Queensland University of Technology, EGB242

Assignment 2

MARS242 — Orbiting The Red Planet

Group 14

Charlie Turner n10752846 Joel Kemp n10746862
Viet Truong Minh n10440151

June 3, 2022

Introduction

After the chief at Brisbane-based Australian Space Agency (BASA) has reviewed previous tasks, they have decided to assign the team 'live' supervision of the MARS-242 mission.

Communication between the ground and the mission must be clear and uninterrupted to ensure the mission is safe and smooth. There are three important tasks to be completed:

- Identifying an appropriate landing site for the crew
- Ensuring the safety of the Martian habitats for the current and future crews

To achieve these objectives, the team must be able to properly interpret and analyse data streams from a rover scouting the surface prior to landing. The landing site must be identified despite periodic and bandlimited noise disrupting signal streams. Once a landing site is determined, the acoustic properties of the habitat's rooms must be analysed, removing interference and noise arising from the communication channel and method.

1 Mars Rover

With the crew landing soon, a rover has been gathering data regarding possible landing sites. Until a malfunction on the rover, data has been communicated directly with Earth, where it can be analysed and the results sent directly to astronauts on board Mars-242. The rover's data is vital for the survival of the mission, so a solution has been found to send the raw data directly to astronauts, but there are complications in this process.

1.1 Communication Channels

The spaceship has a constant magnitude function in the frequency domain between $-f_{bw}$ and f_{bw}. $s_{rov}(t) = 8000 \times \text{sinc}^2(4000t)\cos(2\pi \times 8000t)$

1.2 Rover Transmission Function

$S(f)\times$

1.3 Filtering Information

1.4 Signal Recovery

2 Choosing a Landing Site

The surface-based rover has collected and transmitted images of possible landing sites. Unfortunately, the transmission encountered both periodic and bandlimited noise, so the signals must be denoised to observe the intended images. One of the noisy images is visualised in Fig. 2.1. This is achieved in MATLAB using the `reshape` function, reshaping the incoming 1D data stream into a matrix, knowing that the images are all of the same size (640×480).

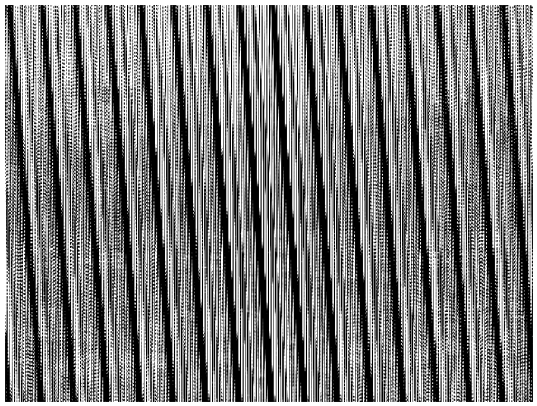


Figure 2.1: Noisy image of possible landing site

2.1 Modelling Periodic Noise

To understand the characteristics of relevant noise, the signal is plotted in both the time and frequency domain as in Fig. 2.2, understanding the importance of the sampling rate for the incoming signal:

```
1  Fs = 1000;                                    % Pixel Sampling Rate [Hz]
2  T = pixels/Fs;                                % Time to recieve an image
3  t = linspace(0, T, length(sig)+1);            % Time vector for full image
4  f = linspace(-Fs/2, Fs/2, length(sig)+1);     % Frequency Vector
```

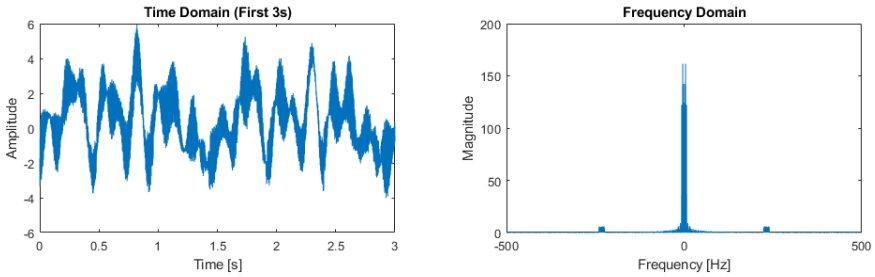


Figure 2.2: Time and Frequency Domain of Incoming Signal

While the full signal for an image is over 300 seconds long, plotting only the first three seconds shows that there is a significant periodic noise present in the time domain. Engineers at mission control have identified a set of possible periods for the noise, and graphically we can determine that it appears to be approximately 1.477 seconds long, confirming the engineers approximations. Using MATLAB and the provided `estimateNoise` function, a vector representing one period of this noise was computed:

```
T = candidateT(1);                                          % period of noise
periodInSamples = T * Fs;                                   % samples in noise period
Noisesig = estimateNoise(sig(1, :), periodInSamples);       % estimated noise function
```

Comparing the estimated noise to the received signal in Fig. 2.3, we can see that the overall shape of the signals is similar.

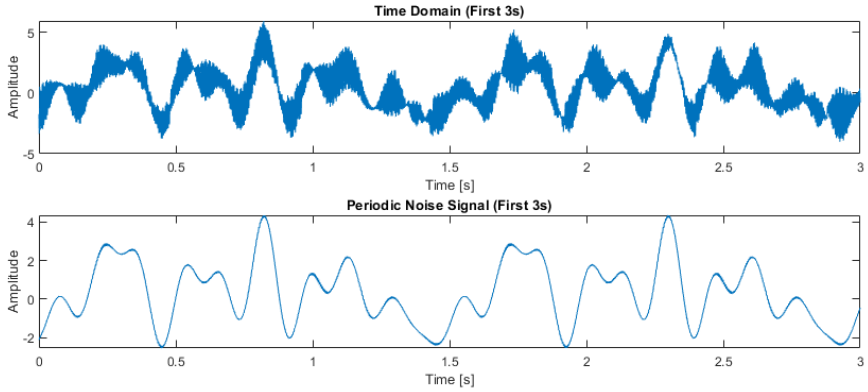


Figure 2.3: Comparison of Estimated Noise and Recieved Signal

With the correct discrete representation of periodic noise determined, we can model `Noisesig` with a complex fourier series as follows:

$$x(t) = \sum_{n=-\infty}^{\infty} c_n e^{j2\pi n f_0 t}$$
$$\text{with } c_n = \frac{1}{T}\int_{t_0}^{t_0+T} x(t)e^{-j2\pi n f_0 t}dt$$

In MATLAB, integrals are substituted for Riemann sums on the discrete vector, computing c_n for $-6 \leq n \leq 6$ taking into account an accurate time vector and sampling rate:

```
t1= t(t<T); t1(end) = [];                                   % Noise Time vector
ts = t1(2)-t1(1);                                           % Sampling interval [s]
f0 = 1/T;                                                   % Fundamental Frequency
N = 6;                                                      % Number of harmonics
n = (-N:N).';                                               % Vector of harmonics
cn = f0 * Noisesig * exp(-1j*2*pi*f0*n*t1).' * ts;          % Fourier coefficients
c0 = cn(N+1);                                               % DC coefficient
```

The DC coefficient is computed as $c_0 = +0.3571$, and the complex coefficients for the first 6 harmonics as follows:

$$\begin{array}{lll} c_{-6} : -0.2208 + 0.4813i & c_{-5} : +0.2479 - 0.2237i & c_{-4} : +0.0260 - 0.1276i \\ c_{-3} : -0.4525 + 0.1048i & c_{-2} : -0.2916 + 0.2814i & c_{-1} : -0.3702 + 0.1522i \\ c_{+1} : -0.3702 - 0.1522i & c_{+2} : -0.2916 - 0.2814i & c_{+3} : -0.4525 - 0.1048i \\ c_{+4} : +0.0260 + 0.1276i & c_{+5} : +0.2479 + 0.2237i & c_{+6} : +0.2208 + 0.4813i \end{array}$$

BASA has confirmed that there should be no DC component to the periodic noise. Because c_0 represents the DC component of the Fourier series, this term can simply be equated to 0 in order to remove this bias:

```
90  c0 = 0;                                          % DC coefficient
91  cn(N+1) = c0;                                    % Corresponding coefficient in cn vector
```

With the correct Fourier coefficients determined, the signal can be reconstructed for the total period of the image signal, as per the following:

```
93  Noisesig_fs = cn * exp(1j*2*pi*f0*n*t);          % Fourier series for total t
```

To test the accuracy of the model, this Fourier approximation was plotted against the estimated noise profile for one period, as seen in Fig. 2.4:

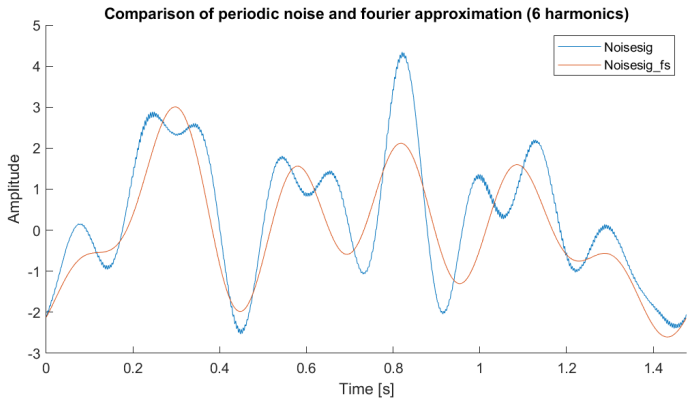


Figure 2.4: Estimated noise and Fourier series approximation

Visually, the Fourier approximation using 6 harmonics does not provide a full representation of the signal, as it is unable to model the more detailed noise at the peaks and such. The overall shape of the noise is represented, but it is expected that the denoising may not be sufficient for recoving detail within the image.

2.2 Removing Periodic Noise

The above period noise approximation can be tested by subtracting the periodic noise from the incoming image signal:

```
130  im1(1,:) = sig(1,:) - Noisesig_fs;               % First image with periodic noise removed
```

As with the original signal, this denoised signal was then reshaped and shown in Fig. 2.5, along with the frequency domain representation:

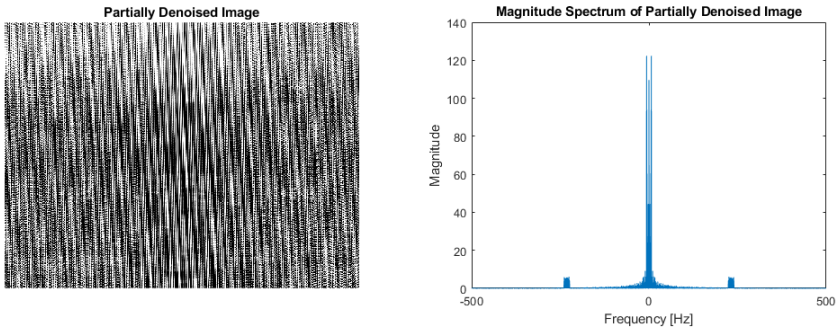


Figure 2.5: Image and Magnitude Spectrum with periodic noise removed

As expected, the image is not noticeably clearer than the original, with strong lines still obscuring the intended picture. The magnitude spectrum confirms this noise, with large frequency spikes present at specific low frequencies. This indicates that there is still considerable periodic noise present. To improve the noise approximation, a more suitable number of harmonics would be 10, as it captures the details much more accurately. It also requires very little extra computational power, taking only about 0.03 seconds longer to compute the Fourier approximation. The effect of 10 harmonics is shown in Fig. 2.6:

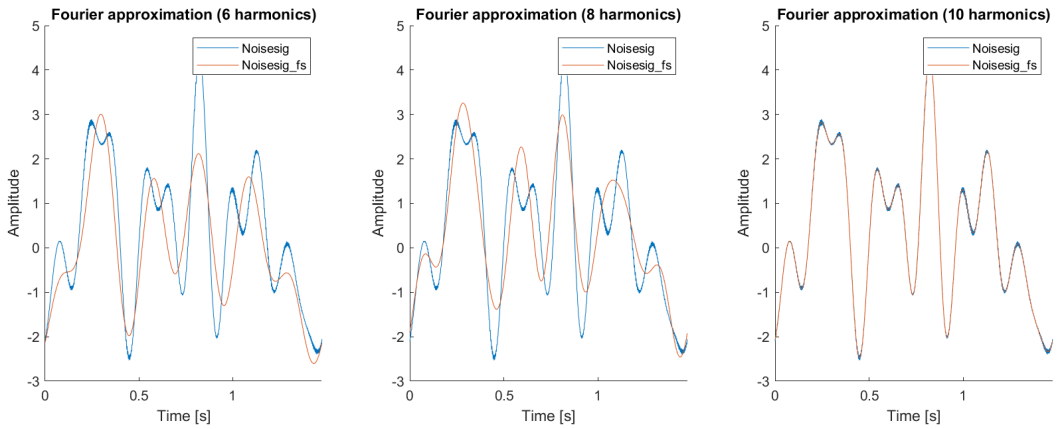


Figure 2.6: Comparison of fourier series (6, 8, and 10 harmonics)

Now with 10 harmonics to better approximate the periodic noise, we can once again denoise the image, shown in Fig. 2.7:

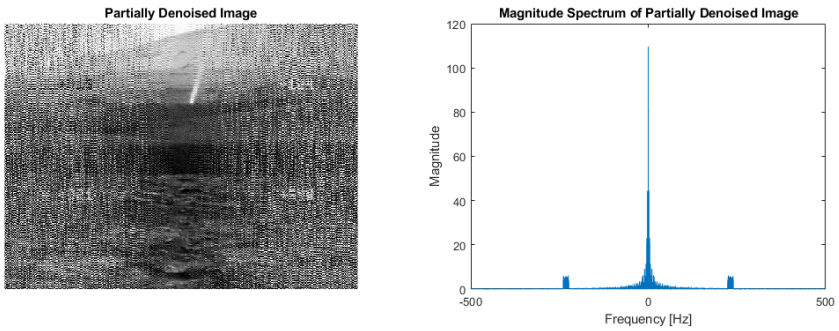


Figure 2.7: Improved Denoised Image and Magnitude Spectrum

The strong predictable lines corrupting the image are now much less pronounced, and the outline of what resembles a mountain and rocky terrain can be distinguished. However, the image is still much too noisy to discern the specific location.

2.3 Removing Bandlimited Noise

The frequency band for bandlimited noise was determined visually as in Fig. 2.8, with a lower limit of ±223Hz, and upper limit of ±240Hz. A band stop filter was created to remove the noise. This is achieved in MATLAB in the frequency domain, which is much more easy to visualise. The filter was constructed as a vector corresponding to the full frequency vector, multiplying frequency components within the band by zero, while leaving other data untouched. Finally, the inverse Fourier transform must be computed to convert the de-noised signal back to the time domain for viewing:

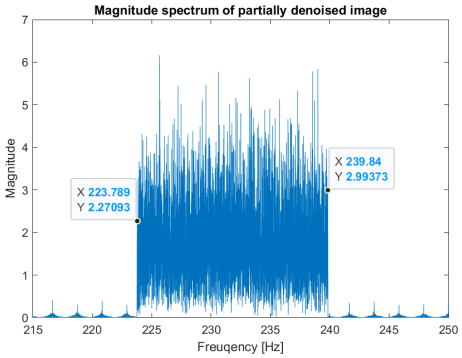


Figure 2.8: Magnitude spectrum of bandlimited noise

```
150  B_low = 223;                                          % Lower bandwidth frequency
151  B_high = 240;                                         % Upper bandwidth frequency
152  filter = ones(size(f));                               % Initiallise with ones
153  filter((abs(f) > B_low) & (abs(f) < B_high)) = 0;     % Assign 0 to elements in band
154  filtered = fftshift(IM1(1,:)) .* filter;              % Filter in frequency domain
155  im2 = zeros(size(sig));                               % Initialise for performance
156  im2(1,:) = ifft(ifftshift(filtered));                 % Filtered image signal in time domain
```

Showing this final image in Fig. 2.9, we can see that the picture has been recovered, including the navigational numbers which were not visible after removing only the periodic noise:

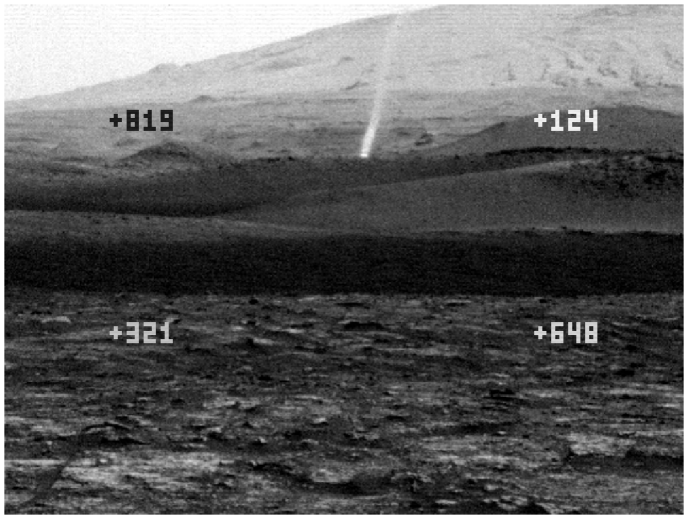


Figure 2.9: Recovered image from the rover

2.4 Choosing A Site

The removal of periodic noise was done as above for all four images, however removing bandlimited noise required reconstructing the band-stop filter for each image. In MATLAB, the frequency bands were determined visually for each image in the frequency domain, and then a for loop was used to filter the corresponding images:

```
% Visually get bands of noise
bands = {[223, 240], [232, 246], [234, 251], [253, 268]};
% Remove bandlimited noise from each image
filter = ones(size(f));                                    % Initialise for performance
im2 = zeros(size(im1));                                    % Initialise for performance
for k = 1:length(bands)
    filter(1:end) = 1;                                     % Reallocate ones
    filter(abs(f) > bands{k}(1) & abs(f) < bands{k}(2)) = 0;% Construct filter
    filteredIm = fftshift(IM1(k, :)) .* filter;            % Apply in frequency domain
    im2(k, :) = ifft(ifftshift(filteredIm));               % Store in time domain
end
```

The result, as seen in Fig. 2.10, is four clear images complete with navigational numbers listed in Table 2.1:

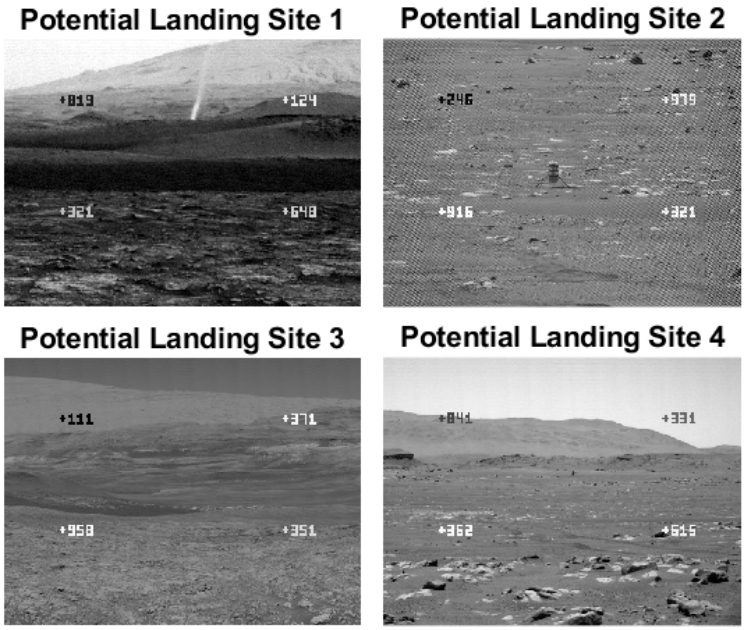


Figure 2.10: All four sites recovered

Table 2.1: Navigational Numbers For Each Site

Site	Top Left	Top Right	Bottom Left	Bottom Right
Site 1	819	124	321	648
Site 2	246	979	916	321
Site 3	111	371	958	351
Site 4	841	331	362	615

It appears as though site 4 may be the ideal landing site, as it is the only image showing a large landscape that is almost entirely flat except for the far away mountains. Site 2 could possibly be a good site, however there is not a large field of view in the image, leaving uncertainty about the surrounding terrain. Sites 1 and 3 have large mountains and valleys, and are therefore unsuitable for ensuring a safe landing.

3 Habitat Impulse Analysis

With a landing site chosen, the base of operations has been set up. In order to ensure safety of occupants, acoustic properties of rooms in the habitat must be analysed. For this analyses, the team has recorded the impulse responses of different rooms, and sent them back to the team at BASA headquarters. The signal is in the form of a Frequency Division Multiplexed (FDM) data stream, with all impulse responses and text describing locations within the same 0.5s signal. In the case of the habitat's acoustic responses, all intended messages are multiplexed to occupy 8kHz of bandwidth at differing sub-carrier frequencies. To recover the messages, the command centre's hardware system analyses the incoming spectrum, demultiplexes the signals, applies analogue filtering and then the signals are sampled by an analogue to digital converter.

3.1 Spectrum Analyser

The Fourier transform of the signal is used to plot the magnitude spectrum in MATLAB. This is done by first determining the time and frequency vectors for the signal, knowing that incoming signal is sampled at `fs=576`kHz:

```
Ts = 1/fs;                                                  % Sampling Period [s]
t = 0:Ts:0.5; t(end) = [];                                  % Time vector
f = linspace(-fs/2, fs/2, length(t)+1); f(end) = [];        % Frequency Vector
MUXSIG = fft(muxSignal);                                    % Fourier Transform
```

Plotting both the time and frequency domains in Fig. 3.1, the signal is clearly composed of 6 multiplexed signals, as shown by the six clear peaks in the frequency domain, mirrored in both the positive and negative frequencies.

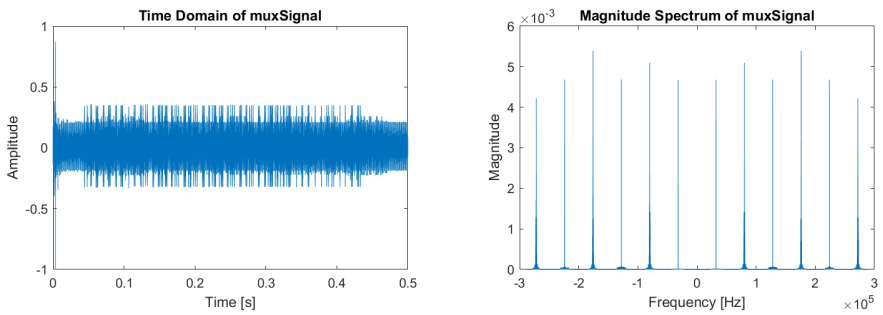


Figure 3.1: Recieved Signal in Time and Frequency Domains

3.2 Demultiplexing

The `findpeaks` MATLAB function was used to find the exact sub-carrier frequencies efficiently and accurately. The magnitude and phase of those frequency shifts was then determined, ensuring that demultiplexing can accurately recover the signals despite any phase shift or magnitude distorsion.

```
MUXSIGshift = fftshift(MUXSIG);                                 % Shift to correspond to f
% Find 6 highest peaks (+ve frequencies)
[~, fshift] = findpeaks(abs(MUXSIGshift(f>0)), f(f>0), 'NPeaks',6, 'SortStr','descend');
fshift = sort(fshift);                                          % Sort in asc order
fshift_indices = find(ismember(f, fshift));                     % Find indices within f
Mag = abs(MUXSIGshift(fshift_indices));                         % Corresponding Magnitudes
Phase = angle(MUXSIGshift(fshift_indices));                     % Corresponding Phase [rads]
```

The 6 signal streams were then recovered using a function provided by the team at BASA, `FDMDemux`. This function takes the signal, the frequency shifts, and the corresponding magnitudes and phases, and returns a matrix containing each individual demultiplexed stream:

```
xdm = FDMDemux(muxSignal, t, Mag, fshift, Phase);          % Demultiplex using known values
XDM = fft(xdm, length(xdm), 2);                            % Row-wise FT
```

Plotting the demultiplexed signals in Fig. 3.2, we can see that there are 6 distinct signals, however the frequency domains clearly show leftover noise from the demultiplexing process. Data within each stream occupies 8kHz bandwidth, so the high-frequency noise will need to be filtered to fully recover each signal. Since the streams have been demultiplexed, a low-pass filter will be ideal for this process.

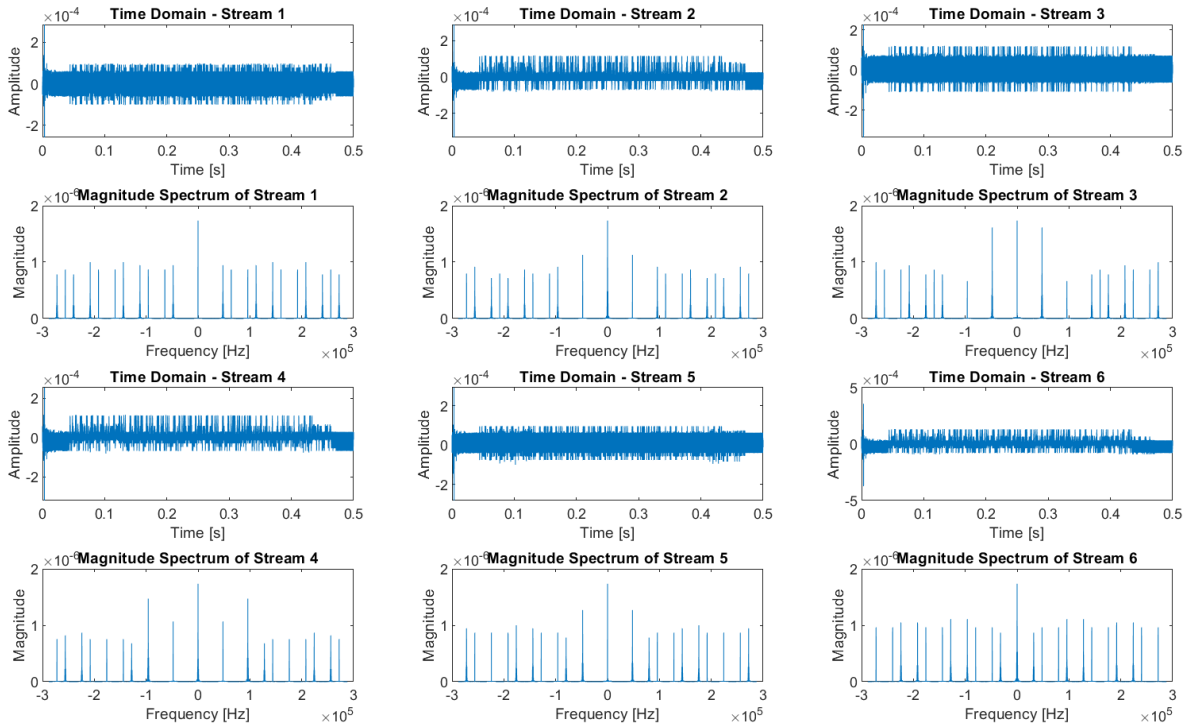


Figure 3.2: Demultiplexed Data Streams in Time and Frequency Domains

3.3 Filtering

BASA has provided four candidate filter circuits and their corresponding transfer functions. The four transfer functions, as shown in MATLAB, are depicted in Fig. 3.3:

$$\text{sys(1)} = \frac{6124595203.1972}{(s + 78259.7941)(s + 78259.7919)}$$

$$\text{sys(2)} = \frac{200119.452s}{(s + 202004.4076)(s + 200119.452)}$$

$$\text{sys(3)} = \frac{s^2}{(s + 13559.707 + j0.00022409)(s + 13559.707 - j0.00022409)}$$

$$\text{sys(4)} = \frac{804247.7193s}{(s - 100530.9649 - j174124.739)(s - 100530.9649 + j174124.739)}$$

Figure 3.3: Transfer Functions of Candidate Circuits

Knowing the properties of Laplace transforms, the poles of each system can indicate important information regarding stability and impulse response. The poles are represented in the denominator of each transfer function, so each was analysed to determine possible suitability.

sys(1): Real, negative poles. Indicates a stable system with exponential decaying impulse response.

sys(2): Real, negative poles. Indicates a stable system. Magnitude of poles greater than system 1, so will likely have a quicker impulse decay comparatively.

sys(3): Negative complex conjugate poles indicate sinusoidal components within the impulse response. Likely a stable system but oscillation may not be ideal.

sys(4): Complex positive poles indicate an unstable system. Not fit for our purposes.

The systems were also visualised in MATLAB, analysing the poles and zeros, impulse response (all except unstable system 4), and simulated frequency response in both a bode plot and our frequencies of interest (see Fig. 3.4).

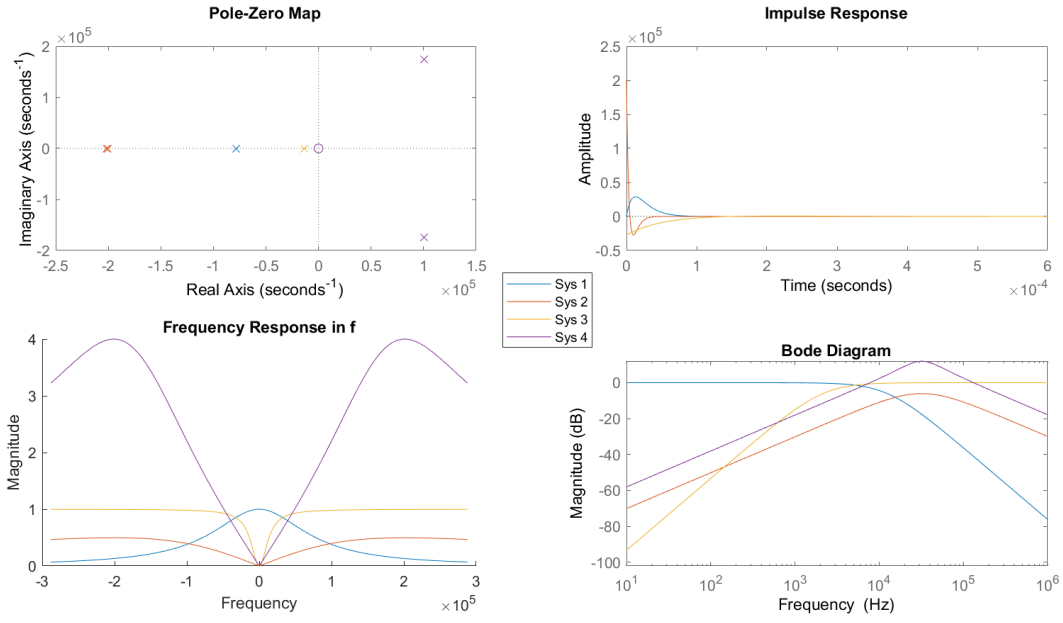


Figure 3.4: System Analysis of Candidate Circuits

System 1 was chosen as the best candidate, as it has characteristics of a low pass filter in the bode diagram. The impulse response rises and decays exponentially without in a short period of time, without oscillatory behaviour. Looking at the frequency response of the system, we can see that the the bode diagram indicates less than 1.5dB attenuation throughout the 0-8kHz band, before rolling off at 40dB per decade. This is the implication of using real analogue filters; the roll off cannot be infinitely steep, so some amount of noise will remain outside the desired band.

To apply the filter in MATLAB, the frequency response of system 1 was multiplied by each stream in the frequency domain. To get this accurate frequency response vector, first the impulse response is computed for `t`, which is the inverse laplace tranform of the systems transfer function in the s-domain. The frequency response is then the fourier transform of this impulse response. The frequency response could also be computed directly by substituting $j\omega$ for s in the transfer function.

```
syms s;                                                    % Define symbolic s
[Num,Den] = tfdata(sys(1),'v');                            % Extract numerator and denominator
sys1_TF = poly2sym(Num, s)/poly2sym(Den, s);               % Get symbolic transfer function
sys1_IR = matlabFunction(ilaplace(sys1_TF));               % Usable function for impulse response
impresp = sys1_IR(t);                                      % Discrete impulse response vector
freqRes = fft(impresp);                                    % Discrete Frequency response
MSG = zeros(size(XDM));                                    % Inilialise size For Performance
for k = 1:size(XDM, 1)
    MSG(k, :) = freqRes .* XDM(k, :);                      % Apply the Transfer Function
end
```

The signals were then converted back to the time domain using an inverse fourier transform. DC offset introduced by the filtering process was removed by adjusting the mean of each signal to 0. Finally, streams were scaled to occupy the amplitude range of $[-1, 1]$. Fig. 3.5 shows the resultant time and frequency domains of each stream after filtering, scaling, and DC removal.

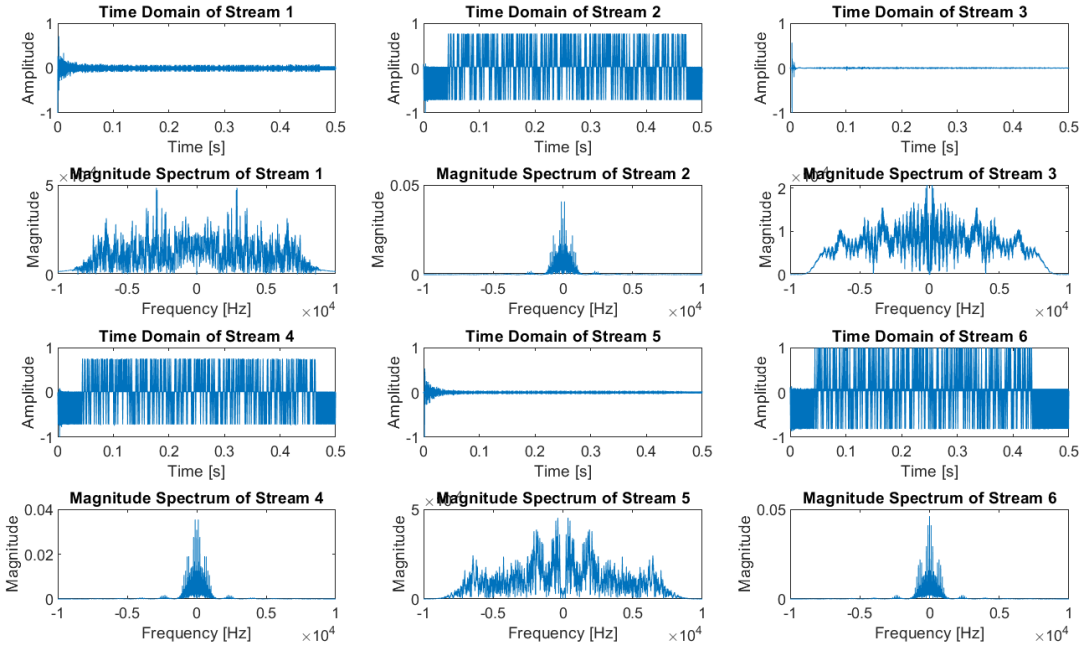


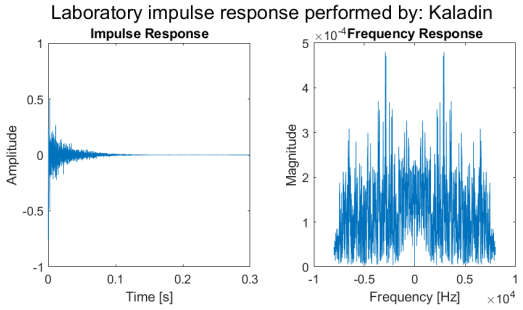
Figure 3.5: Filtered Data Streams in Time and Frequency Domains

3.4 Analogue to Digital Conversion

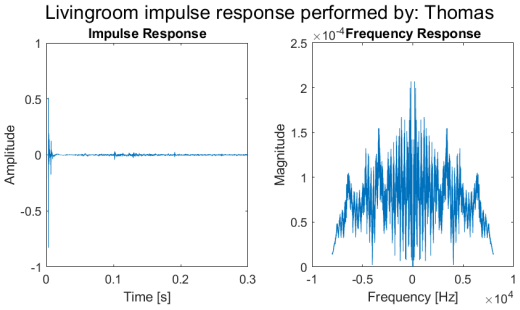
Impulse response signals are sampled at 16kHz by a DAC. To simulate this process, the matlab `resample` function can be used. The following code resamples each of the impulse response signals (stream 1, 3, 5) to 16kHz:

```
FS_recov = 16e3;                                                    % ADC sample rate [Fs]
impulses_recov = resample(msg((1:2:end), :).', FS_recov, fs).'; % Resample each impulse response
↪ stream
```

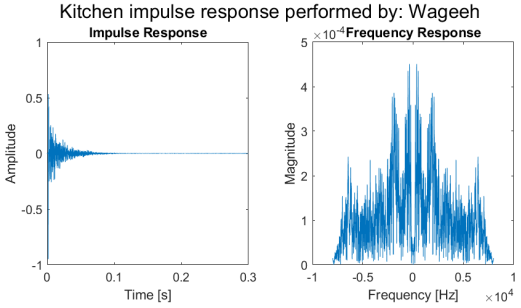
With impulse responses resampled, new time and frequency vectors conforming to Nyquist sampling parameters were created to plot the digital signals in Fig. 3.6. Text streams were decoded using the supplied `decoder` function, displaying the room and astronaut who took the measurements in the title of each corresponding plot.



(a) Laboratory



(b) Livingroom



(c) Kitchen

Figure 3.6: Received Impulse Responses

3.5 Practical Analysis

Finally, to confirm that the impulse responses received are an accurate representation of the locations they were gathered at, the impulse responses were applied to recordings of our own voices. We can do this using a convolution in the time domain, knowing that this represents multiplication in the frequency domain:

$$f(x) * g(x) \xrightarrow{\mathscr{F}} F(f) \times G(f)$$

In MATLAB, the voice recording is read into a row vector, and first resampled to match the sampling rate of the impulse responses. The resampled recording can then be convolved with the impulse response of each room.

```
[voice, fs_voice] = audioread("voice.wav");
% convert to row vector and only use one channel
voice = voice(:, 1).';
voice = resample(voice, FS_recov, fs_voice);                    % Resample voice to match impulse
↪ responses
% apply the impulse response as a convolution in the time domain
voiceTest = zeros([size(impulses_recov, 1) length(voice)]);     % Initialise size for performance
for k=1:size(impulses_recov, 1)
    voiceTest(k, :) = conv(voice, impulses_recov(k, :), 'same');% 'same' ensures length is preserved
end
```

Comparing the original voice recording to the resultant signals in Fig. 3.7, it can be seen that the original shape of the waveform is preserved with minor variations. The resultant waveforms have distinctly different sounds to them, as if the recording was made at the location of the impulse responses. Specifically, when convolved with the impulse responses of these rooms, there is a clear echo present in all three results, with the most obvious being the living room impulse response.

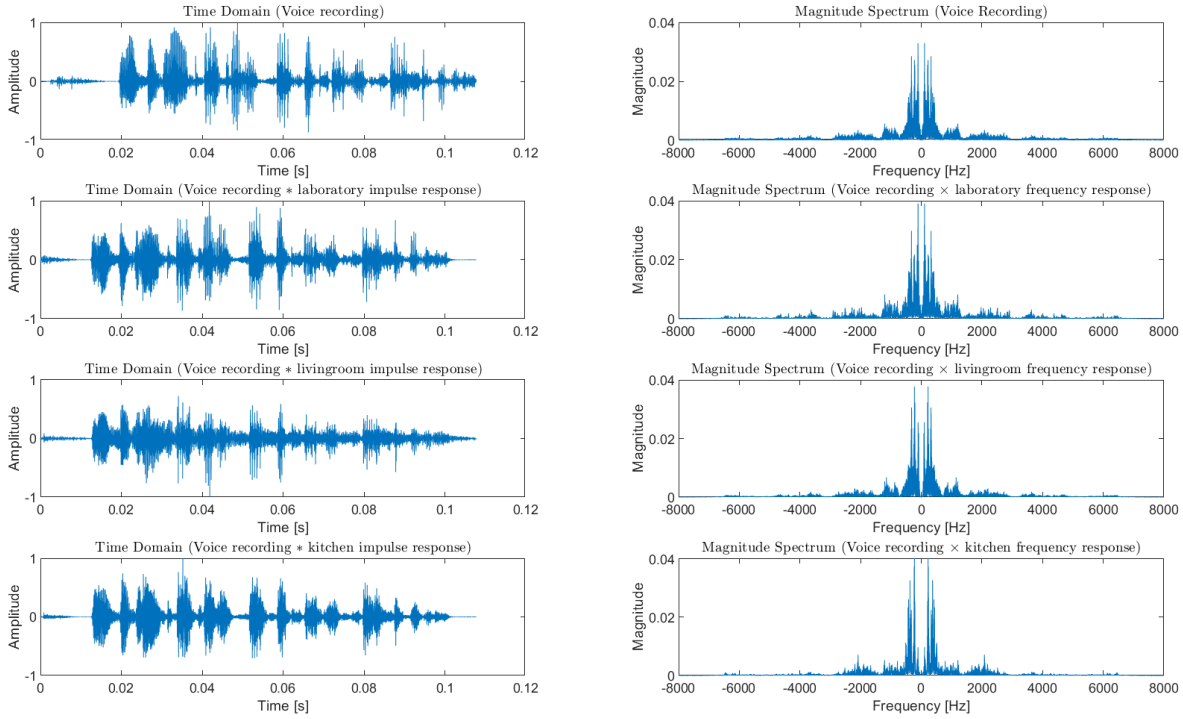


Figure 3.7: Voice and Impulse Response Testing

4 Reflection