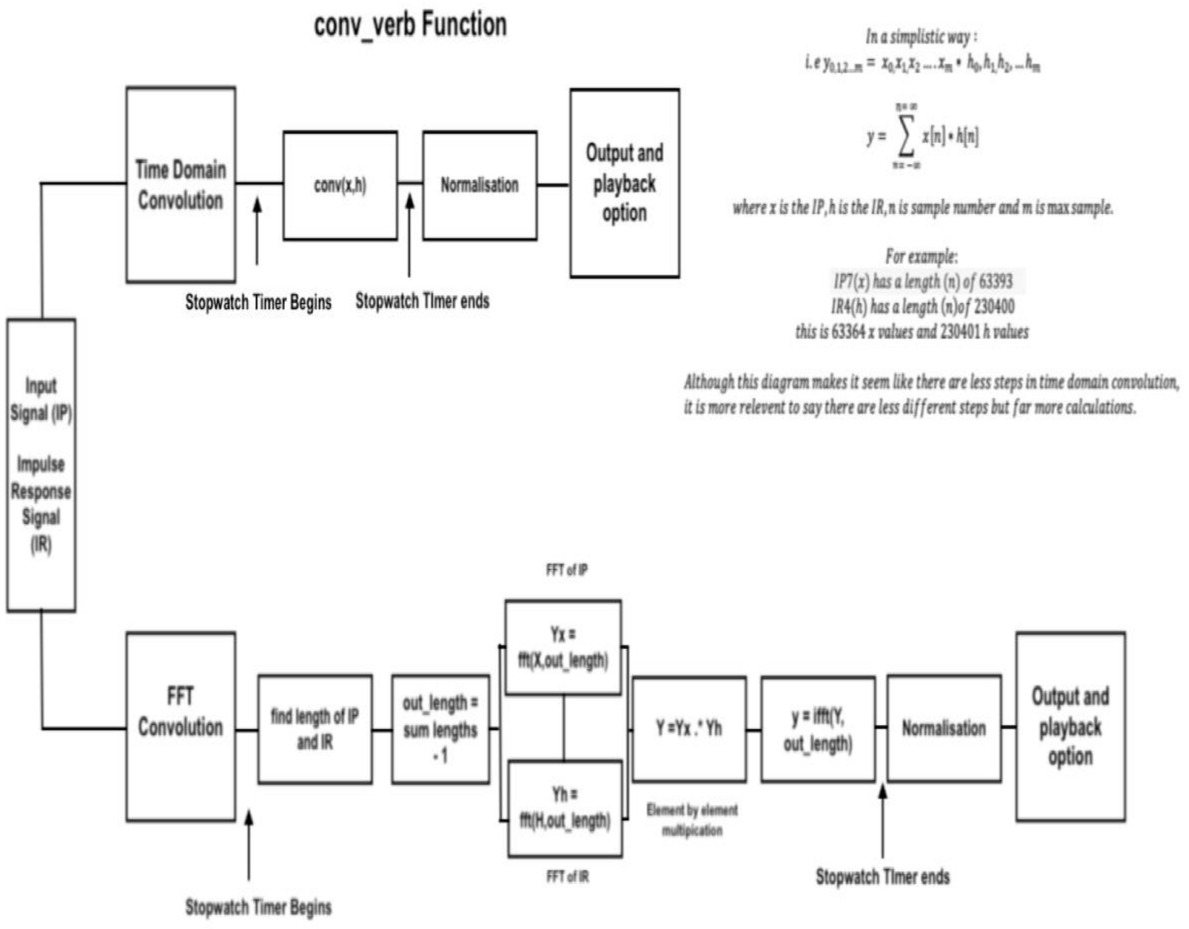
Figure 6 – Stopwatch counter for time domain processing



```
Command Window
  Would you like to convole in the time or frequency
  domain? It is reccomended for longer signals to use the frequency domain.
  time/frequency

  >frequency
Elapsed time is 0.356176 seconds.

  Would you like to playback the convolved signal?

  >yes
fx >> |
```

Figure 7- Stopwatch counter for frequency domain processing

If the time domain convolution option is chosen, the function convolves the signal in a straightforward time domain convolution operation with the built-in function $conv(u, v)$. While significantly less efficient, I decided to leave this in for future versions of this project- perhaps the user would be interested in convolving an IP with an IIR filter, or another time domain-based process. A diagram outlining both processes can be seen below in figure 8.



In a simplistic way :

$$\text{i.e } y_{0,1,2...m} = x_0 x_1 x_2 \ldots x_m \cdot h_0, h_1 h_2, \ldots h_m$$

$$y = \sum_{n=-\infty}^{n=\infty} x[n] \cdot h[n]$$

where x is the IP, h is the IR, n is sample number and m is max sample.

For example:
IP7(x) has a length (n) of 63393
IR4(h) has a length (n)of 230400
this is 63364 x values and 230401 h values

Although this diagram makes it seem like there are less steps in time domain convolution, it is more relevent to say there are less different steps but far more calculations.

Figure 8 - Diagram of the conv_func function

If the frequency domain option is selected, the function begins processing the audio using the inbuilt $fft(X,n)$ and $ifft(Y,n)$ functions respectively, of which the equations can be seen below in figure 9. Prior to the transforms, the lengths of both the IP and IR are found and summed together minus 1 sample (Smith, 1999). It is this value (out_length) that is ‘n’ in the matlab code. When the DFT is found via the FFT, both the IP and the IR and multiplied in an element by element multiplication using $.*$. The IFFT of the output is then found and the signal is normalised to the maximum value of 1.

$$FFT\ :\ Y(k) = \sum_{j=1}^{n} X(j)\ W_{n^{(j-1)(k-1)}}$$

$$where\ W_n = e^{(-2\pi i)/n}$$

$$IFFT\ :\ X[j] = \frac{1}{n}\sum_{k=1}^{n} Y\ (k) W_{n^{-(j-1)(k-1)}}$$

$$where\ W_n = e^{(-2\pi i)/n}$$

$Where\ j\ is\ each\ discrete\ sample\ in\ the\ time\ domain, k\ is\ each\ discrete\ sample\ in\ the\ frequency\ domain\ from\ \ domain\ sample, X\ is\ the\ frequency\ domain\ signal\ and\ n\ the\ transform\ length.$

Figure 9 – FFT and IFFT Algorithms (MathWorks, 2020)

The user is then asked whether they'd like to save the project through another '*input()*' function, and may select the name of the file. The data is formatted into a string by the use of '$sprintf()$' and saved as a wave file through the use of the built in function '$audiowrite(filename, y, fs)$'.

Lastly, the user then has the option to plot the resulting signal in both a spectrograph and an average magnitude spectrum plot for further analysis using the spectrogram_func written by Ella Manor (2020) with the option to choose window size and overlap through a last series of the '$input()$' function .

3 DISCUSSION & CONCLUSION

This digital effect is successful in recreating spaces faithfully using both time domain convolution and FFT convolution, This effect would be effective if used in parallel to the dry audio signal, and this can be heard in the 'Parallel_Signal.wav' file supplied with this paper. An interesting point to note is that the *conv_verb* function written for this assignment could be used to create a whole manner of interesting sounds by swapping out IR's of rooms for IR's of analogue desks, microphones, kitchen sinks or children's toys, as seen in IR5 in this project.

With further experimentation I would like to give the user the option to run both signals through a bandpass limiter in order to 'clean' up the signal for further use. While I did research into adding a 'wet' and 'dry' option, this was very difficult to implement to any degree of usefulness through the user interface chosen. Further implementation of this project will involve building a GUI or standalone application for a more user-friendly digital effect.

Appendix A

Input Signals:

In order to keep the input signals consistent, I played a Bb semibreve and four crochet's in all pitched signals. This was chosen to exemplify what both a sustained and short excitation would like with the effect. Bb was chosen as is the natural key for the trumpet. The snare drum signal was added out of interest for even shorter excitations, and the Anechoic Voice signal chosen for its lack of existing reverberant sound. IP's 1:6 were recorded into Pro-Tools via an AKGd190 and played by me. The Anechoic Voice signal was recorded in the Anechoic Chamber in the Acoustics Lab of the University of Sydney and is the property of the University of Sydney.

Appendix B

Impulse Responses:

Many different impulse responses were auditioned for this paper, but many of them sounded far too similar to warrant them being included (after a while a large church sounds like a large church sounds like a large church). All IR's used in this project were sourced from the Open Acoustic Impulse Response (Open AIR) Library maintained by Damian Murphy and Joe Rees-Jones at the University of York.

IR1 -Nuclear Reactor Hall was recorded by Damian T. Murphy in 2006. This space is a abandoned Nuclear Reactor Hall measuring 3500m^3 sitting 25m under the KTH Royal Institute of Technology, Sweden. Further information can be found at: https://openairlib.net/?page_id=626

IR2 – 'Terrys Factory Warehouse' was recorded by Damian T. Murphy in 2006. The site is an unused warehouse measuring 4500 m^3 that was closed the previous year. Further information can be found at: https://openairlib.net/?page_id=735

IR3 – 'Trotters Gill' was recorded by Andrew Chadwick and Simon Shelley in 2013 around a limestone George with steep cliffs either side. The direct sound source file was used in this project. This was chosen out of interest for an open air reverberation. More information can be found at: https://openairlib.net/?page_id=745

IR4 – 'Falkland Palace Royal Tennis Courts' were recorded by Damian T. Murphy in 2009 in a tennis court with no roof at Falkland Palace in Fife. This was chosen due to the interesting delays in the end of the tail. More information can be found at : https://openairlib.net/?page_id=476

IR5 – Slinky has no documented author but is an impulse response of a toy slinky recorded by attaching a contact microphone to the toy and plucking it. This was chosen purely out of interest, and to give evidence to convolution being used for effects other than reverberation. More information can be found at: https://openairlib.net/?page_id=652

Some very interesting Neolithic tomb IR's were also taken but these were only recorded in b-format and thus out of the scope of this project.

3 REFERENCES

2020. fft (Fast Fourier Transform). Retrieived from https://au.mathworks.com/help/matlab/ref/fft.html#d120e347344

Bekele, A. (2016). *Cooley-Tukey FFT Algorithms.* Retrieved from http://people.scs.carleton.ca/~maheshwa/courses/5703COMP/16Fall/FFT_Report.pdf

Brunton, S. (2020, April, 4). *The Fast Fourier Transform Algorithm* [Video]. YouTube. URL https://www.youtube.com/watch?v=toj_IoCQE-4

Chadwick, A. Shelley, S. (2013). *Trotters Gill* [Impulse Response]. Retrieved from https://openairlib.net/?page_id=745

Heckbert, P. (1995). *Fourier Transforms and the Fast Fourier (FFT Algorithm).* Retrieved from http://www.cs.cmu.edu/afs/andrew/scs/cs/15-463/2001/pub/www/notes/fourier/fourier.pdf

Manor, E. (2020). 'myspectrogram.m' [Matlab Code] [Class Handout]. Retrieved from canvas.sydney.edu.a

Murphy, D, T. (2006). *Nuclear Reactor Hall* [Impulse Response]. Retrieved from https://openairlib.net/?page_id=626

Murphy, D, T. (2006). *Terrys Factory Warehouse* [Impulse Response]. Retrieved from https://openairlib.net/?page_id=735

Murphy, D, T. (2009). *Falklands Palace Royal Tennis Courts* [Impulse Response]. Retrieved from https://openairlib.net/?page_id=476

Open Acoustic Impulse Response Library. (N.D.). *Slinky* [Impulse Response]. Retrieved from https://openairlib.net/?page_id=652

Sjarif, N. (2012). FFT Convolution Function [Matlab Code]. Retrieved from http://hdl.handle.net/2123/8215

Smith, S, W. (1999). *The Scientist and Engineer's Guide to DSP (Second Edition).* California Technical Publishing.

Zölzer, U. (Ed.). (2011). DAFX: digital audio effects. John Wiley & Sons.