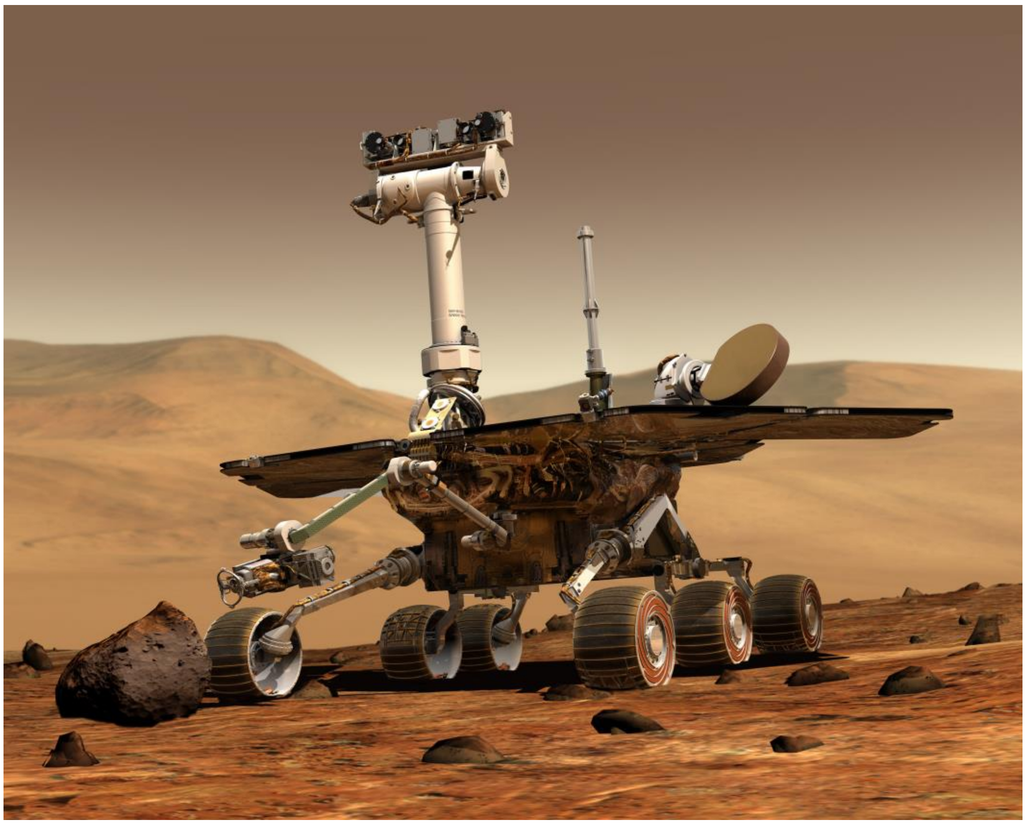
Signal Analysis – EGB242

Engineering the Crewed Mission to Mars

Zackariya Taylor – n11592931
Ben Neal – n10967753
Raymund Domingo – n11560703



Introduction

Previously, a communication system was developed to communicate between BASA and the MARS-242 mission crew. However, a problem has arisen in the communications system's ability to perform in extraterrestrial environments. The transmitter and receiver were working as expected until the craft left Earth's atmosphere, where audio signals became inaudible. This poses a threat to the safety of the crew and the reliability of data collection. Once these signals are validated, a rover is to be deployed on the surface of Mars to retrieve visual information to send to the crew entailing a safe landing zone for the craft. These images must be clear and denoised to ensure the safety and security of the crew's landing.

1.0 De-noising the Communication Channel

Figure 1.1 depicts a model for the channel, including realistic conditions.

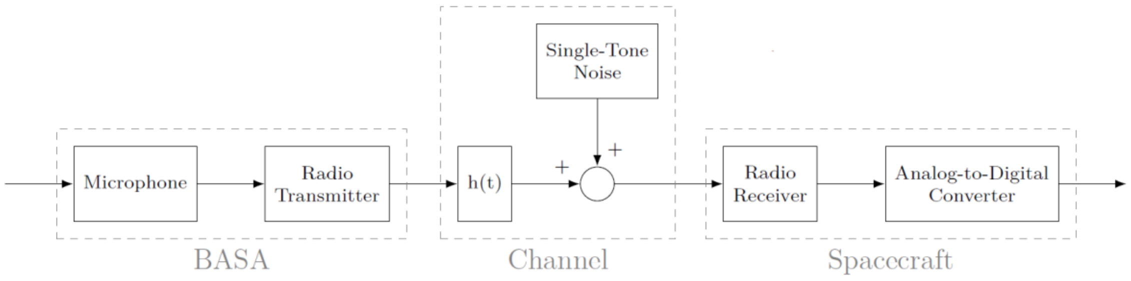


Figure 1.1: Audio channel model

The atmospheric distortion is given by $h(t)$, a frequency-dependent system (LTI system). A single-tone noise is added to the channel, providing further interference which will need to be attenuated. The signal being transmitted through the model (Figure 1) is a multiplexed audio signal. A multiplexed signal is multiple streams of information being transmitted over a single line/channel, where each signal is modulated at a distinct carrier frequency (Sheldon & Burke, n.d.). Five different messages are being transmitted over the channel and can be seen in the frequency domain (Figure 1.2).

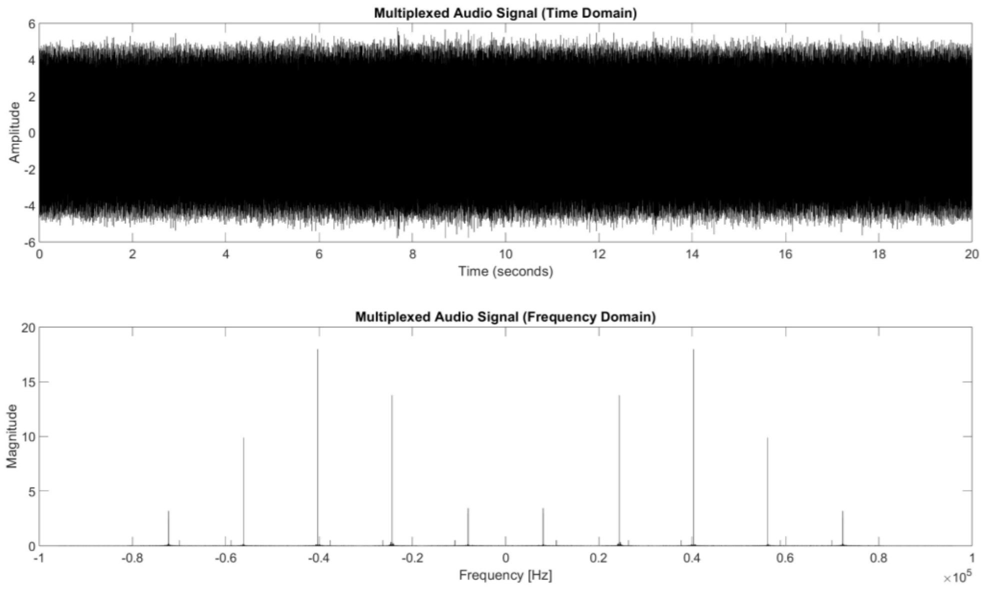


Figure 1.2: Multiplexed signal

Upon visual inspection, the carrier frequencies of each audio signal can be determined and demodulated.

```
% General form for demodulating signals:
% Signal centred at X kHz.
carrierFrequency = X;

carrierSignal = cos(2*pi*carrierFrequency*t);

% Demdulation.
signalDemodulated = audioMultiplexNoisy .* carrierSignal;

sound(signalDemodulated, fs);

% Convert to frequency domain.
SignalDemodulated =  ft(signalDemodulated, fs);
```

Figure 1.3: MATLAB code to demodulate multiplex audio

The first signal inspected was centred around 8050 Hz.

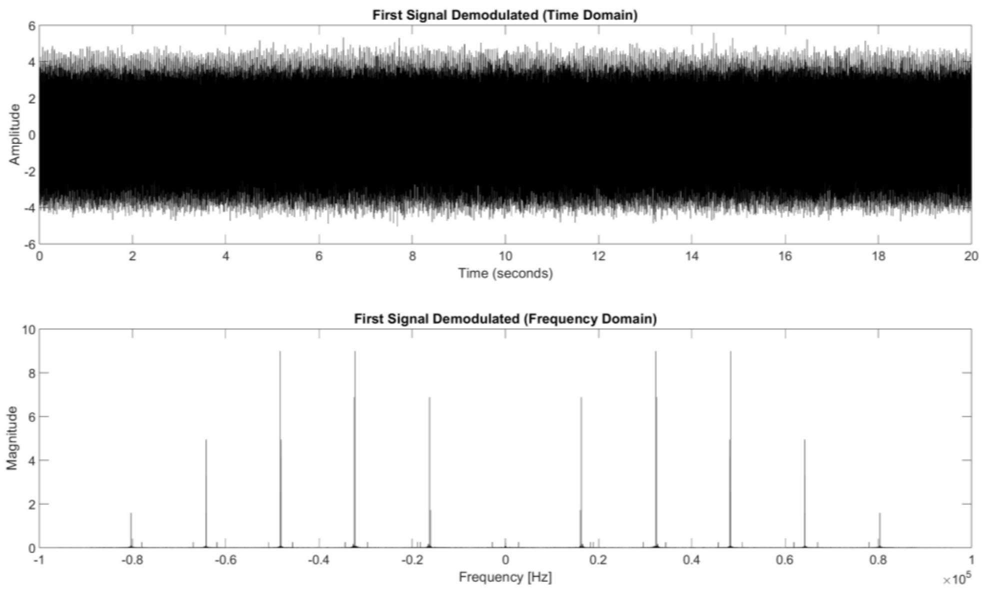


Figure 1.4: First signal demodulated

The second audio signal's carrier frequency was determined to be 24.33 kHz.

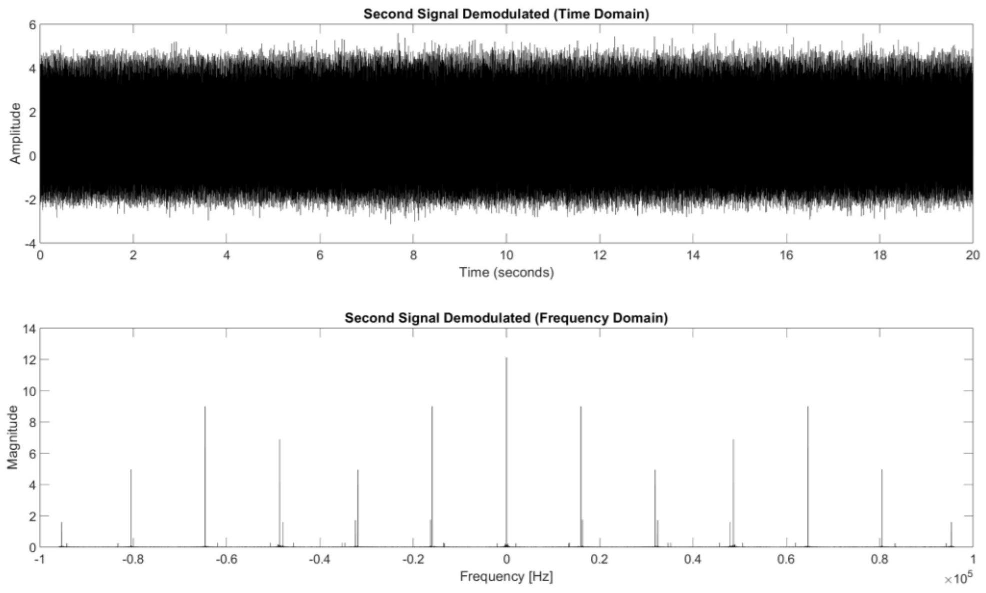


Figure 1.5: Second signal demodulated

The third signal was centred around 40.27 kHz.

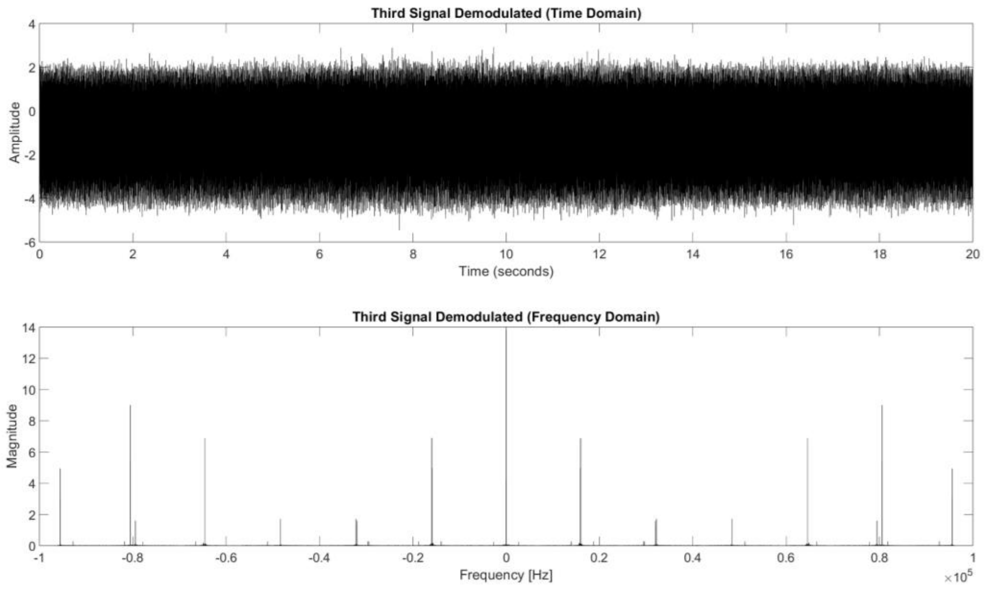


Figure 1.6: Third signal demodulated

Centred around 56.17 kHz the fourth signal was found.

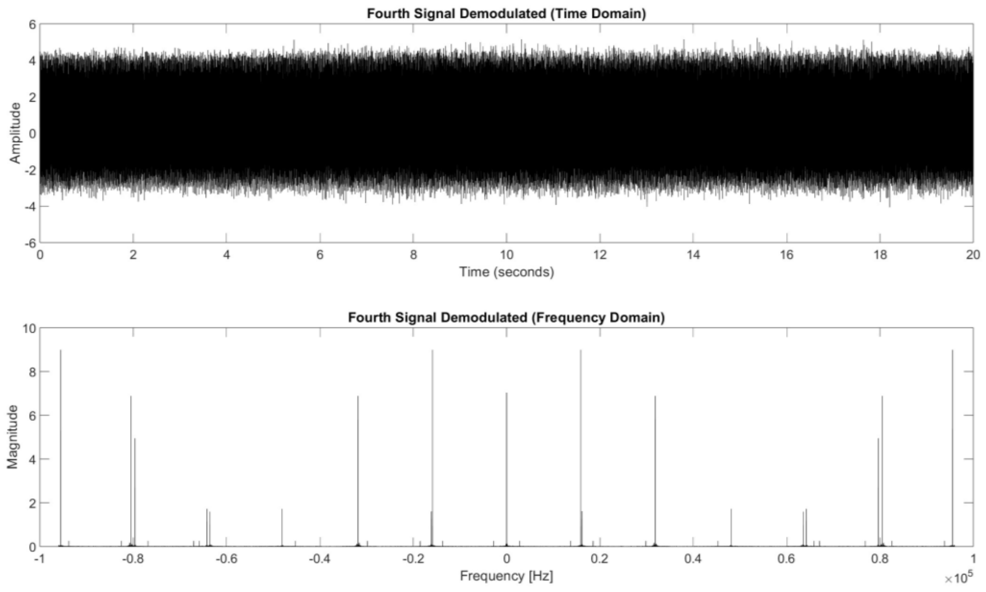


Figure 1.7: Fourth signal demodulated

Finally, the fifth signal was centred around 72.29 kHz.

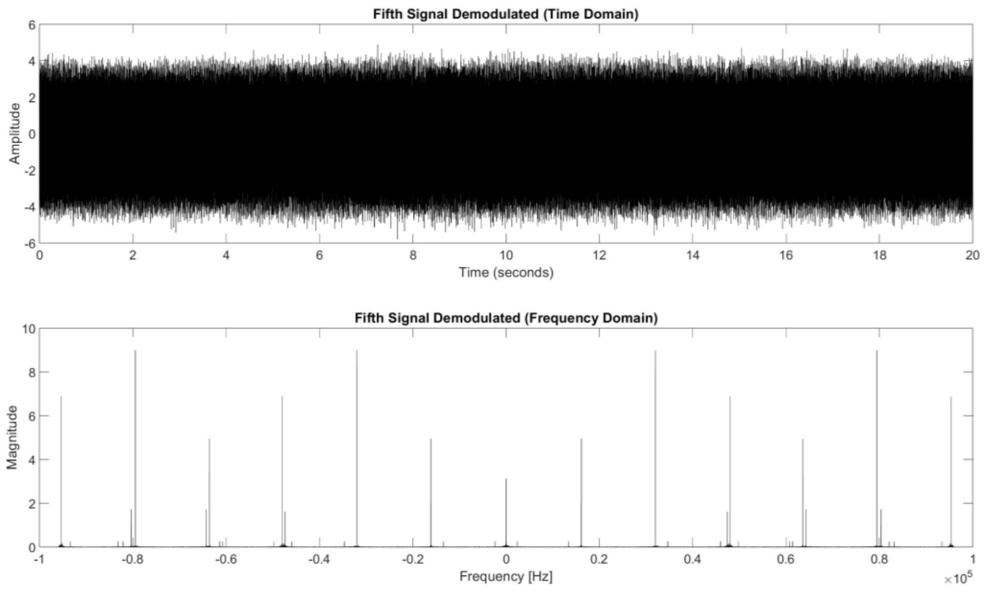


Figure 1.8: Fifth signal demodulated

When listening to these signals, a drone static pulsing could be heard with no true harmonics being audible. This would be due to the h(t) LTI system seen in Figure 1, which can be seen to be affecting the magnitude of each frequency band across each signal's spectrum.

The noise in the audio signals is caused by the frequency-dependent distortion $h(t)$ (Figure 1). The channel is a linear time-invariant (LTI) system, a time-invariant system will behave consistently over time. Using this characteristic, if we can know the response of the system to an input signal, we can infer its response to any input and determine the relationship between the input and output.

The impulse response of an LTI system is the system's output when the input is an impulse function, using the Dirac delta function $\delta(t-a)$. This function is defined by:

$$\delta(t-a) = \begin{cases} \infty, t = a \\ 0, t \neq 0 \end{cases}$$

The received signal $y(t)$ is the convolution of the transmitted signal $x(t)$ with the impulse response of the channel $h(t)$. Convolution can be mathematically represented by the convolution integral:

$$y(t) = \int_0^\infty x(\tau)h(t-\tau)d\tau = x(t) * h(t)$$

Given that convolution in the time domain corresponds to multiplication in the Laplace domain (Stanford University, n.d.).

$$Y(s) = X(s) \times H(s)$$

The Laplace transform of the delta function is given by:

$$X(s) = \mathcal{L}\{\delta(t-a)\} = \int_0^\infty \delta(t-a)e^{-st}dt = e^{-sa}$$

Therefore,

$$X(f) = \mathcal{L}\{\delta(t)\} = 1$$

$$Y(s) = H(s)$$

Applying the inverse Laplace transform to both sides:

$$\mathcal{L}^{-1}\{Y(s)\} = \mathcal{L}^{-1}\{H(s)\}$$

$$y(t) = h(t)$$

Therefore, when the input of an LTI system is the delta function, its output will be equal to the impulse response $h(t)$. The system linearity can be used to easily create an input, without the need to produce an infinitely large peak for the delta function. The magnitude of $\delta(0)$ is a scaling factor, meaning that any value at $t = 0$ will produce the same impulse response $h(t)$, scaled proportionally to the magnitude of $\delta(0)$. By creating a zero vector with the length of the audio signal, where the first element in the array is an arbitrary number, an impulse response can be simulated.

```
% Initialise vector of zeros.
impulse = zeros(1, length(audioMultiplexNoisy));
 % Set t=0 to an arbitrarily large number, simulating a delta function.
impulse(1) = fs;
```

Figure 1.9: Simulating impulse response

```
% Input the Dirac delta function into the channel.
h_t = channel(sid, impulse, fs);
% Convert the output to the frequency domain.
H_f = ft(h_t, fs);
```

Figure 1.10: Finding LTI system

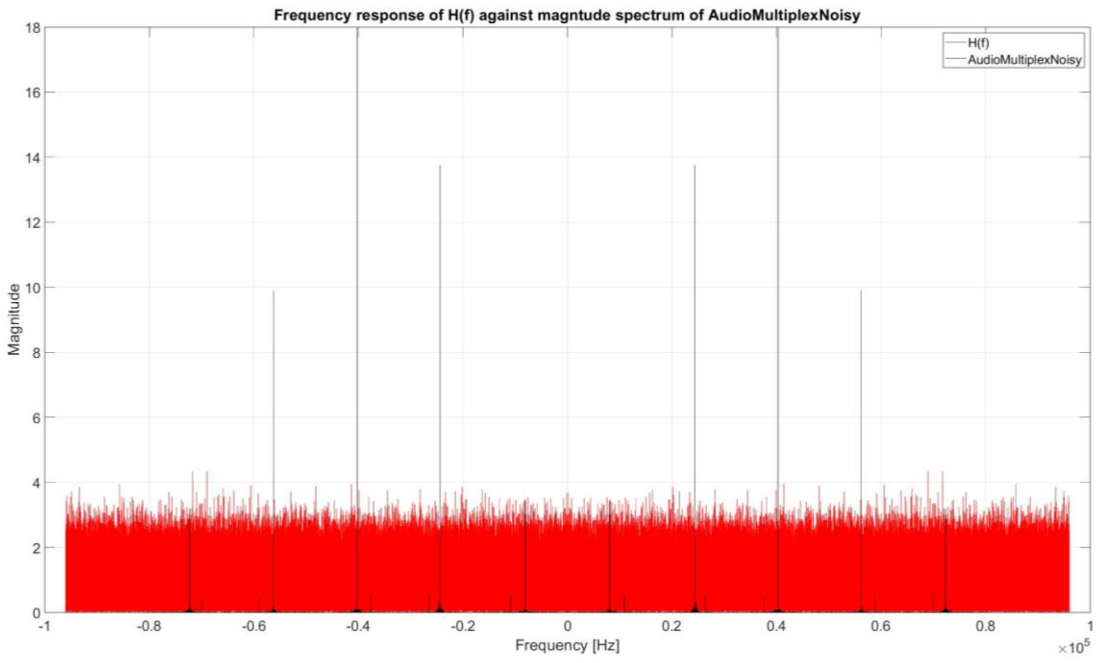


Figure 1.11: Multiplex audio against LTI system

After evaluating this figure, it was seen that each individual signal found on the multiplexed signal had varying magnitudes, with signals at 8050 Hz and 72.29 kHz even matching the peaks of the LTI system. This is because an LTI system will amplify/attenuate the signals in the multiplexed audio based on the magnitude of the system at each respective frequency (Yagle, 2005). This is indicated in the figure above, as when the LTI system has a lower magnitude at a given frequency, the multiplex signal at that frequency is attenuated accordingly.

Using the model of the frequency-dependent distortion, the noise process can be reversed. Since the noising process acts like an LTI system, it can be modelled as below:

$$\frac{A_{output}(f)}{A_{input}(f)} = H(f)$$

$$A_{input}(f) = \frac{A_{output}(f)}{H(f)}$$

This allows us to calculate the clean signal that was inputted into the channel, using the impulse response and the noisy output. The frequency spectrum of this cleaned signal was given as:

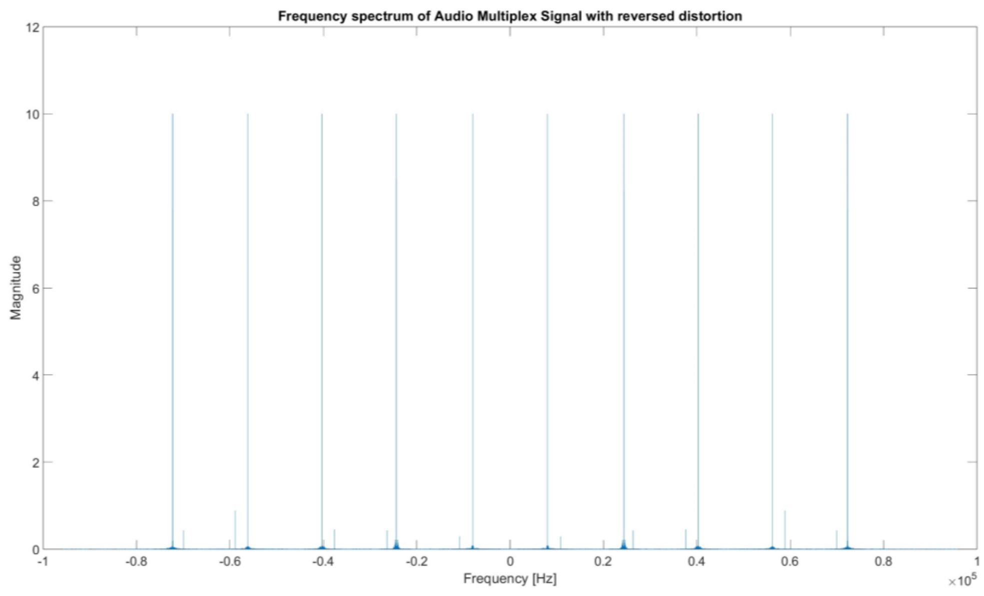


Figure 1.12: Reversed distortion audio frequency spectrum

With the removal of the LTI system, it was shown that the amplification/attenuation of each signal in the multiplex was reversed. This left each signal with an equal frequency magnitude, proving that the cause of this was the LTI system. It can also be seen that each signal had a set of single-tone frequencies surrounding that central signal. This indicated that there was still noise present in each signal that needed to be removed.

Each signal was then demodulated again by the carrier frequencies previously found (8050 Hz, 24.33 kHz, 40.27 kHz, 56.17 kHz and 72.29 kHz). To ensure each signal was properly normalised, the DC offset of each signal was removed. This was done by finding the average of the signal using the following equation:

$$\mu = \frac{1}{N}\sum_{i=0}^{N-1} t_i$$

Which is given using the `mean()` function in MATLAB, and then subtracting this mean from the initial signal. This ensured that all signals' amplitudes were properly centred around zero in the time domain (Smith, 1997).

After listening to each signal, it was heard that the distortion was reversed. However, there was still a large amount of noise remaining as shown in this graph:

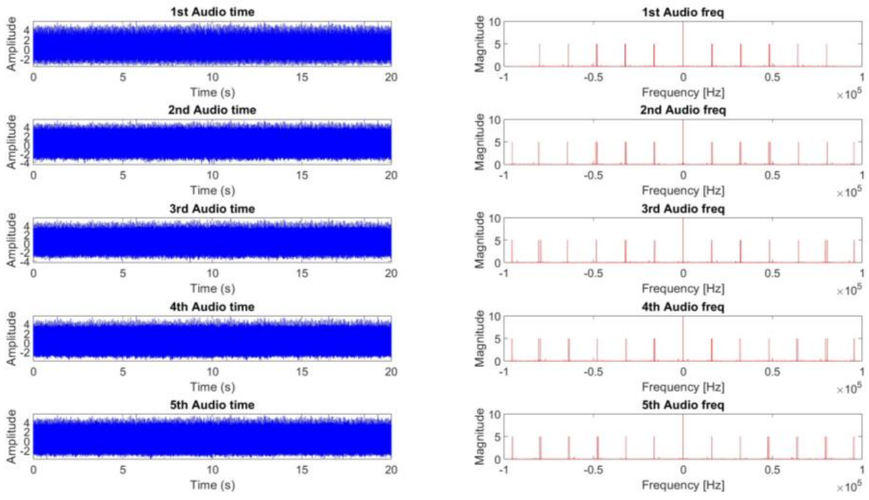


Figure 1.13: Time and frequency of demodulated signals after LTI removal

Nevertheless, a clean audio signal was starting to be heard beneath the noise seen above. This indicated that each signal needed to be processed through a lowpass filter to attenuate these higher unwanted frequencies causing noise. Each signal underwent the same process, as it was identified that each signal had similar intermittent noise across the higher frequency ranges.

The signals were initially passed through a lowpass filter that was created with a passband frequency of 10 kHz. This value was chosen as analysis of each signal's frequency spectrum showed 10 kHz to be an appropriate frequency for where there was no further noise below that band. After filtering, each signal became:

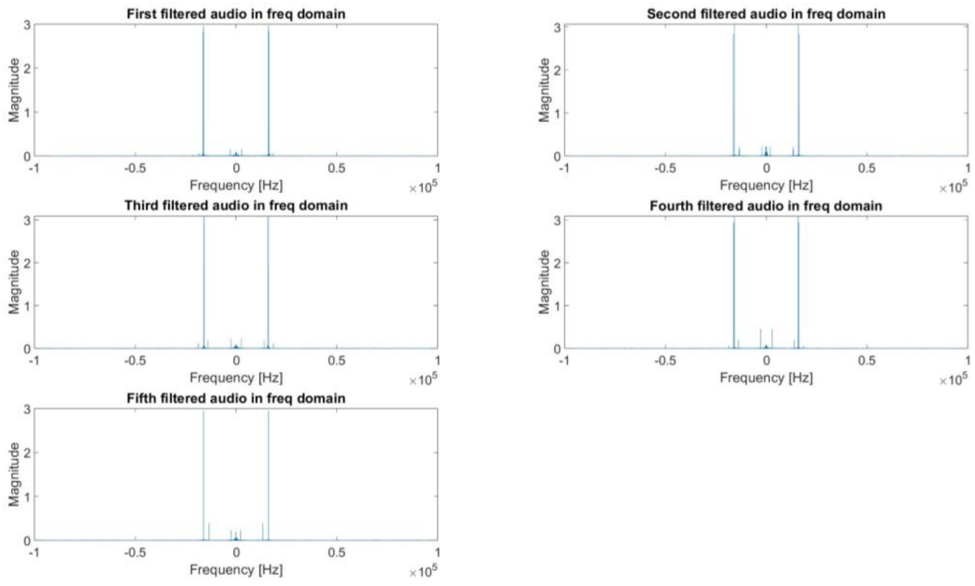


Figure 1.14: Signal frequency spectrums after initial lowpass filtering

It's evident that this filter initially wasn't suitable, as there were still tones around 2-3 kHz and 16 kHz which were audible in each audio signal.

Using this information, it was decided that a cascading filter system made up of a lowpass and band-stop filter was needed to filter out the noise sufficiently. This is because a lowpass filter placed with a f_{pass} of 2-3 kHz would not sufficiently remove the single tone at this frequency band because of the behaviour of this type of filter individually. This would cause a severe lack of important audio information to be retained and would defeat the purpose of removing the noise. Lowpass filters don't precisely cut all frequencies above a given cutoff, rather they transition smoothly from the passband to a given stopband (Keim, 2018).

Using an f_{pass} frequency of 2 kHz with a filter of this behaviour would attenuate the signals seen at the 16 kHz range, however, the single tones in the 2-3kHz range would still be quite present. This is further validated by this graph representation of this filter created in MATLAB:

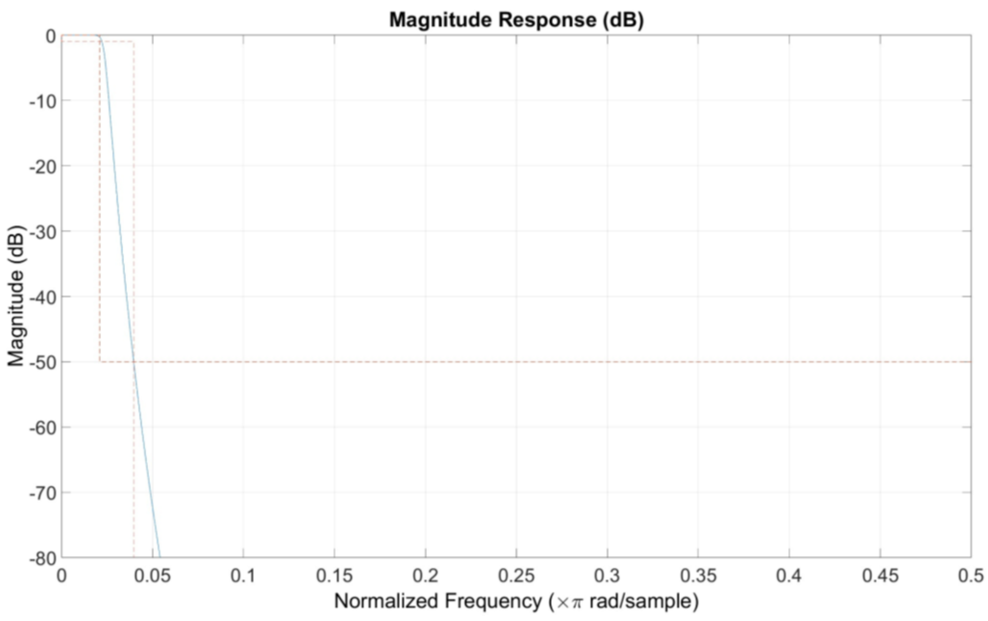


Figure 1.15: Lowpass filter behaviour

The x axis is measured as normalized frequency, which is given as a fraction of the Nyquist frequency of the signal. This is a common dimensionless measurement that is often used when describing the behaviour of filters. The Nyquist frequency f_N is half of the sampling frequency of the signal (192 kHz in this case, which gives f_N of 96 kHz). Normalised frequency is found by this equation:

$$f_{norm} = \frac{f}{fN}$$

The Nyquist frequency is then equal to 1 as the normalised frequency (Proakis & Manolakis, 2007). Using this information, it can be seen from the magnitude response of the lowpass filter given that it would not properly attenuate or reduce the magnitude of the frequencies in that 2-3 kHz range (for reference, 0.05 as a normalised frequency gives 4.8 kHz).

This further validated the need for a second cascading band-stop filter which would attenuate these remaining frequencies in this range. A band-stop filter attenuates signals between the two given cutoff frequencies while passing all other frequencies outside this range (Electronics Tutorials, 2018b).

The final parameters that were chosen for this cascading filter system were an f_{pass} of 2kHz for the lowpass filter and a band-stop range of 1.8-3kHz. These ranges would mean that higher frequencies included in the original signal would be lost, however, these values ensured that there was still a sufficient balance between noise removal and retention of signal information. The steepness of the lowpass filter was also set to 0.9 to ensure that the frequencies at 16 kHz were properly attenuated.

MATLAB was used to create this cascading filter system using a for loop, which was done for greater maintainability and to ensure all signals went through the same process. The code was developed to be:

```
signals = {demodAudioMultiplexReverse1, demodAudioMultiplexReverse2, demodAudi-
oMultiplexReverse3, demodAudioMultiplexReverse4, demodAudioMultiplexReverse5};
newSignals = {'filteredAudioMultiplexReverse1', 'filteredAudioMultiplexReverse2',
'filteredAudioMultiplexReverse3', 'filteredAudioMultiplexReverse4', 'filteredAudi-
oMultiplexReverse5'};
filteredSignals = struct();
fpass = 2e3;
bandfpass = [1.8e3 3e3];

for i = 1:length(signals)
    lowSignal = lowpass(signals{i}, fpass, fs, "Steepness", 0.9, "StopbandAttenua-
tion", 60);
    bandSignal = bandstop(lowSignal, bandfpass, fs);
    filteredSignals.(newSignals{i}) = bandSignal;
end
filteredAudioMultiplexReverse1 = filteredSignals.filteredAudioMultiplexReverse1;
filteredAudioMultiplexReverse2 = filteredSignals.filteredAudioMultiplexReverse2;
filteredAudioMultiplexReverse3 = filteredSignals.filteredAudioMultiplexReverse3;
filteredAudioMultiplexReverse4 = filteredSignals.filteredAudioMultiplexReverse4;
filteredAudioMultiplexReverse5 = filteredSignals.filteredAudioMultiplexReverse5;
```

Figure 1.16: MATLAB for loop to filter signals.

The behaviour of this filter system plotted shows:

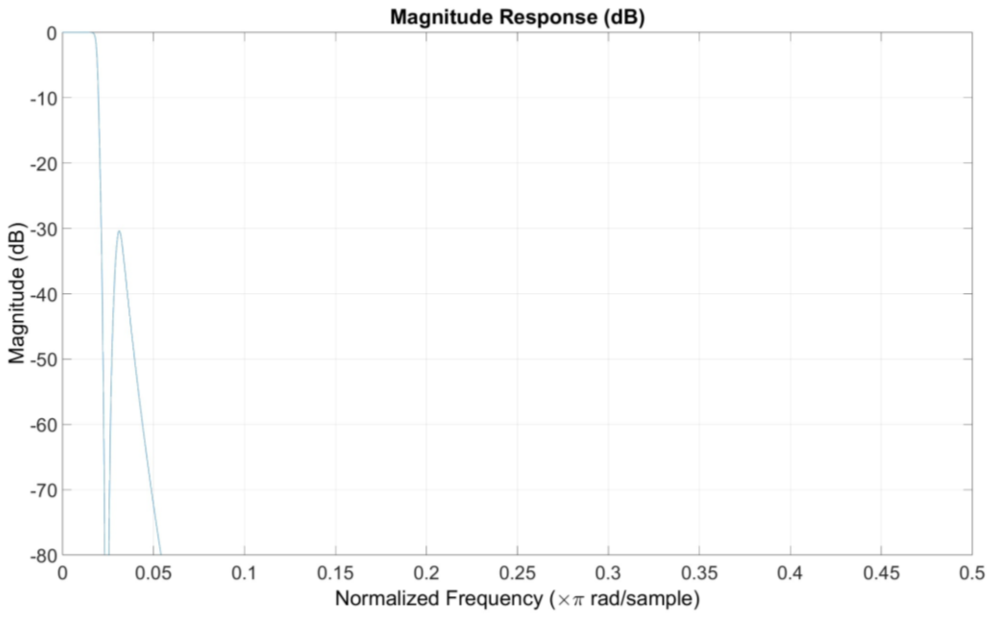


Figure 1.17: Cascading filter system behaviour

It can be seen that the curve of the filter system from -30dB to -80dB is more or less the same as the curve given from the lowpass filter individually, meaning that the behaviours of both

filters are being used to their full potential without affecting the other and attenuating unnecessary frequencies.
Plotting these final signals showed:

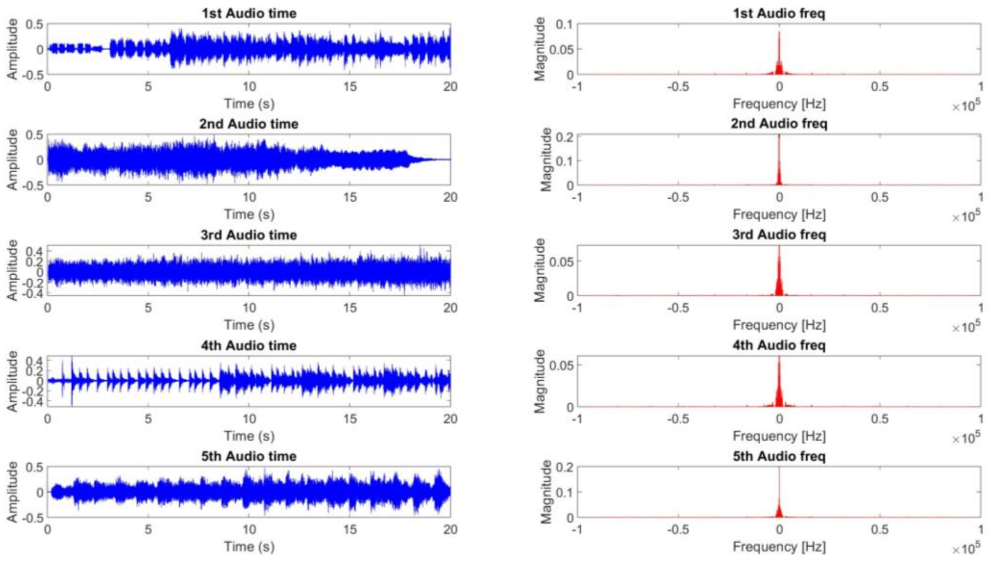


Figure 1.18: Final signals after noise removal

After listening back to these signals, all noise was removed. However, there were still some issues that stemmed from the original structure of the multiplex signal. In signals 3 and 5, some signal bleed could be heard. This occurs when audio information from one channel or source unintentionally interferes with and becomes audible in another channel or source. Listening to these signals, it was clear that this phenomenon was occurring (Brown, 2011).

In the background behind the aforementioned signals 3 and 4, the percussion from signal 4 was audible, indicating signal bleed (or cross talk). The following table shows the carrier frequencies of these signals, and the separation between each:

	Carrier Frequency (Hz)	**Frequency separation from Signal 4 (Hz)**
Signal 3	40,270	15,900
Signal 4	56,170	NA
Signal 5	72,290	16,120

Signals on a multiplex channel will generally have “guard band” frequencies, which are small bands of frequencies to add extra separation between signals to reduce bleed. This value is combined with the bandwidth of each signal to show the minimum separation between signals on a channel (Proakis, 1995). However, it is known that each signal could arguably have a bandwidth of up to 16 kHz due to the initial noise occurring on the system, and the remaining noise at that frequency band after the LTI system was removed. Therefore, the separation of 16 kHz for these 3 signals is inadequate and would explain the signal bleed. Furthermore, only percussion from signal 4 was heard bleeding into these 2 adjacent signals, which can be due to percussion sounds often having sharp transients and broad frequency contents making them more susceptible to bleeding into other signals on a multiplex channel (Rossing et al., 2014). To remove this bleed, the initial multiplexed channel would need to have the signals modulated to a greater separation degree relative to the spectral content of each audio signal.

2.0 Rover Camera Control

Once the spacecraft arrives on Mars, the Mars rover will traverse the landscape to find a suitable landing location. The rover is equipped with a camera which can send images of candidate landing locations back to the MARS-242 crew.

The camera's ability to rotate to capture panoramic images is crucial for the crew, allowing them to effectively choose a landing site. The angle of the camera is controlled by a servo motor, which rotates from 0 to 2π radians around the z-axis (given by a function of time, $\psi_{out}(t)$), its rotation is controlled by an input voltage signal $v_{in}(t)$. The input signal is binary and range from a logic low (0V) to a logic high (1V).

The input signal, motor, and output rotation can be modelled as an LTI system (Figure 2.1).

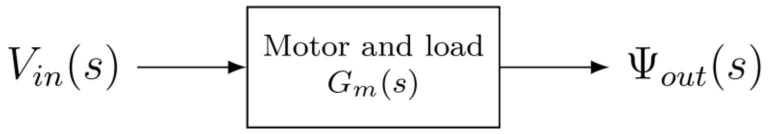


Figure 2.1: LTI system block diagram (EGB242 Assessment 2 task sheet)

$$G_m(s) = \frac{\Psi_{out}(s)}{V_{in}(s)} = \frac{1}{s(s+0.5)}$$

The step response in control systems provides insights into the system's stability and transient response (StudySmarter, n.d.), as well as showing how a system responds to a sudden change in input signal voltage (in this case, 0V to 1V). The output of the system, $\psi_{out}(t)$, should stabilise at 2π so that the camera does not overturn.

The step response of the transfer function can be determined by multiplying $G_m(s)$ by the unit step function $u(s) = \frac{1}{s}$, then computing the inverse Laplace transform.

$$c(t) = 2t + 4e^{-0.5t} - 4u(t)$$

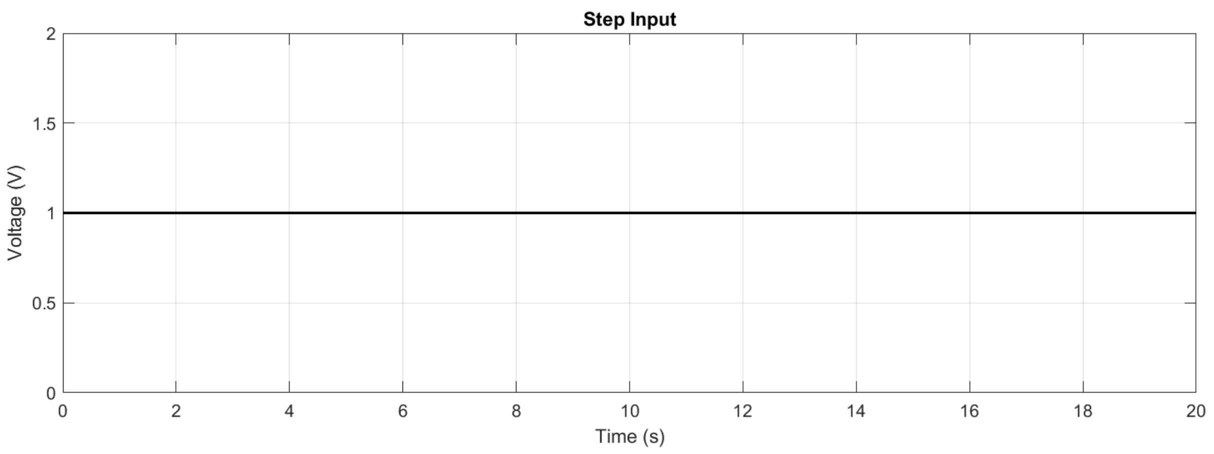


Figure 2.2: LTI system's step input (1V)

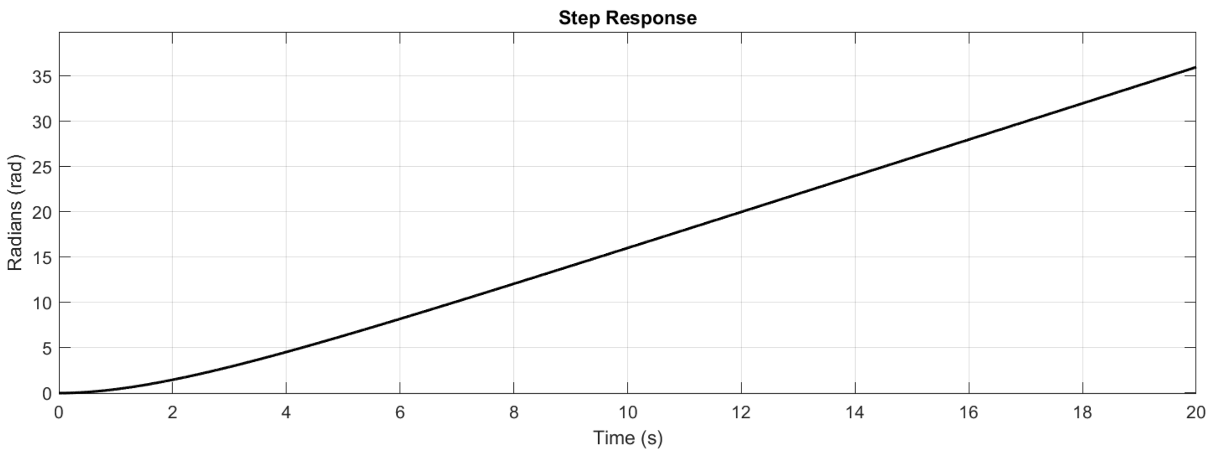


Figure 2.3: LTI system step response.

When the input is 1V for 20 seconds, the step response produces a graph (Figure 2.3). As seen in Figure 2.3, the rotation of the camera exceeds 2π radians, reaching up to 35 radians, deeming the system as unstable.

To quantitively determine the stability of the system, and to reinforce the belief that the step response is unstable, the poles of the control system can be derived (Figure 2.4) and analysed.

$$s(s + 0.5) = 0$$
$$\sigma_{1,2} = -0.5, 0$$

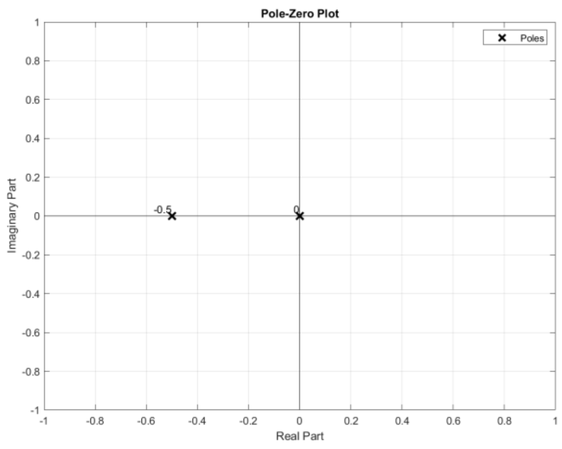


Figure 2.4: Poles of the LTI system

When $\sigma < 0$, the pole is stable and has an exponentially decaying component. When $\sigma = 0$, the pole is marginally unstable, and the systems behaviour is determined by initial conditions (Massachusetts Institute of Technology Department of Mechanical Engineering, n.d.). In this case, the system's marginally unstable pole is associated with the linear component of the step response $(2t - 4u(t))$, which causes the step response to grow linearly and unboundedly.

To control the camera's yaw accurately without exceeding a full rotation, a feedback system is required. A feedback system is where the output signal is manipulated and fed back into the input. There are two main types of feedback systems; positive feedback, and negative feedback (Electronics Tutorials, 2018a). A positive feedback system is where the output signal is processed and added to the input signal before being applied to the system. Characteristics of such a system may cause instability and oscillations, which is undesirable in this application.

On the other hand, a negative feedback system is where the output is processed and subtracted from the input, before being applied to the system. This aids with system stability and accuracy (Lumen Learning, 2019), deeming it suitable for the camera rotation's LTI system.

Coupling a potentiometer to the motor's axle can generate a voltage, $v_{pot}(t)$, proportional to the angular displacement of the camera. The potentiometer can be used in the negative feedback system to provide live information about the camera's rotation.

The relationship between the potentiometer's voltage and output rotation in the Laplace domain is given by:

$$H_p(s) = \frac{V_{pot}(s)}{\Psi_{out}(s)} = K_{pot}$$

The new feedback LTI system is then visualised as (Figure 2.5):

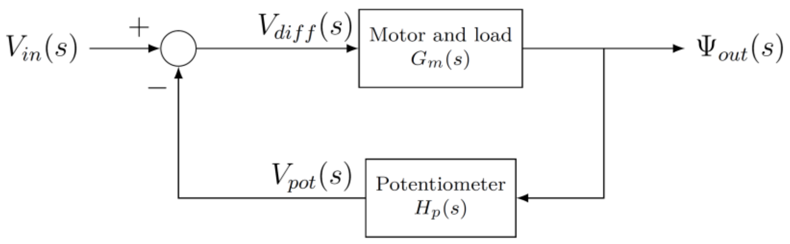


Figure 2.5: Modified feedback LTI system

If $K_{pot} = 1$, then the transfer function is:

$$F(s) = \frac{\Psi_{out}(s)}{V_{in}(s)} = \frac{G_m(s)}{1 + G_m(s)} = \frac{1}{s^2 + 0.5s + 1}$$

This gives a second-order transfer function with complex conjugate poles at $-\frac{1}{4} \pm \frac{\sqrt{15}}{4}j$.

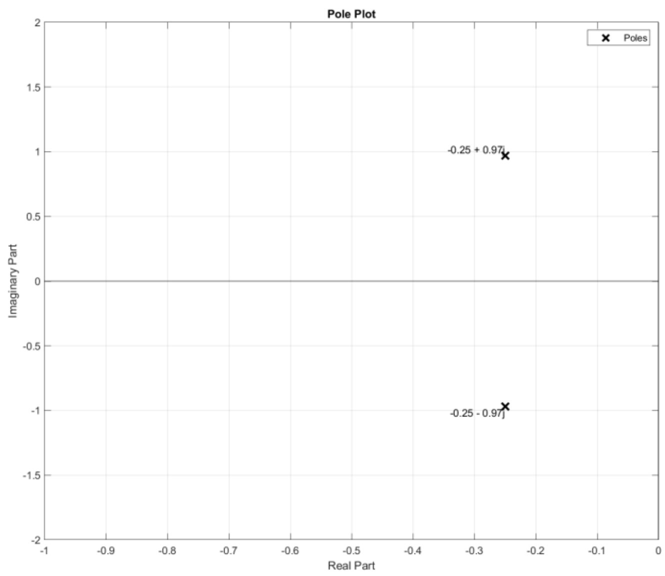


Figure 2.6: Poles of the modified feedback LTI system

The system is now stable as $\sigma < 0$ is satisfied for both poles. Using the 'lsim' function in MATLAB, the new step response can be plotted.

```
% Define transfer function G1
numerator_G1 = [1];
denominator_G1 = [1, 0.5, 1];
G1 = tf(numerator_G1, denominator_G1);

% Use lsim to simulate the response of the feedback system
y1 = lsim(G1, step_input, t2);
```

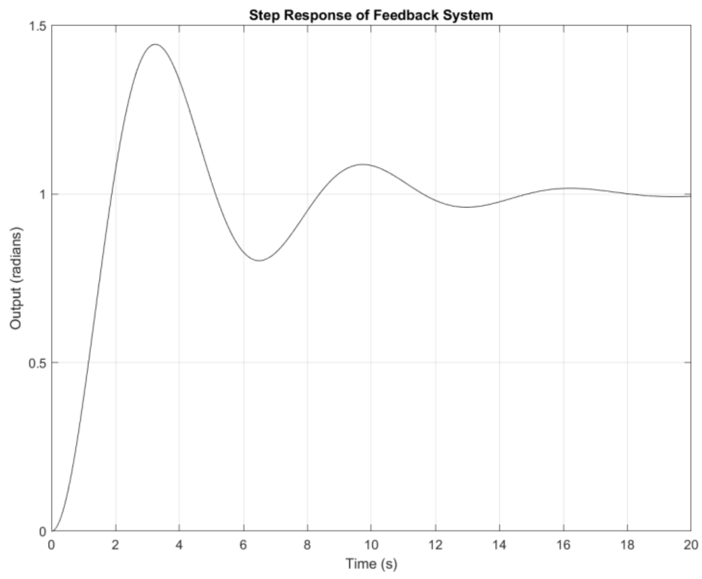


Figure 2.7: Step response of modified feedback LTI system

Using a negative feedback system provided the system with stability, so that as $t \to \infty$, the step response of the system converges to 1 (Figure 2.7). This system is more suitable as the step response resists change regardless of the input, preventing it from rotating further. Stability provides the system with reliability and performance consistency (Paradkar, 2024), which is desirable for the high stakes of the MARS-242 mission. The specifications of a second-order system include the natural frequency (ω_n) and damping ratio (ζ). The natural frequency is the frequency of oscillation of the system without damping, and the damping ratio is the ratio between the exponential decay frequency and the natural frequency.

If the transfer function takes the standard form:

$$G(s) = \frac{b}{s^2 + as + b}$$

Then:

$$b = \omega_n^2$$

$$\zeta = \frac{Exponential\ decay\ frequency}{Natural\ frequency} = \frac{\frac{a}{2}}{\omega_n}$$

$$a = 2\zeta\omega_n$$

Substituting these values back into the transfer function gives the characteristic equation:

$$G(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

By making the characteristic equation $G(s)$ equal to the systems transfer function $F(s)$, the damping ratio and natural frequency can be derived.

$$\omega_n = 1$$
$$\zeta = \frac{1}{4}$$

If $0 \leq \zeta \leq 1$, the system is deemed to be underdamped. The system is stable but oscillates before converging to the steady-state value (Control Tutorials for MATLAB and SIMULINK, n.d.), which is depicted in the step response, Figure 2.7. Using the natural frequency and damping ratio, peak time, settling time, and overshoot percent can be calculated:

$$T_P = \frac{\pi}{\omega_n\sqrt{1-\zeta^2}} = 3.24s$$

$$T_s = \frac{4}{\zeta\omega_n} = 16s$$

$$\%OS = e^{\left(\frac{-\zeta\pi}{\sqrt{1-\zeta^2}}\right)} = 44.4\%$$

Although the feedback system is now stable, it is still unable to control the full range of motion of the camera's rotation. A full rotation is 2π radians, while the step response in the graph does not meet this and settles at 1 radian, this can be mathematically justified using the final value theorem to find the steady state of the step response (Vedantu, 2024).

$$\lim_{t\to\infty} c(t) = \lim_{s\to 0} sF(s)\frac{1}{s} = 1$$

To overcome this issue, gain circuits were added to the control system and were calibrated to determine the appropriate amount of gain.

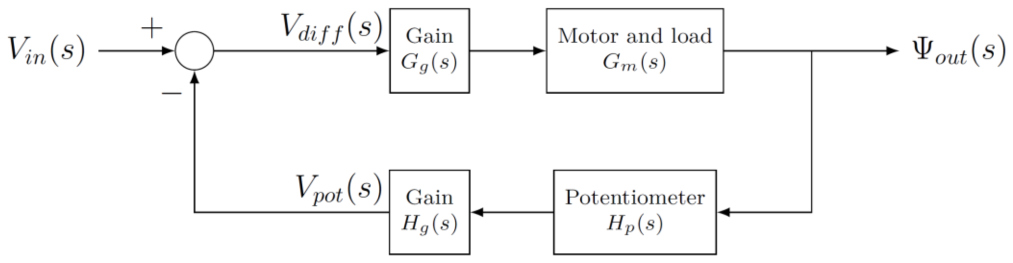


Figure 2.8: Modified feedback LTI system with gain circuits

The new transfer function is then derived to be:

$$H(s) = \frac{K_{fwd}G_m(s)}{1 + K_{fb}G_m(s)} = \frac{K_{fwd}}{s^2 + 0.5s + K_{fb}}$$

Where $K_{fwd} = G_g(s)$ and $K_{fb} = H_g(s)$.

This represents the new control system with the added gain circuits. To gain further understanding of the effect of gain on the step response, various gain values were plotted. Where $K_{fwd} = 1$, K_{fb} can take the gain magnitudes of $\{0.1, 0.2, 0.5, 1, 2\}$.

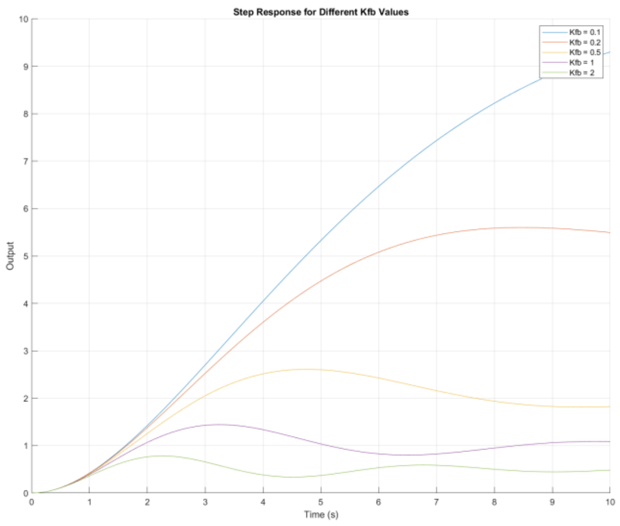


Figure 2.9: Step response for variations of K_{fb}

When $K_{fb} = 1$, K_{fwd} can take the values $\{0.1, 0.2, 0.5, 1, 2\}$.

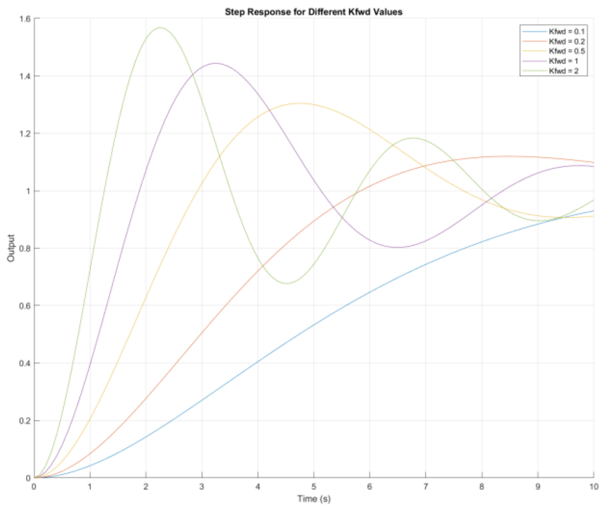


Figure 2.10: Step response for variations of K_{fwd}

Based on the output graph above (Figure 2.8), when K_{fwd} is constant, decreasing K_{fb} increases the camera's rotational steady state. On the other hand, where K_{fb} is fixed (Figure 2.9), K_{fwd} affects the transient response of the system, where a higher gain leads to a greater oscillation amplitude and natural frequency. Based on these behaviours, an ideal system will have $K_{fwd} \approx 1$ to provide maximal stability. It will also have a K_{fb} gain somewhere between 0.1 and 0.2 to allow the system to stabilise to 2π radians (one full rotation).

Based on the requirements — the camera should be able to rotate to any angle between 0 and 2π radians, and that the time to peak should be 13 seconds — the natural frequency and damping ratio can be calculated and substituted back into the general form to determine the gain magnitudes (K_{fwd} and K_{fb}).

$$H(s) = \frac{K_{fwd}}{s^2 + 0.5s + K_{fb}} = \frac{\omega_n^2 K}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

Where K is a gain factor.

$$13 = \frac{\pi}{\omega_n\sqrt{1-\zeta^2}} \rightarrow \omega_n = 0.348 \rightarrow \omega_n^2 = 0.121$$

Where $\zeta = \frac{0.5}{2\omega_n}$.

Using the final value theorem of a step response, the gain factor (K) can be calculated from the general form:

$$2\pi = \lim_{s\to 0} sH(s)\frac{1}{s} = K$$

Based on these calculations, $K_{fwd} = \omega_n^2 K = 0.760$ and $K_{fb} = \omega_n^2 = 0.121$. These values are reflected in the qualitative analysis of the graphs with different gains (Figure 2.9 and Figure 2.10). Giving the transfer function of the modified second order LTI system shows:

$$H(s) = \frac{0.760}{s^2 + 0.5s + 0.121}$$

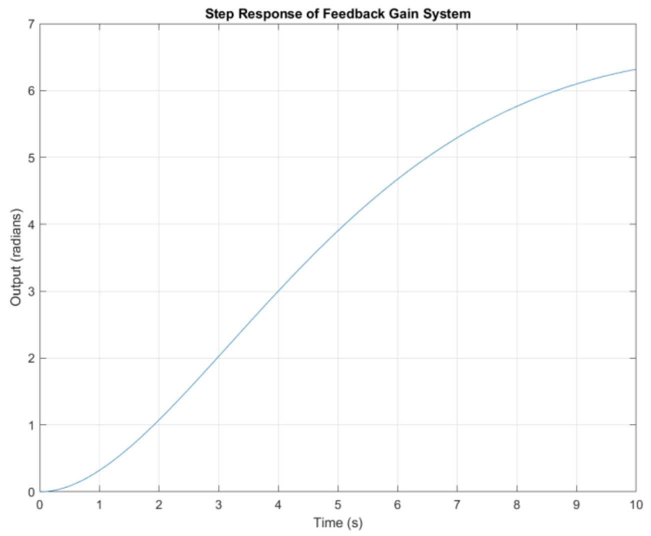


Figure 2.11: Step response of LTI system with feedback and gain

Now that the camera's control system is ready and the Mars rover has landed on red deserts of Mars, images are ready to be captured. To orient the Mars rover at the landing site, BASA HQ has decided that first task of the rover is to capture a panoramic image of the landscape, to sweep from $30°$ to $210°$.

Given the linearity of LTI systems (Chumbley, 2016), it can be assumed that the input voltage is linearly proportional to the output rotation, where $1V \rightarrow 2\pi$ and $0V \rightarrow 0$. A formula can be generated to determine the input to obtain desired outputs:

$$input_{voltage} = \frac{output_{radians}}{2\pi}$$

Using this the inputs were determined and entered into the `cameraPan.p()` function, which will provide a live feed of the panoramic sweep.

```
startVoltage = 1/12;
endVoltage = 7/12;
[startIm, finalIm] = cameraPan(startVoltage, endVoltage, cameraTF);
```

The output camera sweep was smooth and constantly in focus, met the peak time requirements of 13 seconds, and accurately swept between $30°$ and $210°$.

3.0 Choosing a Landing Site

Upon effectively finalising the design of the control system for the camera on the Mars rover, it was sent ahead of the astronauts to locate an optimal landing location. The rover has then been transmitting images of the planet's landscape to BASA HQ.

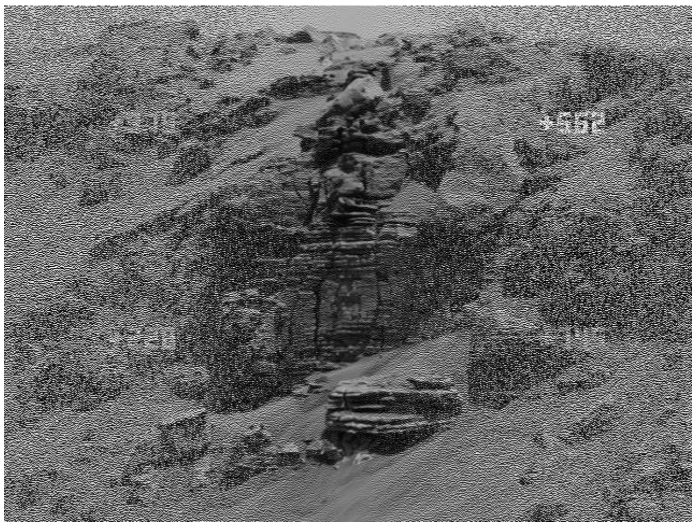


Figure 3.1: Noisy image from Mars

As encountered in previous difficulties with designing a communication system for the mission, travelling long distances and surrounding equipment may incur disturbances in the signals.

Using MATLAB for digital image processing, images are stored as 2D matrices with each element representing a pixel of the image. Working with grayscale images, the numeric value for each of the elements are floating point numbers between 0 and 1; corresponding to the intensity of colours black and white respectively (Gonzalez et al., 2020).

The images are received through the communication channel as a 1D data stream. To display the image, using the reshape function on MATLAB converts the data to a 2D matrix. The size of each image is given as 640 pixels wide and 480 pixels tall, the image matrix is organised in column-major order accordingly.

$$1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12 \rightarrow \begin{array}{cccc} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{array}$$

```
firstImage = imagesReceived(1,:);

numRows = 480;
numCols = 640;

% Converting a received pixel stream to an image matrix
reshapedFirstImage = reshape(firstImage, numRows, numCols);

% Displaying an image in a figure
figure;
imshow(reshapedFirstImage);
```

Plotting the image signal in both the time and frequency domain describes the nature of the noise and image signal. This way, the astronauts can differentiate and remove the noise from the image.

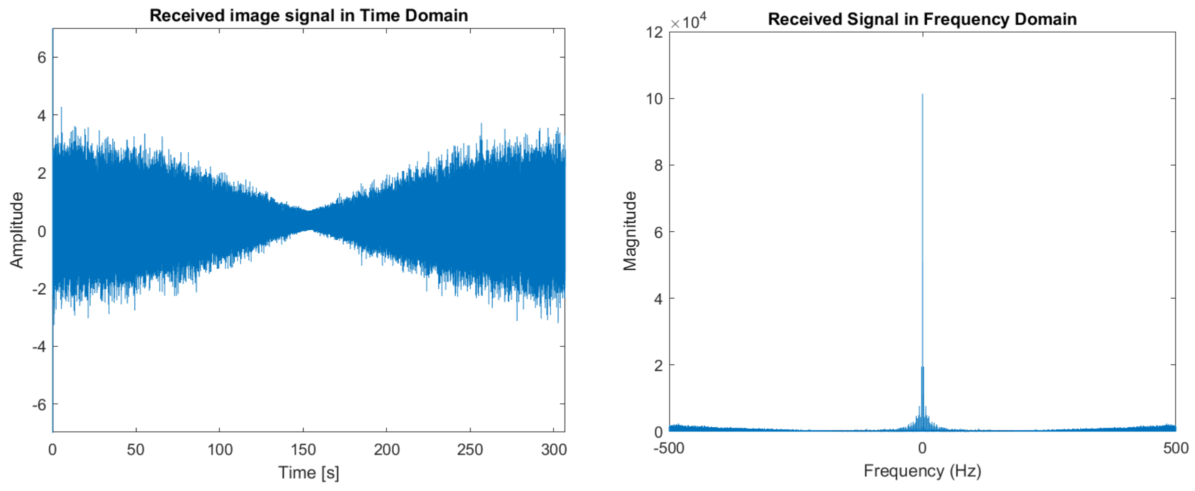
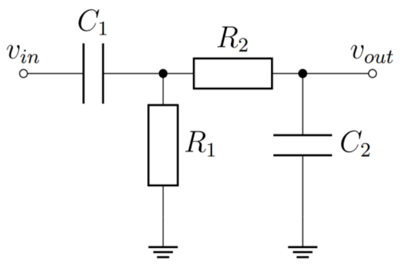
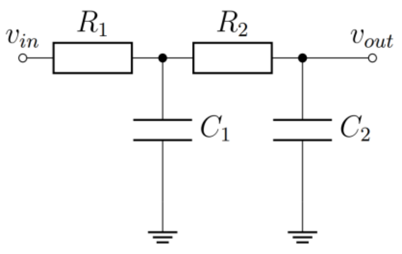


Figure 3.2: Original image signal in time and frequency domain

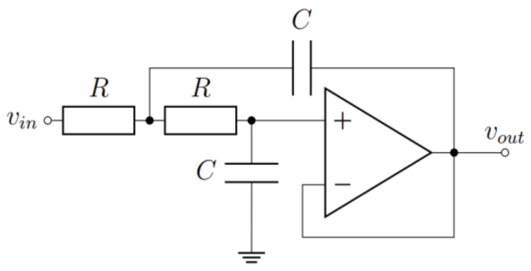
Analysing the signal in the frequency domain, it is evident that the actual image signal contains noise that needs to be attenuated. To clean these signals, it is transmitted through a signal processing filter. These filters remove any unwanted features that make a signal distorted. This works by attenuating signals over specific cutoff frequencies depending on the filter's design (Proakis & Manolakis, 1992).



Passive filter 1

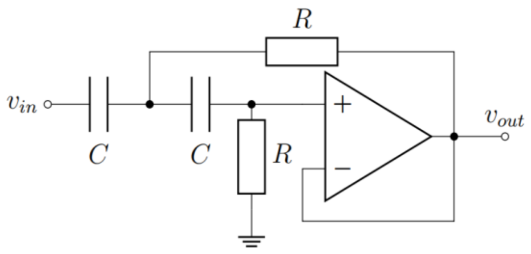


Passive filter 2



Active filter 1

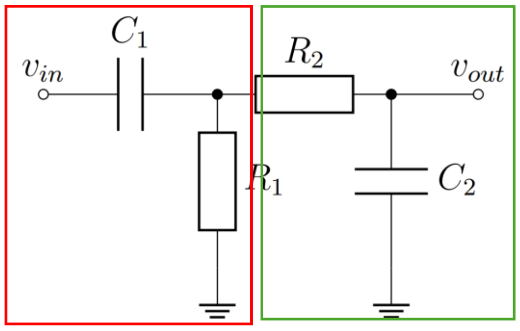
$$\frac{V_{out}}{V_{in}} = \frac{1/(RC)^2}{s^2 + 2s/RC + 1/(RC)^2}$$



Active filter 2

$$\frac{V_{out}}{V_{in}} = \frac{s^2}{s^2 + 2s/RC + 1/(RC)^2}$$

BASA has provided these filters to use for filtering the distortions in the image. In analysing the filter circuits, the transfer function can be taken.



Passive filter 1

In analysing the first passive filter, this would work as a bandpass. It allows high and mid frequencies pass and attenuate the low frequencies in the first mesh. After which, the high frequency is passed on to ground to only the mid frequency remains (Haykin & Van Veen, 2007, pp. 338-345).

Using circuit analysis, the transfer function of the passive filter 1 is simplified as the equation given below. $F_1(s)$ (red) and $F_2(s)$ (green) corresponds to the transfer function of a high and low pass filter respectively.

$$F_1(s) = \frac{s}{s + \frac{1}{RC}}, \qquad F_2(s) = \frac{1}{1 + RsC}, \qquad H(s) = F_1(s) \times F_2(s)$$

From there, the transfer function is reorganised to match the general form equation of a second order system as mentioned previously. The step response for the clean image is plotted using the 'lsim' function.

```
% Apply the filter to the received signal
 firstImage_filtered = lsim(Active1tf, firstImage, t);

% Reshape the filtered 1D signal back into a 2D image matrix
 reshapedFiltFirstImage = reshape(firstImage_filtered, numRows, numCols);

% Display the clean image
figure;
 imshow(reshapedFiltFirstImage);
 title('Filtered Landing Site Image');
```

By looking at the signal in the frequency domain previously, the signal of the original image is composed of lower frequencies compared to the noise. A lowpass signal filter would remove the higher frequency noise. This eliminates the use of passive filter 1 and active filter 2 as it attenuates low frequency signals, which leaves the passive filter 2 and active filter 1.

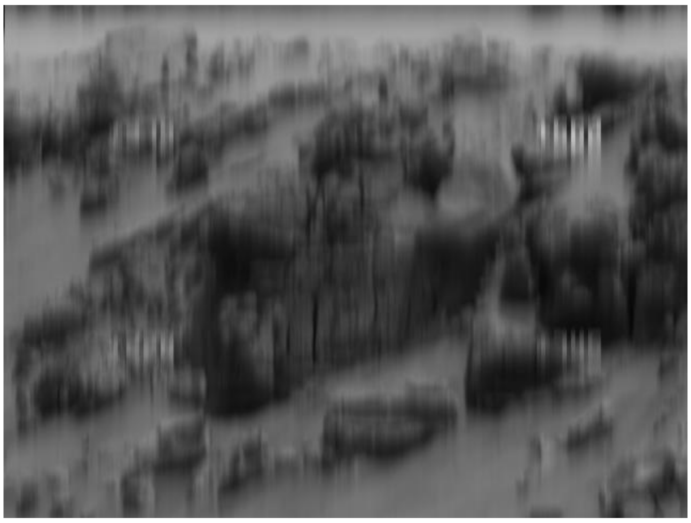


Figure 3.3: Processed image using passive filter 2

The image above is the output using passive filter 2. It still contains unwanted signals which makes the image appear blurry. Since it is a passive lowpass filter, it is only best for basic filtering tasks. Unlike active filters, passive filters have limitations in aspects such as impedance matching, flexibility, etc. Additionally, shown below is the plot for the two other filters given by BASA.

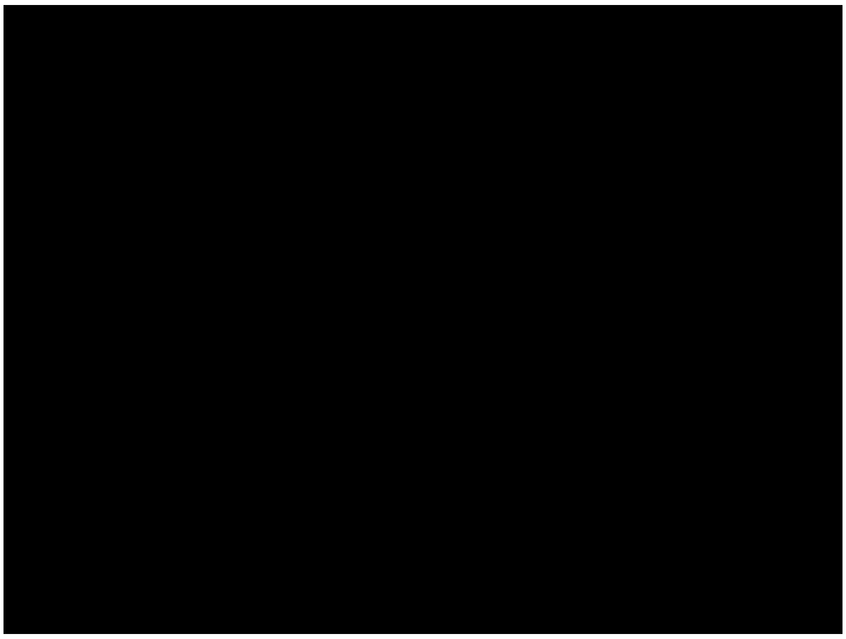


Figure 3.4: Processed image using passive filter 1 and active filter 2

This proves the earlier statement that only passive filter 2 and active filter 1 work on the signal based on the signal plot in the frequency domain. Both filters passive filter 1 and active filter 2 remove low-frequency signals which contain the image signal. An active lowpass filter is more practical in this situation compared to its passive counterpart as it is not limited to certain conditions and is more suitable for precise and complex applications (Ambardar, 1998).

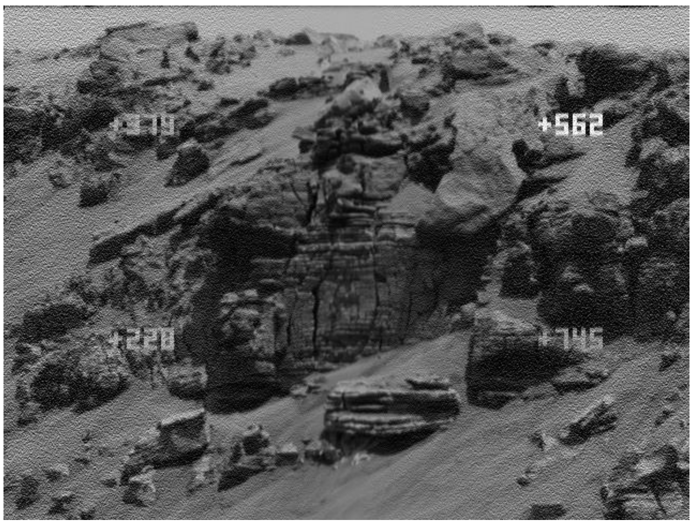


Figure 3.5: Clean 1st image using active filter 1

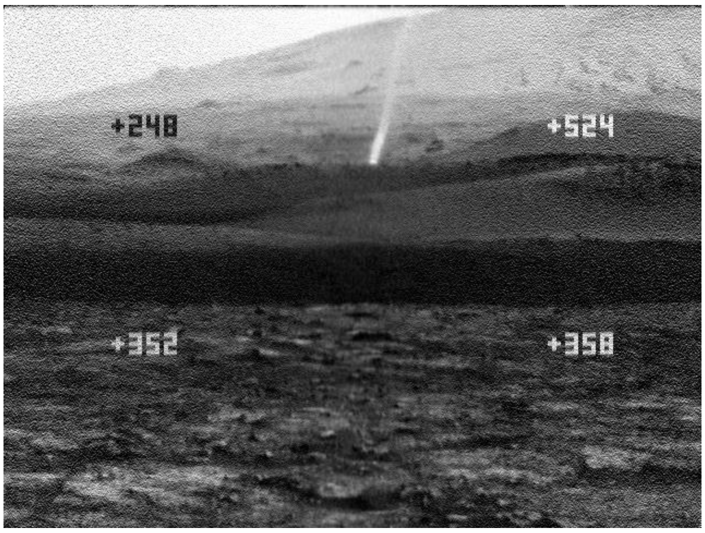


Figure 3.61: Clean 2nd image using active filter 1

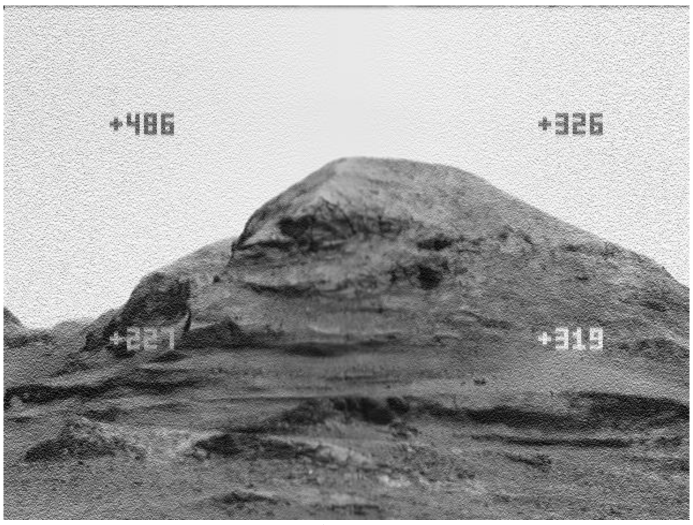


Figure 3.72: Clean 3rd image using active filter 1

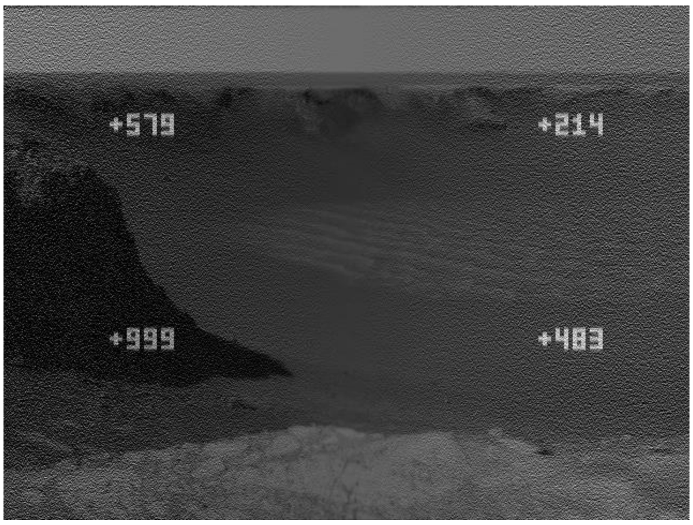


Figure 3.83: Clean 4th image using active filter 1

The figure above shows the clear images of the Mars landscape sent by the Mars rover. With this software, all the potential landing locations sent by the Mars rover can be filtered into a clear image to determine the most optimal site for the spacecraft. The areas shown in images 1 and 3 show a mountain-like landscape that would not work as a landing site. Image 2 might be a plausible area, but the rough landscape might create some difficulties upon arrival. After careful analysis, the 4thimage taken by the Mars rover shows the most optimal landing site for the MARS-242 spacecraft. Unlike the other images, it is the only image of the landscape that has a smooth surface over a large clearance.

4.0 Conclusion

To conclude, this report details the advancement of the BASA MARS-242 orbital mission's communication system. Each step of this advancement has been laid out in a logical manner, including reasonings and findings from sources within the engineering community.

It was found that as the craft leaves Earth's atmosphere, the received audio at BASA HQ becomes inaudible due to a large amount of noise. This noise was due to a distortion of the signal by the atmosphere, which has now been accounted for and removed from the signal. When this spacecraft now lands on Mars, the astronauts will send a rover ahead of time to scout the environment. The rover is equipped with a camera mounted on a servo motor, to capture panoramic images of the Mars landscape. An LTI system was designed for the mission crew to be able to control the camera's yaw angle using a voltage signal. The design behind the control system took several iterations, to produce a design which was best suited for the task. Upon successfully designing the control system for the camera pan, the rover took four images of the planet's landscape and sent it to BASA Headquarters. Based on previous experiences, signals that travel through long distances in space tend to acquire noise signals and distort the original image. BASA provided four different filters in order to clean the image signal. After processing the transfer function of the aforementioned filters, the response of the feedback system via lsim function on MATLAB shows the clean images of the Mars landscape. With these, the astronauts of the MARS-242 mission are able to determine the ideal landing spot to commence settlement on the red planet.

5.0 Reflections

5.1 Ben

I believe I have demonstrated my understanding of the concepts used in this assignment thoroughly. This has been through incorporating research from a wide variety of sources of many disciplines within the electrical engineering space, which assisted in finding the best possible. I have attempted to go through and detail each step of the process I took to come to each conclusion. All of us focused on an individual section as well, which meant that we would have a greater specific knowledge to apply to both the report and the MATLAB code. This significantly increased the quality of the entire project.

From completing this assignment, I have personally learnt some integral lessons that I will take with me for any future assessment that I am to complete. I believe that if I personally started this assessment earlier, I would have identified some hidden issues earlier, which would have allowed for a more comprehensive solution to each problem. Furthermore, this would have allowed for more time to complete other sections of the assignment with my groupmates and have enough time for all the other assessment that I've been given this semester.

5.2 Raymund

Conducting a report on applying the different concepts of signal analysis and processing in doing the assessment expanded my understanding of the unit. Through a practical approach in creating solutions for the tasks allowed me to demonstrate my capabilities and determine which areas of the course I was lacking in. With this, I have gained practical experiences through hands-on exercises. In comparison to the previous assessment that we have done in this unit, the intricacies of dealing with multiplexed audio and multiple data was undoubtedly more complex. Everyone in the group was briefed in all parts of the report but each one has a specific section that was focused. This allowed us to be more efficient in our work while also figuring out which concepts should be applied in dealing with certain problems.

We were successful in applying our previous knowledge in signal demodulation for this report and determined how to deal with multiplexed noisy audio by using two different filters. We have also dealt with analysing the properties of the transfer function of a second-order system. By doing the last section of the report, I discovered that we can also use the concepts of digital signal processing not just for communication systems but image processing as well. Reflecting upon the process of finishing this report, I enhanced my understanding of signal analysis while being able to hone my teamwork and collaboration skills. It is invaluable to be able to apply theoretical knowledge and practical experience in developing my comprehension of the unit for my future endeavours.

5.3 Zack

Understanding of concepts used throughout the report was demonstrated through justification and expansion of knowledge via research. Modelling transfer functions for the block diagram feedback system was done successfully, the behaviour of the transfer functions was analysed through the second-order characteristic equation and the system's poles. This showed my ability to come to an accurate conclusion and justify it through other means. This was reflected in all parts of my section, where no claim was left unjustified.

There were many valuable lessons learnt from working on this assignment, however, the ones that stood out to me were learning how to effectively work in a like-minded team, and the effectiveness of starting assessments early. As a group, we came together and agreed on starting the report early, as it seemed to be a common shortfall in Assessment 1. The workload was effectively distributed across members. Although we started early, the consistency of effort varied throughout the weeks, leading to times when we were not making much progress, and to times when we were cramming in work. To solve this issue, we could implement some sort of team progress-tracking mechanism, to ensure that we meet deadlines without the need for rushing.

6.0 References

Ambardar, A. (1998). *Analog and Digital Signal Processing Second Edition*. https://www.academia.edu/34431893/Analog_and_Digital_Signal_Processing_Second_Edition

Brown, M. (2011). *Power Sources and Supplies: World Class Designs*. Elsevier.

Chumbley, A. (2016). *Linear Time Invariant Systems*. Brilliant.org. https://brilliant.org/wiki/linear-time-invariant-systems/

Control Tutorials for MATLAB and SIMULINK. (n.d.). *Control Tutorials for MATLAB and Simulink - Introduction: System Analysis*. Ctms.engin.umich.edu. https://ctms.engin.umich.edu/CTMS/index.php?example=Introduction§ion=SystemAnalysis

Electronics Tutorials. (2018a, March 4). *Feedback Systems and Feedback Control Systems*. Basic Electronics Tutorials. https://www.electronics-tutorials.ws/systems/feedback-systems.html

Electronics Tutorials. (2018b, May 22). *Band Stop Filters are called Reject Filters*. Basic Electronics Tutorials. https://www.electronics-tutorials.ws/filter/band-stop-filter.html

Gonzalez, R. C., Woods, R. E., & Eddins, S. L. (2020). *Digital image processing using MATLAB* (3rd ed.). Gatesmark Publishing.

Haykin, S., & Veen, V. (2007). *Signals and systems, 2nd ed*. Wiley India Pvt. Limited. https://books.google.com.au/books?id=oQgNac7L0oIC

Keim, R. (2018, June 29). *How to Select the Cutoff Frequency of Your Low-Pass Filter*. Allaboutcircuits.com; All About Circuits. https://www.allaboutcircuits.com/technical-articles/how-to-select-the-cutoff-frequency-of-your-low-pass-filter/

Lumen Learning. (2019). *Homeostasis and Feedback Loops | Anatomy and Physiology I*. Lumenlearning.com. https://courses.lumenlearning.com/suny-ap1/chapter/homeostasis-and-feedback-loops/

Massachusetts Institute of Technology Department of Mechanical Engineering. (n.d.). *Understanding Poles and Zeros*. https://web.mit.edu/2.14/www/Handouts/PoleZero.pdf

Paradkar, S. (2024, March 25). *Crafting Robust Systems: Practices, Patterns and Principles*. Oolooroo. https://medium.com/oolooroo/beyond-resilience-the-quest-for-system-stability-in-modern-software-cee8be02f20f

Proakis, J. G. (1995). *Digital communications*. Mcgraw-Hill.

Proakis, J. G., & Manolakis, D. G. (1992). *Digital signal processing: Principles, algorithms, and applications*. Macmillan. https://books.google.com.au/books?id=ywgfAQAAIAAJ

Proakis, J. G., & Manolakis, D. G. (2007). *Digital signal processing*. Pearson/Prentice Hall.

Rossing, T. D., F Richard Moore, & Wheeler, P. (2014). *The Science of sound*. Pearson Education, Cop.

Sheldon, R., & Burke, J. (n.d.). *What is multiplexing and how does it work?* Networking. Retrieved May 26, 2024, from https://www.techtarget.com/searchnetworking/definition/multiplexing/

Smith, S. W. (1997). *The scientist and engineer's guide to digital signal processing*. California Technical Pub. https://www.dspguide.com/pdfbook.htm

Stanford University. (n.d.). *Convolution Theorem*. Ccrma.stanford.edu. https://ccrma.stanford.edu/~jos/sasp/Convolution_Theorem_DTFT.html

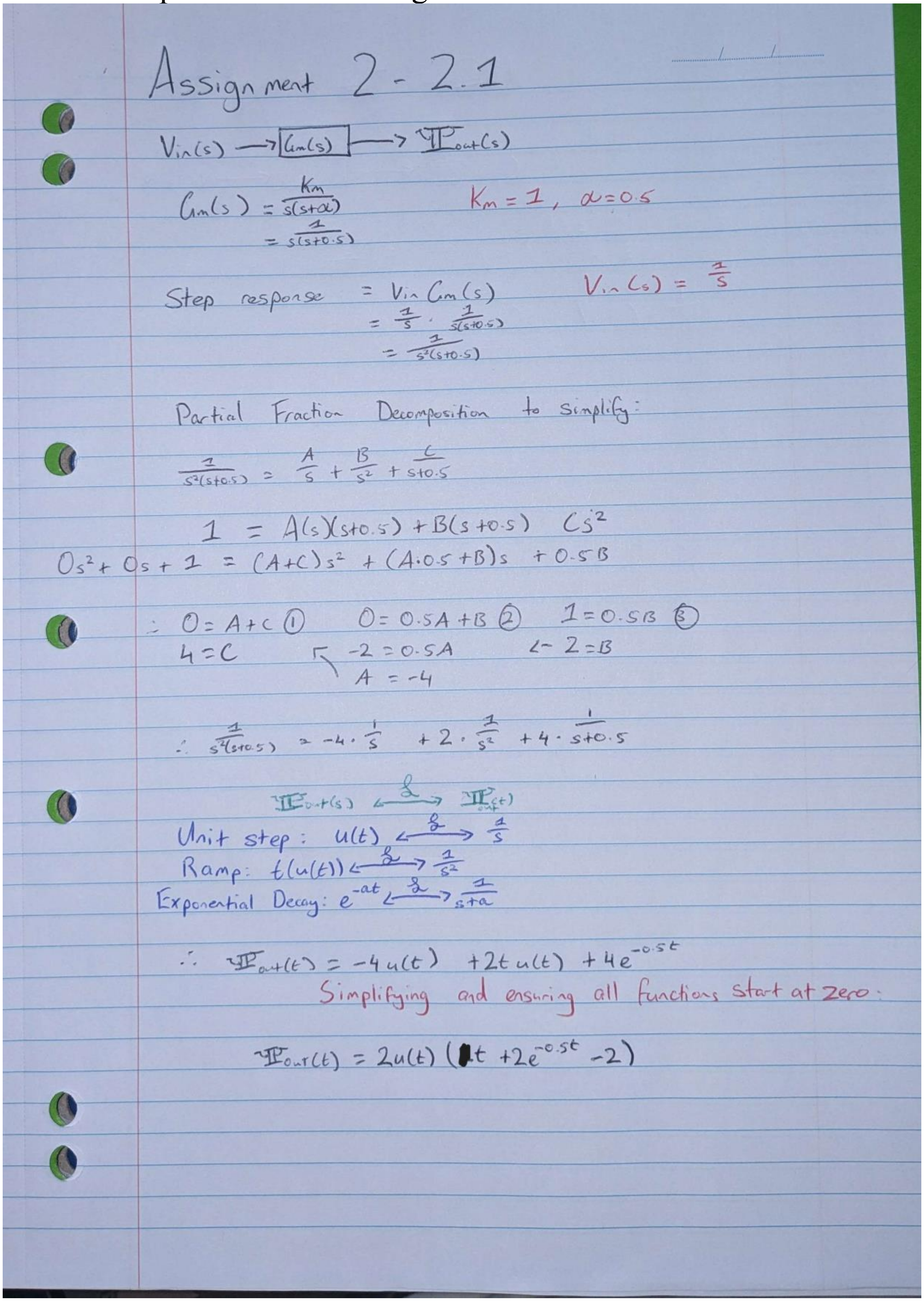
StudySmarter. (n.d.). *Step Response: Transfer Function, RC Circuit, Unit Step*. StudySmarter UK. https://www.studysmarter.co.uk/explanations/physics/electric-charge-field-and-potential/step-response/

Vedantu. (2024, May 23). *Final Value Theorem of Laplace Transform*. VEDANTU. https://www.vedantu.com/maths/final-value-theorem-of-laplace-transform

Yagle, A. (2005). *LINEAR TIME-INVARIANT SYSTEMS AND THEIR FREQUENCY RESPONSE*. University of Michigan. https://web.eecs.umich.edu/~aey/eecs206/lectures/lti2.pdf

7.0 Appendices

7.1 Laplace Hand Working for Section 2



Assignment 2 - 2.1

$$V_{in}(s) \longrightarrow \boxed{G_m(s)} \longrightarrow \Psi_{out}(s)$$

$$G_m(s) = \frac{K_m}{s(s+\alpha)} = \frac{1}{s(s+0.5)}$$ $K_m = 1,\ \alpha = 0.5$

Step response $= V_{in} G_m(s) = \frac{1}{s} \cdot \frac{1}{s(s+0.5)} = \frac{1}{s^2(s+0.5)}$ $V_{in}(s) = \frac{1}{s}$

Partial Fraction Decomposition to Simplify:

$$\frac{1}{s^2(s+0.5)} = \frac{A}{s} + \frac{B}{s^2} + \frac{C}{s+0.5}$$

$$1 = A(s)(s+0.5) + B(s+0.5) \quad Cs^2$$

$$0s^2 + 0s + 1 = (A+C)s^2 + (A \cdot 0.5 + B)s + 0.5B$$

$\therefore 0 = A + C$ ① $0 = 0.5A + B$ ② $1 = 0.5B$ ③

$4 = C$ $-2 = 0.5A$ $A = -4$ $2 = B$

$$\therefore \frac{1}{s^2(s+0.5)} = -4 \cdot \frac{1}{s} + 2 \cdot \frac{1}{s^2} + 4 \cdot \frac{1}{s+0.5}$$

$$\Psi_{out}(s) \xleftrightarrow{\mathcal{L}} \Psi_{out}(t)$$

Unit step: $u(t) \xleftrightarrow{\mathcal{L}} \frac{1}{s}$

Ramp: $t(u(t)) \xleftrightarrow{\mathcal{L}} \frac{1}{s^2}$

Exponential Decay: $e^{-at} \xleftrightarrow{\mathcal{L}} \frac{1}{s+a}$

$$\therefore \Psi_{out}(t) = -4u(t) + 2t\,u(t) + 4e^{-0.5t}$$

Simplifying and ensuring all functions start at zero:

$$\Psi_{out}(t) = 2u(t)\,(t + 2e^{-0.5t} - 2)$$

7.2 Section 1 MATLAB source code

```
%% EGB242 Assignment 2, Section 1 %%
% This file is a template for your MATLAB solution to Section 1.
%
% Before starting to write code, generate your data with the ??? as
% described in the assignment task.

%% Initialise workspace
clear all; %close all;
load DataA2 audioMultiplexNoisy fs sid;

% Begin writing your MATLAB solution below this line.

%% 1.1 Plot the time and frequency domain of audioMultiplexNoisy

% Generate appropriate time vector.
samples = length(audioMultiplexNoisy);
t = timevec(0, samples / fs, samples);


% Plot the time domain of audioMultiplexNoisy.
figure;
subplot(2, 1, 1);
plot(t, audioMultiplexNoisy, 'k');
xlabel('Time (seconds)', 'FontSize', 30);
ylabel('Amplitude', 'FontSize', 30);
title('Multiplexed Audio Signal (Time Domain)', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;

% Generate appropriate frequency vector.
samples = length(audioMultiplexNoisy);
f = freqvec(fs, samples);


% Convert audioMultiplexNoisy to frequency domain using Fourier transform.
AudioMultiplexNoisy =  ft(audioMultiplexNoisy, fs);

% Plot the frequency domain of audioMultiplexNoisy.
subplot(2, 1, 2);
plot(f, abs(AudioMultiplexNoisy), 'k');
title('Multiplexed Audio Signal (Frequency Domain)', 'FontSize', 30);
xlabel('Frequency [Hz]', 'FontSize', 30);
ylabel('Magnitude', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;
%% 1.2 Demodulate the signals in the frequency domain, listen, and plot.

% Signals probably need filtering?

% Frequencies to demodulate:
% 8.05  kHz
% 24.33 kHz
% 40.27 kHz
% 56.17 kHz
% 72.29 kHz

% First signal centred at 8.05 kHz.
carrierFrequency2 = 8050;

carrierSignal2 = cos(2*pi*carrierFrequency2*t);

firstSignalDemodulated = audioMultiplexNoisy .* carrierSignal2;

%sound(firstSignalDemodulated, fs);

FirstSignalDemodulated =  ft(firstSignalDemodulated, fs);

figure;
subplot(2, 1, 1);
plot(t, firstSignalDemodulated, 'k');
xlabel('Time (seconds)', 'FontSize', 30);
ylabel('Amplitude', 'FontSize', 30);
title('First Signal Demodulated (Time Domain)', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;

subplot(2, 1, 2);
plot(f, abs(FirstSignalDemodulated), 'k');
xlabel('Frequency [Hz]', 'FontSize', 30);
ylabel('Magnitude', 'FontSize', 30);
title('First Signal Demodulated (Frequency Domain)', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;

%% Second signal centred at 24.33 kHz
carrierFrequency2 = 24330;

carrierSignal2 = cos(2*pi*carrierFrequency2*t);

secondSignalDemodulated = audioMultiplexNoisy .* carrierSignal2;

%sound(secondSignalDemodulated, fs);

SecondSignalDemodulated =  ft(secondSignalDemodulated, fs);

figure;
subplot(2, 1, 1);
plot(t, secondSignalDemodulated, 'k');
xlabel('Time (seconds)', 'FontSize', 30);
ylabel('Amplitude', 'FontSize', 30);
title('Second Signal Demodulated (Time Domain)', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;

subplot(2, 1, 2);
plot(f, abs(SecondSignalDemodulated), 'k');
xlabel('Frequency [Hz]', 'FontSize', 30);
ylabel('Magnitude', 'FontSize', 30);
title('Second Signal Demodulated (Frequency Domain)', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;

%% Third signal centred at 40.27 kHz
```

```
carrierFrequency2 = 40270;

carrierSignal2 = cos(2*pi*carrierFrequency2*t);

thirdSignalDemodulated = audioMultiplexNoisy .* carrierSignal2;

%sound(thirdSignalDemodulated, fs);

ThirdSignalDemodulated =  ft(thirdSignalDemodulated, fs);

figure;
subplot(2, 1, 1);
plot(t, thirdSignalDemodulated, 'k');
xlabel('Time (seconds)', 'FontSize', 30);
ylabel('Amplitude', 'FontSize', 30);
title('Third Signal Demodulated (Time Domain)', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;

subplot(2, 1, 2);
plot(f, abs(ThirdSignalDemodulated), 'k');
xlabel('Frequency [Hz]', 'FontSize', 30);
ylabel('Magnitude', 'FontSize', 30);
title('Third Signal Demodulated (Frequency Domain)', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;

%% Fourth signal centred at 56.17 kHz
carrierFrequency2 = 56170;

carrierSignal2 = cos(2*pi*carrierFrequency2*t);

fourthSignalDemodulated = audioMultiplexNoisy .* carrierSignal2;

%sound(fourthSignalDemodulated, fs);

FourthSignalDemodulated =  ft(fourthSignalDemodulated, fs);

figure;
subplot(2, 1, 1);
plot(t, fourthSignalDemodulated, 'k');
xlabel('Time (seconds)', 'FontSize', 30);
ylabel('Amplitude', 'FontSize', 30);
title('Fourth Signal Demodulated (Time Domain)', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;

subplot(2, 1, 2);
plot(f, abs(FourthSignalDemodulated), 'k');
xlabel('Frequency [Hz]', 'FontSize', 30);
ylabel('Magnitude', 'FontSize', 30);
title('Fourth Signal Demodulated (Frequency Domain)', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;

%% Fifth signal centred at 72.29 kHz
carrierFrequency2 = 72290;

carrierSignal2 = cos(2*pi*carrierFrequency2*t);
```

```
fifthSignalDemodulated = audioMultiplexNoisy .* carrierSignal2;

%sound(fifthSignalDemodulated, fs);

FifthSignalDemodulated =  ft(fifthSignalDemodulated, fs);

figure;
subplot(2, 1, 1);
plot(t, fifthSignalDemodulated, 'k');
xlabel('Time (seconds)', 'FontSize', 30);
ylabel('Amplitude', 'FontSize', 30);
title('Fifth Signal Demodulated (Time Domain)', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;

subplot(2, 1, 2);
plot(f, abs(FifthSignalDemodulated), 'k');
xlabel('Frequency [Hz]', 'FontSize', 30);
ylabel('Magnitude', 'FontSize', 30);
title('Fifth Signal Demodulated (Frequency Domain)', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;

%% 1.3 Frequency and impulse response of the LTI system

% Vout(t)/Vin(t) = h(t)
% The output 'y' is the convolution of the input 'x' and the transfer function
'h'.

impulse = zeros(1, length(audioMultiplexNoisy));
impulse(1) = 1/(1/fs);
h_t = channel(sid, impulse, fs);
H_f = ft(h_t, fs);
figure;
plot(f, abs(H_f), 'r', f, abs(AudioMultiplexNoisy), 'k');
legend('H(f)', 'AudioMultiplexNoisy')
xlabel('Frequency [Hz]', 'FontSize', 30);
ylabel('Magnitude', 'FontSize', 30);
title('Frequency response of H(f) against magntude spectrum of
AudioMultiplexNoisy', 'FontSize', 30)
ax = gca;
ax.FontSize = 20;

grid on;

%% 1.4 Reverse distortion

% Input(f) = Output(f) / H(f)
AudioMultiplexReverse = AudioMultiplexNoisy ./ H_f;
audioMultiplexReverse = ift(AudioMultiplexReverse, fs);

figure;
plot(f, abs(AudioMultiplexReverse));
xlabel('Frequency [Hz]', 'FontSize', 30);
ylabel('Magnitude', 'FontSize', 30);
title('Frequency spectrum of Audio Multiplex Signal with reversed distortion',
'FontSize',30);
ax = gca;
```

```
ax.FontSize = 20;

% Demodulate signal again
carrierFrequency1 = 8050;

carrierSignal1 = cos(2*pi*carrierFrequency1*t);
demodAudioMultiplexReverse1 = audioMultiplexReverse .* carrierSignal1;

DemodAudioMultiplexReverse1 = ft(demodAudioMultiplexReverse1, fs);

%sound(demodAudioMultiplexReverse1, fs);

%%
% Demodulate signal again
carrierFrequency2 = 24330;

carrierSignal2 = cos(2*pi*carrierFrequency2*t);
demodAudioMultiplexReverse2 = audioMultiplexReverse .* carrierSignal2;

DemodAudioMultiplexReverse2 = ft(demodAudioMultiplexReverse2, fs);

%sound(DemodAudioMultiplexReverse2, fs);

%%
% Demodulate signal again
carrierFrequency3 = 40270;

carrierSignal3 = cos(2*pi*carrierFrequency3*t);
demodAudioMultiplexReverse3 = audioMultiplexReverse .* carrierSignal3;

DemodAudioMultiplexReverse3 = ft(demodAudioMultiplexReverse3, fs);

%sound(DemodAudioMultiplexReverse3, fs);

%%
% Demodulate signal again
carrierFrequency4 = 56170;

carrierSignal4 = cos(2*pi*carrierFrequency4*t);
demodAudioMultiplexReverse4 = audioMultiplexReverse .* carrierSignal4;

DemodAudioMultiplexReverse4 = ft(demodAudioMultiplexReverse4, fs);

%sound(DemodAudioMultiplexReverse4, fs);

%%
% Demodulate signal again
carrierFrequency5 = 72290;

carrierSignal5 = cos(2*pi*carrierFrequency5*t);
demodAudioMultiplexReverse5 = audioMultiplexReverse .* carrierSignal5;

DemodAudioMultiplexReverse5 = ft(demodAudioMultiplexReverse5, fs);

%sound(DemodAudioMultiplexReverse5, fs);

%% 1.4 Plots

% plot in time and frequency domains
% remove DC offset
meanValue1 = mean(demodAudioMultiplexReverse1);
demodAudioMultiplexReverse1 = demodAudioMultiplexReverse1 - meanValue1;


subplot(5,2,1);
plot(t, demodAudioMultiplexReverse1, 'b')
xlabel('Time (s)', 'FontSize', 30);
ylabel('Amplitude', 'FontSize', 30);
title('1st Audio time', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;
subplot(5,2,2);
plot(f, abs(DemodAudioMultiplexReverse1), 'r');
xlabel('Frequency [Hz]', 'FontSize', 30);
ylabel('Magnitude', 'FontSize', 30);
title('1st Audio freq', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;

meanValue2 = mean(demodAudioMultiplexReverse2);
demodAudioMultiplexReverse2 = demodAudioMultiplexReverse2 - meanValue2;
subplot(5,2,3);
plot(t, demodAudioMultiplexReverse2, 'b');
xlabel('Time (s)', 'FontSize', 30);
ylabel('Amplitude', 'FontSize', 30);
title('2nd Audio time', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;
subplot(5,2,4);
plot(f, abs(DemodAudioMultiplexReverse2), 'r');
xlabel('Frequency [Hz]', 'FontSize', 30);
ylabel('Magnitude', 'FontSize', 30);
title('2nd Audio freq', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;

meanValue3 = mean(demodAudioMultiplexReverse3);
demodAudioMultiplexReverse3 = demodAudioMultiplexReverse3 - meanValue3;
subplot(5,2,5);
plot(t, demodAudioMultiplexReverse3, 'b');
xlabel('Time (s)', 'FontSize', 30);
ylabel('Amplitude', 'FontSize', 30);
title('3rd Audio time', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;
subplot(5,2,6);
plot(f, abs(DemodAudioMultiplexReverse3), 'r');
xlabel('Frequency [Hz]', 'FontSize', 30);
ylabel('Magnitude', 'FontSize', 30);
title('3rd Audio freq', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;

meanValue4 = mean(demodAudioMultiplexReverse4);
demodAudioMultiplexReverse4 = demodAudioMultiplexReverse4 - meanValue4;
subplot(5,2,7);
```

```
plot(t, demodAudioMultiplexReverse4, 'b');
xlabel('Time (s)', 'FontSize', 30);
ylabel('Amplitude', 'FontSize', 30);
title('4th Audio time', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;
subplot(5,2,8);
plot(f, abs(DemodAudioMultiplexReverse4), 'r');
xlabel('Frequency [Hz]', 'FontSize', 30);
ylabel('Magnitude', 'FontSize', 30);
title('4th Audio freq', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;

meanValue5 = mean(demodAudioMultiplexReverse5);
demodAudioMultiplexReverse5 = demodAudioMultiplexReverse5 - meanValue5;
subplot(5,2,9);
plot(t, demodAudioMultiplexReverse5, 'b');
xlabel('Time (s)', 'FontSize', 30);
ylabel('Amplitude', 'FontSize', 30);
title('5th Audio time', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;
subplot(5,2,10);
plot(f, abs(DemodAudioMultiplexReverse5), 'r');
xlabel('Frequency [Hz]', 'FontSize', 30);
ylabel('Magnitude', 'FontSize', 30);
title('5th Audio freq', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;

%% freq spectrums together
figure;
subplot(5,1,1);
plot(f, abs(DemodAudioMultiplexReverse1), 'r');
xlabel('Frequency [Hz]', 'FontSize', 30);
ylabel('Magnitude', 'FontSize', 30);
title('1st Audio freq', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;

subplot(5,1,2);
plot(f, abs(DemodAudioMultiplexReverse2), 'r');
xlabel('Frequency [Hz]', 'FontSize', 30);
ylabel('Magnitude', 'FontSize', 30);
title('2nd Audio freq', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;

subplot(5,1,3);
plot(f, abs(DemodAudioMultiplexReverse3), 'r');
xlabel('Frequency [Hz]', 'FontSize', 30);
ylabel('Magnitude', 'FontSize', 30);
title('3rd Audio freq', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;

subplot(5,1,4);
plot(f, abs(DemodAudioMultiplexReverse4), 'r');
```

```
xlabel('Frequency [Hz]', 'FontSize', 30);
ylabel('Magnitude', 'FontSize', 30);
title('4th Audio freq', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;

subplot(5,1,5);
plot(f, abs(DemodAudioMultiplexReverse5), 'r');
xlabel('Frequency [Hz]', 'FontSize', 30);
ylabel('Magnitude', 'FontSize', 30);
title('5th Audio freq', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;

%% 1.5 Fully de-noising audio

signals = {demodAudioMultiplexReverse1, demodAudioMultiplexReverse2,
demodAudioMultiplexReverse3, demodAudioMultiplexReverse4,
demodAudioMultiplexReverse5};
newSignals = {'filteredAudioMultiplexReverse1', 'filteredAudioMultiplexReverse2',
'filteredAudioMultiplexReverse3', 'filteredAudioMultiplexReverse4',
'filteredAudioMultiplexReverse5'};
filteredSignals = struct();
fpass = 2e3;
bandfpass = [1.8e3 3e3];

for i = 1:length(signals)
    lowSignal = lowpass(signals{i}, fpass, fs, "Steepness", 0.9,
"StopbandAttenuation", 60);
    bandSignal = bandstop(lowSignal, bandfpass, fs);
    filteredSignals.(newSignals{i}) = bandSignal;
end
filteredAudioMultiplexReverse1 = filteredSignals.filteredAudioMultiplexReverse1;
filteredAudioMultiplexReverse2 = filteredSignals.filteredAudioMultiplexReverse2;
filteredAudioMultiplexReverse3 = filteredSignals.filteredAudioMultiplexReverse3;
filteredAudioMultiplexReverse4 = filteredSignals.filteredAudioMultiplexReverse4;
filteredAudioMultiplexReverse5 = filteredSignals.filteredAudioMultiplexReverse5;
%%
d = designfilt('lowpassiir', 'PassbandFrequency', fpass/(fs/2), ...
    'StopbandFrequency', (fpass+fpass*steepness)/(fs/2), ...
    'StopbandAttenuation', stopband, 'DesignMethod', 'butter');

% Visualize the filter using fvtool
h = fvtool(d);

% Set the analysis to magnitude response
h.Analysis = 'magnitude';

% Obtain the handle to the axes
ax = get(h, 'CurrentAxes');

% Set the Y-axis limits to focus on the range from 0 dB to -80 dB
ax.YLim = [-80 0];

% Optionally, limit the X-axis range, for example, from 0 to 0.5 (normalized
frequency)
ax.XLim = [0 0.5];
%%
d = designfilt('bandstopiir', 'FilterOrder', 10, ...
    'HalfPowerFrequency1', bandfpass(1), 'HalfPowerFrequency2', bandfpass(2), ...
    'DesignMethod', 'butter', 'SampleRate', fs);

% Visualize the bandstop filter using fvtool
h = fvtool(d);

% Set the analysis to magnitude response
h.Analysis = 'magnitude';

% Obtain the handle to the axes
ax = get(h, 'CurrentAxes');

% Set the Y-axis limits to focus on the range from 0 dB to -80 dB
ax.YLim = [-80 0];

% Optionally, limit the X-axis range to a specific range (e.g., from 0 to 0.5
normalized frequency)
ax.XLim = [0 0.5];
%%
% Design the lowpass filter
lowpassFilter = designfilt('lowpassiir', 'PassbandFrequency', fpass/(fs/2), ...
    'StopbandFrequency', (fpass + fpass * steepness)/(fs/2), ...
    'StopbandAttenuation', stopband, 'DesignMethod', 'butter');

% Design the bandstop filter
bandstopFilter = designfilt('bandstopiir', 'FilterOrder', 10, ...
    'HalfPowerFrequency1', bandfpass(1)/(fs/2), 'HalfPowerFrequency2',
bandfpass(2)/(fs/2), ...
    'DesignMethod', 'butter');

% Convert the digitalFilter objects to second-order sections (SOS)
lowpassSOS = ss2sos(lowpassFilter);
bandstopSOS = ss2sos(bandstopFilter);

% Create dfilt objects from SOS
lowpassDfilt = dfilt.df2sos(lowpassSOS);
bandstopDfilt = dfilt.df2sos(bandstopSOS);

% Combine the filters
combinedFilter = dfilt.cascade(lowpassDfilt, bandstopDfilt);

% Visualize the combined filter using fvtool
h = fvtool(combinedFilter);

% Set the analysis to magnitude response
h.Analysis = 'magnitude';

% Obtain the handle to the axes
ax = get(h, 'CurrentAxes');

% Set the Y-axis limits to focus on the range from 0 dB to -80 dB
ax.YLim = [-80 0];

% Optionally, limit the X-axis range to a specific range (e.g., from 0 to 0.5
normalized frequency)
ax.XLim = [0 0.5];

% Function to convert digitalFilter object to SOS matrix
function sos = ss2sos(d)
```

```
    sos = d.Coefficients;
end
%% 1.5 freq check plots
FilteredAudioMultiplexReverse1 = ft(filteredAudioMultiplexReverse1, fs);
FilteredAudioMultiplexReverse2 = ft(filteredAudioMultiplexReverse2, fs);
FilteredAudioMultiplexReverse3 = ft(filteredAudioMultiplexReverse3, fs);
FilteredAudioMultiplexReverse4 = ft(filteredAudioMultiplexReverse4, fs);
FilteredAudioMultiplexReverse5 = ft(filteredAudioMultiplexReverse5, fs);

subplot(5,2,1);
plot(t, filteredAudioMultiplexReverse1, 'b')
xlabel('Time (s)', 'FontSize', 30);
ylabel('Amplitude', 'FontSize', 30);
title('1st Audio time', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;
subplot(5,2,2);
plot(f, abs(FilteredAudioMultiplexReverse1), 'r');
xlabel('Frequency [Hz]', 'FontSize', 30);
ylabel('Magnitude', 'FontSize', 30);
title('1st Audio freq', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;

subplot(5,2,3);
plot(t, filteredAudioMultiplexReverse2, 'b');
xlabel('Time (s)', 'FontSize', 30);
ylabel('Amplitude', 'FontSize', 30);
title('2nd Audio time', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;
subplot(5,2,4);
plot(f, abs(FilteredAudioMultiplexReverse2), 'r');
xlabel('Frequency [Hz]', 'FontSize', 30);
ylabel('Magnitude', 'FontSize', 30);
title('2nd Audio freq', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;

subplot(5,2,5);
plot(t, filteredAudioMultiplexReverse3, 'b');
xlabel('Time (s)', 'FontSize', 30);
ylabel('Amplitude', 'FontSize', 30);
title('3rd Audio time', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;
subplot(5,2,6);
plot(f, abs(FilteredAudioMultiplexReverse3), 'r');
xlabel('Frequency [Hz]', 'FontSize', 30);
ylabel('Magnitude', 'FontSize', 30);
title('3rd Audio freq', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;

subplot(5,2,7);
plot(t, filteredAudioMultiplexReverse4, 'b');
xlabel('Time (s)', 'FontSize', 30);
ylabel('Amplitude', 'FontSize', 30);
title('4th Audio time', 'FontSize', 30);
```

```
ax = gca;
ax.FontSize = 20;
subplot(5,2,8);
plot(f, abs(FilteredAudioMultiplexReverse4), 'r');
xlabel('Frequency [Hz]', 'FontSize', 30);
ylabel('Magnitude', 'FontSize', 30);
title('4th Audio freq', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;

subplot(5,2,9);
plot(t, filteredAudioMultiplexReverse5, 'b');
xlabel('Time (s)', 'FontSize', 30);
ylabel('Amplitude', 'FontSize', 30);
title('5th Audio time', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;
subplot(5,2,10);
plot(f, abs(FilteredAudioMultiplexReverse5), 'r');
xlabel('Frequency [Hz]', 'FontSize', 30);
ylabel('Magnitude', 'FontSize', 30);
title('5th Audio freq', 'FontSize', 30);
ax = gca;
ax.FontSize = 20;




function t=timevec(t0, t0_plus_T, n)
% Creates time vector, where upper limit is non-inclusive
%           t0 <= t < t0_plus_T
%   It is the responsibility of the user to ensure that for the use-case
%   that they want the lower limit included, and the upper-limit
%   not included.
%
%   Args:
%   t0 = start time
%   t0_plus_T = end time (t0 + T)
%   n = number of samples

    t = linspace(t0, t0_plus_T, n + 1);
    t = t(1:end - 1);
end


function fourierTransform = ft(freq, n)
% Applies complex Fourier transform with the assigned sampling frequency
%
%   Args:
%   freq = frequency/signal input
%   n = sample frequency in Hz

    fourierTransform = fftshift(fft(freq)) / n;
end

function inversefourierTransform = ift(freq, n)
```

```
% Applies inverse Fourier transform with the assigned sampling frequency
%
%   Args:
%   freq = frequency/signal input
%   n = sample frequency in Hz

    inversefourierTransform = ifft(ifftshift(freq)) * n;
end


function f=freqvec(fs, n)

    % if is an even sequence length, generating the frequency vector
    % is just like doing it for our time vector in the timevec function
    if mod(n, 2) == 0
        f_str = sprintf('Generating freq vec\n [%.2f, %.2f)\n', -fs/2, fs/2);
        disp(f_str);
        % compute the frequency vector
        f = linspace(-fs / 2, fs / 2, n + 1);
        f = f(1:end - 1);
    % otherwise is of odd length
    else
        f_str = sprintf('Generating freq vec\n (%.2f, %.2f)\n', ...
            -(n -1)/2 * fs / n, (n -1)/2 * fs / n);
        disp(f_str);
        % compute the frequency vector
        f = linspace(-(n -1)/ 2, (n -1) / 2, n) * fs / n;
    end
end
```

7.3 Section 2 MATLAB source code

```
%% EGB242 Assignment 2, Section 2 %%
% This file is a template for your MATLAB solution to Section 2.
%
% Before starting to write code, generate your data with the ??? as
% described in the assignment task.

%% Initialise workspace
clear all; close all;

%% 2.1

t2 = timevec(0, 20, 100000); % Time vector

% Step input
step_input = ones(size(t2));

% Step response
step_response = 2*t2 + 4*exp(-0.5*t2) - 4;

% Plotting
figure;
subplot(2,1,1);
plot(t2, step_input, 'b', 'LineWidth', 1.5); % Step input
xlabel('Time (s)');
ylabel('Voltage (V)');
```

```
title('Step Input');
grid on;

subplot(2,1,2);
plot(t2, step_response, 'r', 'LineWidth', 1.5); % Step response
xlabel('Time (s)');
ylabel('Radians (rad)');
title('Step Response');
grid on;

%% 2.1.1 Poles

poles = [-0.5, 0];

figure;

plot(real(poles), imag(poles), 'kx', 'MarkerSize', 10, 'LineWidth', 2);
hold on;

grid on;

xlim([-1, 1]);
ylim([-1, 1]);

line([-1, 1], [0, 0], 'Color', 'k', 'LineStyle', '-'); % Real axis
line([0, 0], [-1, 1], 'Color', 'k', 'LineStyle', '-'); % Imaginary axis

xlabel('Real Part');
ylabel('Imaginary Part');

title('Pole-Zero Plot');

text(real(poles(1)), imag(poles(1)), '  -0.5', 'VerticalAlignment', 'bottom',
'HorizontalAlignment', 'right');
text(real(poles(2)), imag(poles(2)), '  0', 'VerticalAlignment', 'bottom',
'HorizontalAlignment', 'right');

legend('Poles');

hold off;

%% 2.2

% Define transfer function G1
numerator_G1 = [1];
denominator_G1 = [1, 0.5, 1];
G1 = tf(numerator_G1, denominator_G1);

% Use lsim to simulate the response of the feedback system
y1 = lsim(G1, step_input, t2);

% Plot the step response
figure;
plot(t2, y1);
title('Step Response of Feedback System');
xlabel('Time (s)');
ylabel('Output');
grid on;
```

```matlab
% This isn't good enough because it doesn't rotate all the way.

%% 2.3

% Underdamped
% Natural frequency (omega-n): 1
% Damping ratio (zeta): 0.25

% Peak time: 3.24s
% Settling time: 16s
% Overshoot %: 44.4%

%% 2.4

% Define the values for Ggs and Hgs
Ggs = [0.1, 0.2, 0.5, 1, 2];
Hgs = [0.1, 0.2, 0.5, 1, 2];

% Define step input and time vector t2
step_input = ones(100, 1); % Assuming a unit step input
t2 = linspace(0, 10, 100); % Assuming a time vector from 0 to 10 seconds

% Create a new figure
figure;

% Loop over the values in Ggs while keeping Hgs constant
hold on; % Keep all plots on the same graph
for i = 1:5
    Ggs_i = 1;
    num = Ggs_i;
    den = [1, 0.5, Hgs(i)];
    G2 = tf(num, den);
    y = lsim(G2, step_input, t2);
    plot(t2, y, 'DisplayName', ['Kfb = ', num2str(Hgs(i))]);
end

% Add labels, title, and legend
xlabel('Time (s)');
ylabel('Output');
title('Step Response for Different Kfb Values');
legend;
grid on; % Turn on the grid
hold off;

% Create a new figure for the second set of plots
figure;

% Loop over the values in Hgs while keeping Ggs constant
hold on; % Keep all plots on the same graph
for i = 1:5
    Hgs_i = 1;
    num2 = Hgs(i);
    den2 = [1, 0.5, Ggs(i)];
    G2 = tf(num2, den2);
    y = lsim(G2, step_input, t2);
    plot(t2, y, 'DisplayName', ['Kfwd = ', num2str(Ggs(i))]);
end

% Add labels, title, and legend
```

```
xlabel('Time (s)');
ylabel('Output');
title('Step Response for Different Kfwd Values');
legend;
grid on; % Turn on the grid
hold off;

%% 2.5

% Get the system parameters (2.3) of all the plots created in 2.4, using
% this determine which gain values (Hg(s) and Gg(s)) is suitable. Requires
% of the system:

% - Output must accurately travel between 0 to 2pi rads (so preferrably no
% oscillation).
% - The camera shouldn't rotate to quickly (peak time shouldn't be too
% fast, aim for 13 seconds).

% Define transfer function G1
Fnum = [0.7596]; % 2pi * Wn^2
Fden = [1, 0.5, 0.1208]; % G(s) General second order transfer function denominator
cameraTF = tf(Fnum, Fden);

% Use lsim to simulate the response of the feedback system
y2 = lsim(cameraTF, step_input, t2);

% Plot the step response
figure;
plot(t2, y2);
title('Step Response of Feedback System');
xlabel('Time (s)');
ylabel('Output (rad)');
grid on;

%% 2.6
% Output = 30 deg = pi/6
% Output = 210 deg = 7pi/6
% Input * 2pi = Output ~ Input = Output / 2pi

startVoltage = 1/12;
endVoltage = 7/12;
[startIm, finalIm] = cameraPan(startVoltage, endVoltage, cameraTF);
%% helper functions

function t=timevec(t0, t0_plus_T, n)
% Creates time vector, where upper limit is non-inclusive
%           t0 <= t < t0_plus_T
%   It is the responsibility of the user to ensure that for the use-case
%   that they want the lower limit included, and the upper-limit
%   not included.
%
%   Args:
%   t0 = start time
%   t0_plus_T = end time (t0 + T)
%   n = number of samples

    t = linspace(t0, t0_plus_T, n + 1);
    t = t(1:end - 1);
end
```

```
function fourierTransform = ft(freq, n)
% Applies complex Fourier transform with the assigned sampling frequency
%
%   Args:
%   freq = frequency/signal input
%   n = sample frequency in Hz

    fourierTransform = fftshift(fft(freq)) / n;
end

function inversefourierTransform = ift(freq, n)
% Applies inverse Fourier transform with the assigned sampling frequency
%
%   Args:
%   freq = frequency/signal input
%   n = sample frequency in Hz

    inversefourierTransform = ifft(ifftshift(freq)) * n;
end


function f=freqvec(fs, n)

    % if is an even sequence length, generating the frequency vector
    % is just like doing it for our time vector in the timevec function
    if mod(n, 2) == 0
        f_str = sprintf('Generating freq vec\n [%.2f, %.2f)\n', -fs/2, fs/2);
        disp(f_str);
        % compute the frequency vector
        f = linspace(-fs / 2, fs / 2, n + 1);
        f = f(1:end - 1);
    % otherwise is of odd length
    else
        f_str = sprintf('Generating freq vec\n (%.2f, %.2f)\n', ...
            -(n -1)/2 * fs / n, (n -1)/2 * fs / n);
        disp(f_str);
        % compute the frequency vector
        f = linspace(-(n -1)/ 2, (n -1) / 2, n) * fs / n;
    end
end
```

7.4 Section 3 MATLAB source code

```
%%%% EGB242 Assignment 2, Section 3 %%
% This file is a template for your MATLAB solution to Section 3.
%
% Before starting to write code, generate your data with the ??? as
% described in the assignment task.

%% Initialise workspace
clear all; close all; clc;
load DataA2 imagesReceived;

% Begin writing your MATLAB solution below this line.
%% Question 3.1
```

```
firstImage = imagesReceived(1,:);

numRows = 480;
numCols = 640;

% Converting a received pixel stream to an image matrix
reshapedFirstImage = reshape(firstImage, numRows, numCols);

% Displaying an image in a figure
figure;
imshow(reshapedFirstImage);

% Saving an image matrix as an image file
imwrite(reshapedFirstImage, '1stImage.png');


%% Question 3.2

% Define the parameters
pixelRate = 1000; % pixels per second
numPixels = numel(firstImage); % returns the number of elements in an array
(samples).
duration = numPixels / pixelRate; % total duration of the signal

% Construct the time vector
t = timevec(0, duration, numPixels);

% Construct the frequency vector
Fs = pixelRate; % Sampling frequency
f = freqvec(Fs, numPixels);

% Visualize the received signal in time domain
figure;
plot(t, firstImage);
xlim([0,307]);
ylim([-7,7]);
xlabel('Time [s]');
ylabel('Amplitude');
title('Received image signal in Time Domain');

% Visualize the received signal in frequency domain
FirstImage = ft(firstImage);
figure;
plot(f, abs(FirstImage));
title('Received Signal in Frequency Domain');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

%% Question 3.3

% Define transfer function Filter 1
Pnum1 = [1.2e-3];
Pden1 = [5.64e-5, 5.9e-3, 1];
Passive1tf = tf(Pnum1, Pden1);

% Define transfer function Filter 2
Pnum2 = [1];
Pden2 = [5.64e-5, 16.7e-3, 1];
Passive2tf = tf(Pnum2, Pden2);
```

```
% Define transfer function Filter 1
Anum1 = [1.49e6];
Aden1 = [1, 2.44e3,1.49e6];
Active1tf = tf(Anum1, Aden1);

% Define transfer function Filter 2
Anum2 = [1];
Aden2 = [1, 2.44e3, 1.49e6];
Active2tf = tf(Anum2, Aden2);

% %% LTI view
% ltiview(Passive1tf);
% ltiview(Passive2tf);
% ltiview(Active1tf);
% ltiview(Active2tf);

%% Use lsim to simulate the response of the feedback system
% P1 = lsim(Passive1tf, firstImage, t);
% P2 = lsim(Passive2tf, firstImage, t);
% A1 = lsim(Active1tf, firstImage, t);
% A2 = lsim(Active2tf, firstImage, t);

%% All images are best cleaned with Active filter 1.
%% 3.4
% First image

% Apply the filter to the received signal
 firstImage_filtered = lsim(Active1tf, firstImage, t);

% Reshape the filtered 1D signal back into a 2D image matrix
 reshapedFiltFirstImage = reshape(firstImage_filtered, numRows, numCols);

% Display the clean image
figure;
 imshow(reshapedFiltFirstImage);
 title('Filtered Landing Site Image');

% Visualize the clean signal in time domain
figure;
plot(t, firstImage_filtered);
xlim([0,307]);
ylim([-2,2]);
title('Filtered Signal in Time Domain');
xlabel('Time (s)');
ylabel('Pixel Intensity');

% Visualize the clean signal in frequency domain
FirstImage_filtered = ft(firstImage_filtered);
figure;
plot(f, abs(FirstImage_filtered));
title('Filtered Signal in Frequency Domain');
xlabel('Frequency (Hz)');
ylabel('Magnitude');


%% 3.5
for i = 2:4
    image = imagesReceived(i, :);
    reshapedImage = reshape(image, numRows, numCols);
    imgString = [num2str(i) 'image.png'];
    imwrite(reshapedImage, imgString);

    figure;
    imshow(reshapedImage);

    image_filtered = lsim(Active1tf, image, t);

    reshapedFiltImage = reshape(image_filtered, numRows, numCols);

    figure;
    imshow(reshapedFiltImage);
    imgTitleString = ['Filtered Landing Site Image' num2str(i)];
    title(imgTitleString);

    figure;
    plot(t, image_filtered);
    xlim([0,307]);
    ylim([-2,2]);
    title('Filtered Signal in Time Domain');
    xlabel('Time (s)');
    ylabel('Pixel Intensity');

    Image_filteredP1 = ft(image_filtered);
    figure;
    plot(f, abs(Image_filteredP1));
    title('Filtered Signal in Frequency Domain');
    xlabel('Frequency (Hz)');
    ylabel('Magnitude');
end



%% helper functions


function t=timevec(t0, t0_plus_T, n)
% Creates time vector, where upper limit is non-inclusive
%          t0 <= t < t0_plus_T
%   It is the responsibility of the user to ensure that for the use-case
%   that they want the lower limit included, and the upper-limit
%   not included.
%
%   Args:
%   t0 = start time
%   t0_plus_T = end time (t0 + T)
%   n = number of samples

    t = linspace(t0, t0_plus_T, n + 1);
    t = t(1:end - 1);
end

function f=freqvec(fs, n)

    % if is an even sequence length, generating the frequency vector
    % is just like doing it for our time vector in the timevec function
    if mod(n, 2) == 0
        f_str = sprintf('Generating freq vec\n [%.2f, %.2f)\n', -fs/2, fs/2);
        disp(f_str);
        % compute the frequency vector
        f = linspace(-fs / 2, fs / 2, n + 1);
        f = f(1:end - 1);
    % otherwise is of odd length
    else
        f_str = sprintf('Generating freq vec\n (%.2f, %.2f)\n', ...
            -(n -1)/2 * fs / n, (n -1)/2 * fs / n);
        disp(f_str);
        % compute the frequency vector
        f = linspace(-(n -1)/ 2, (n -1) / 2, n) * fs / n;
    end
end

function fourierTransform = ft(freq)
% Applies complex Fourier transform with the assigned sampling frequency
%
%   Args:
%   freq = frequency/signal input
%   n = sample frequency in Hz

    fourierTransform = fftshift(fft(freq));
end
```