Signal Analysis & Image
Processing

Designing a FIR filter
for Time Variant Signals & Detecting boundary pixels in a Plant Imagery Dataset

1 Signal Processing

1.1 The dataset

The three discrete signals given seem to be part of a song, with the third being the former two songs combined. Signals are sampled at 1380hz and 11025hz.

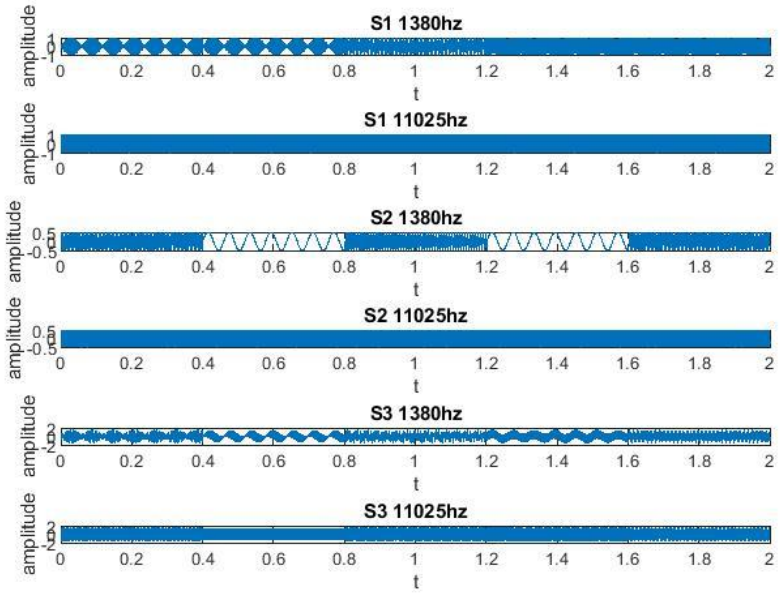

Figure 1 First 2 Seconds of each signal plotted.

After plotting the first two seconds, the second signal may be periodic and repeat every 0.8 seconds, but in this time span the other signals appear non-periodic. Visually, it is more difficult to check if the first and second signal at 11025hz equals the third signal at that frequency, but it can be checked via matlab's isequal() function, and returns 'True'. Another method is checking the correlation of signal 3 and the sum of 1 and 2. The plot is symmetrical, indicating autocorrelation, which could be used to spot repeating musical beats.

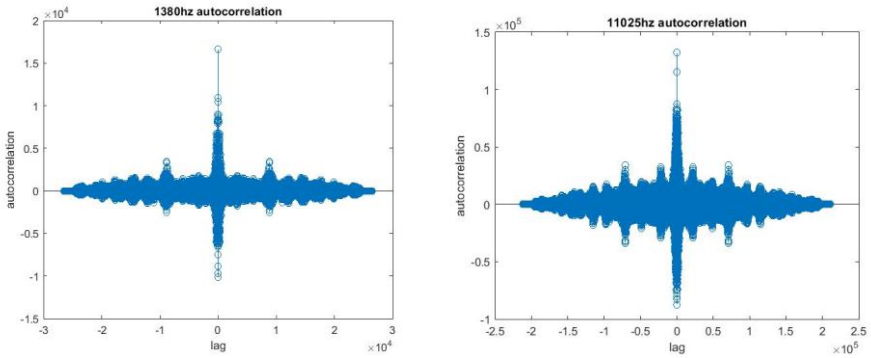

Figure 2 Correlation of signal 1+2 and 3, repeated for each sampling hz.

1.2 Fast Fourier Transform

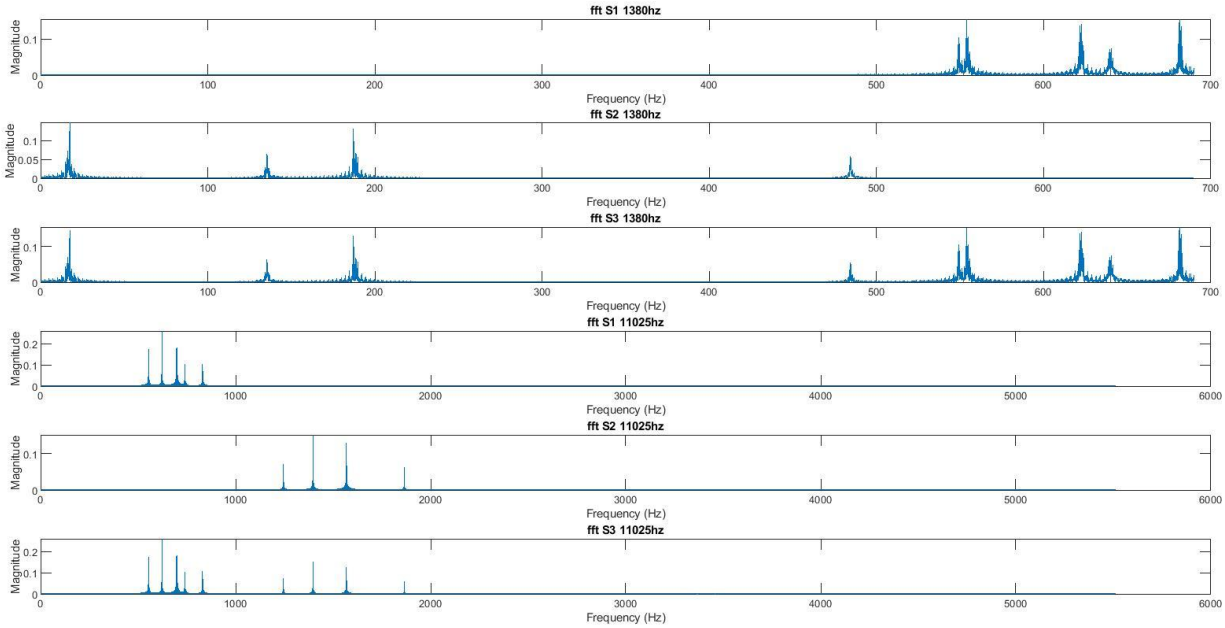

Figure 3 Fast Fourier Transformed signals.

Each signal was transformed using Fast Fourier Transform, which converts them into their sinusoidal frequency elements, plotting them in the frequency domain (fig3) as opposed to time. Regarding the 1380hz signals, for the first signal at higher frequencies (over 500hz) there are periods of high magnitude. The second signal appears to have periods of high magnitude at lower frequencies (less than 200hz), with an outlier at approximately 470hz. It is very clear that the third signal is the first and second combined on examination of these plots.

For the signals sampled at 11025hz, the first signals frequency components do not pass 1000hz, while the second signals remain over 1000hz. The magnitude of these peaks is much higher than those of the 1380hz signal, indicating that the signal is much noisier/distorted.

1.3 Short Time Fourier Transform

The short-time Fourier transform (STFT) is used to analyse how the frequency content of a nonstationary signal changes over time (MathWorks 2022). It performs a Discrete Fourier Transform over many time periods of the signal 'windows' with a certain overlap.

The stft is performed on all signals. Firstly the frequencies are plotted over different window sizes(fig 4,5, appendix A,B). It appears that the larger the window size (4096), the closer the result visually resembles a total Fast Fourier Transform. Smaller window sizes (512) show wider magnitude peaks, indicating that more varied distortion can be captured which can help identify separate signals more uniquely than the previous FFT method. The stft method comes with a forfeit of a longer computation runtime as the smaller the window size the more DFT calculations are needed to be performed.

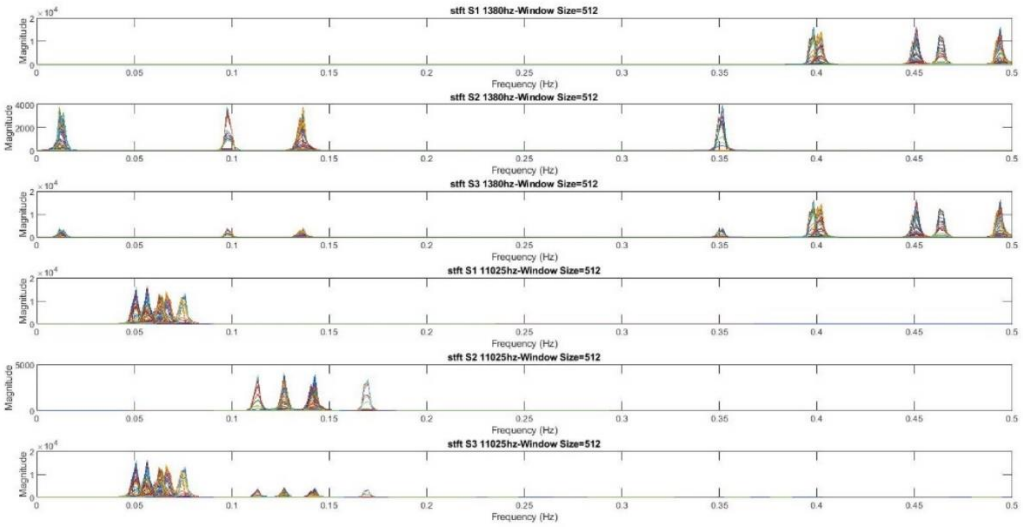

Figure 4 STFT'd signals (Window size 512).

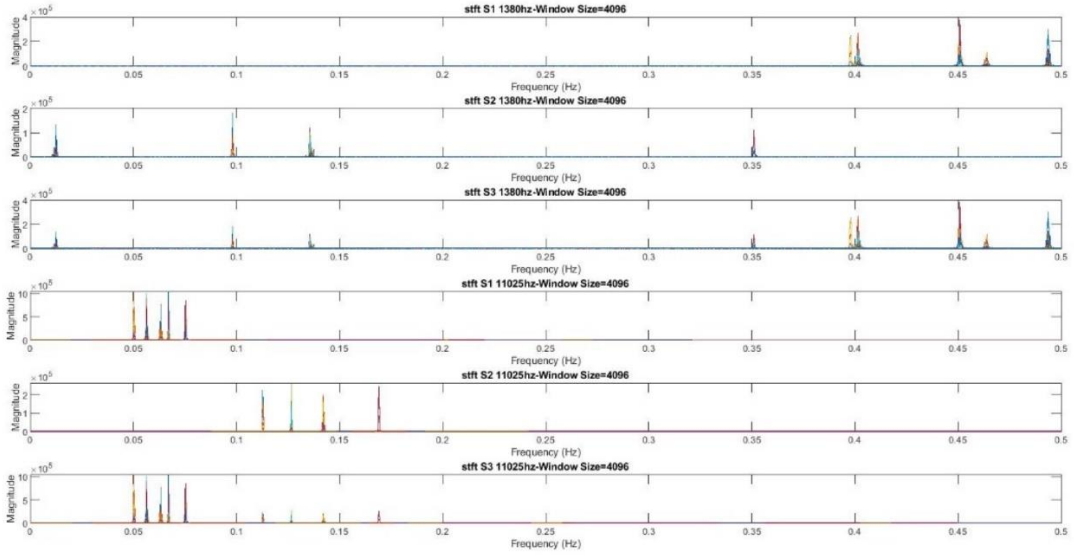

Figure 5 STFT'd signals (Window size 4096).

The spectrograms are also plotted. To do so, the magnitude squared of the STFT requires calculation.

On spectrogram analysis, for the 1380Hz signals, the first signal's frequencies vary less than the second signal and remain tightly packed around the same 0.3-0.4 values. The first signal's frequencies are higher than that of the second, while also having its maximum magnitude double that of the

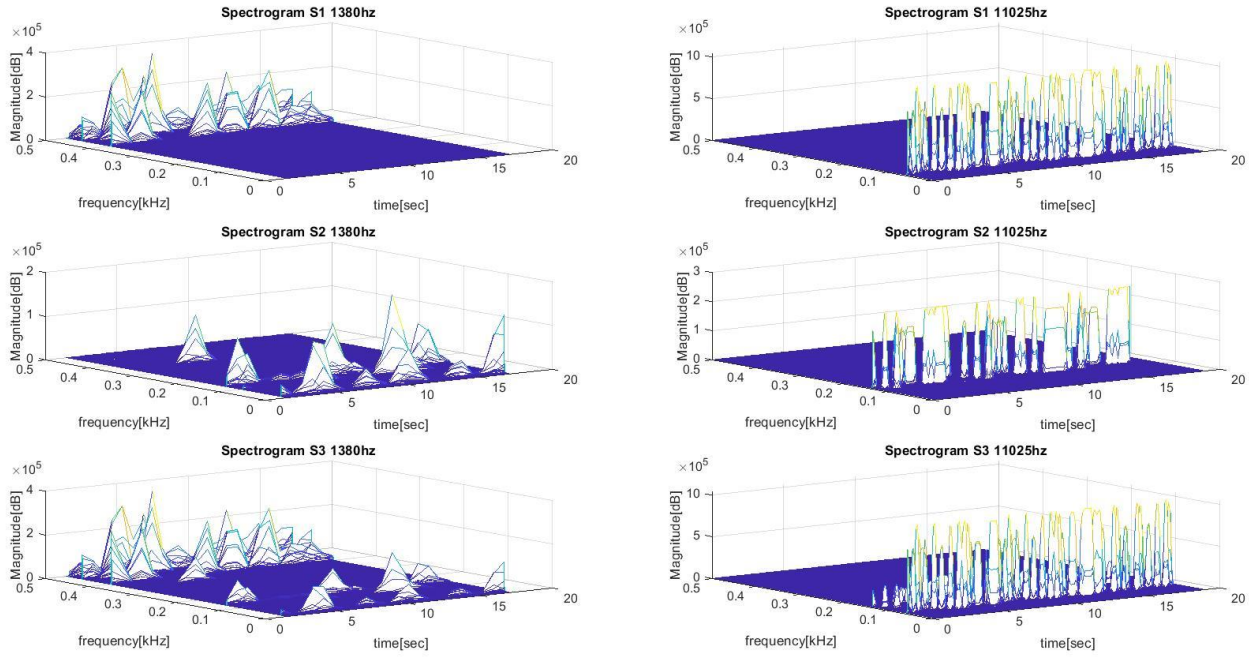

Figure 6 Signal Spectrograms

highest second signal magnitude. The magnitudes of the first signals peaks appear higher nearing the beginning of the first signal, whilst they are higher nearing the end of the second signal.

For the 11025Hz signals, it is more difficult to differentiate between them initially. The frequencies of the second signal appear less than 0.1 higher than the first signal. Like the 1380Hz, the highest peak of the first signal is almost double that of the second signal. Unlike the 1380Hz, the higher Hz signals seem to remain uniform in magnitude over time. It is easy to see the difference in the total magnitude between the two signals in signal three however, compared to signal three of 1380Hz, which is more blended.

1.4 Continuous Wavelet Transform

A continuous wavelet transform was performed on each signal and the frequency/time relationship was plotted in a scalogram (fig 8, appendix C, D).

Three filter banks were created, with bump, amor, and morse wavelets respectively. All 3 filter banks showed similarities with their frequency resolution, with the 'morse' wavelet showing slightly more differentiation between signals on production of scalograms.

On analysis, for the 1380Hz, the second signal's frequencies vary erratically over time compared to the first signal, where the frequency varies uniformly over the same period. The second signal's

remain lower in general than the first signal, however. The elements from the first signal can be seen in signal three above that of the second signal.

For the signals of 11025Hz, it is easy to see that the second signal has slightly fluctuating higher frequencies, than the first of slightly fluctuating lower frequencies. They only vary slightly over time. There is a clear difference between them both in the third signal, where the scalogram shows the unique elements of signal one and signal two in parallel to each other, with ‘two’ lying above ‘one’. It appears that continuous wavelet transform can localise frequency resolutions of high frequency signals better than stft can, in seeing the difference in readability between stft’s spectrogram and cwt’s scalogram.

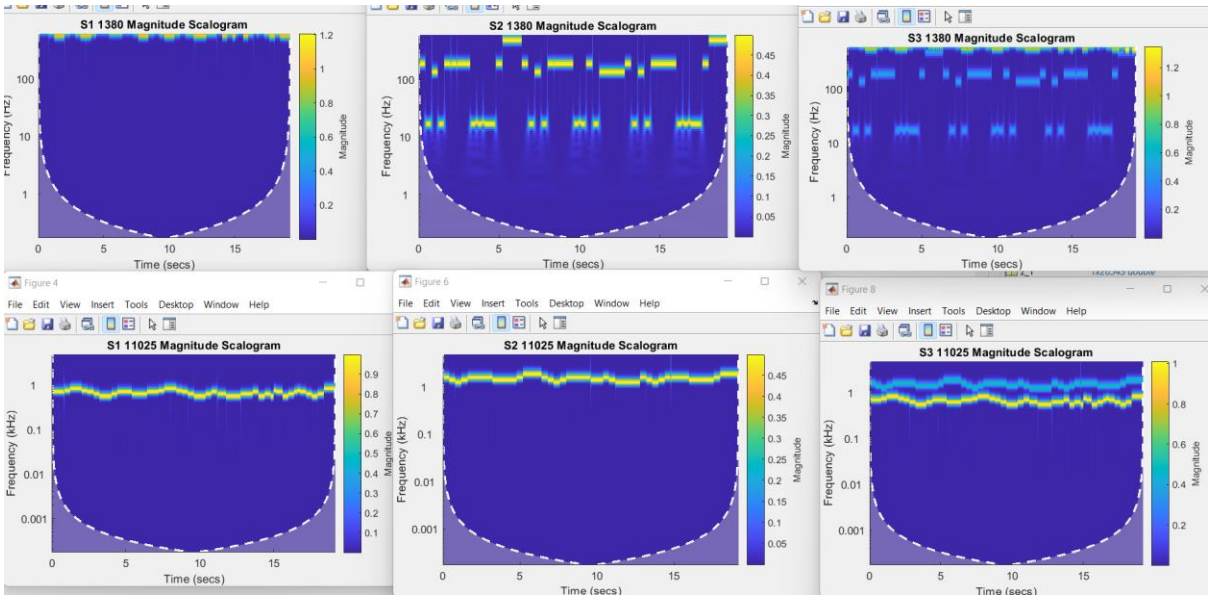

Figure 7 Scalogram of signals after 'Morse' Continuous Wavelet Transform.

1.5 Finite Impulse Response (FIR) Filter

Two FIR filters were created, one for the 1380Hz signals and one for the 11025Hz signals.

1.5.1 1380Hz

For the 1380Hz signals, a lowpass filter was created. This cut-off the frequencies over 470hz.

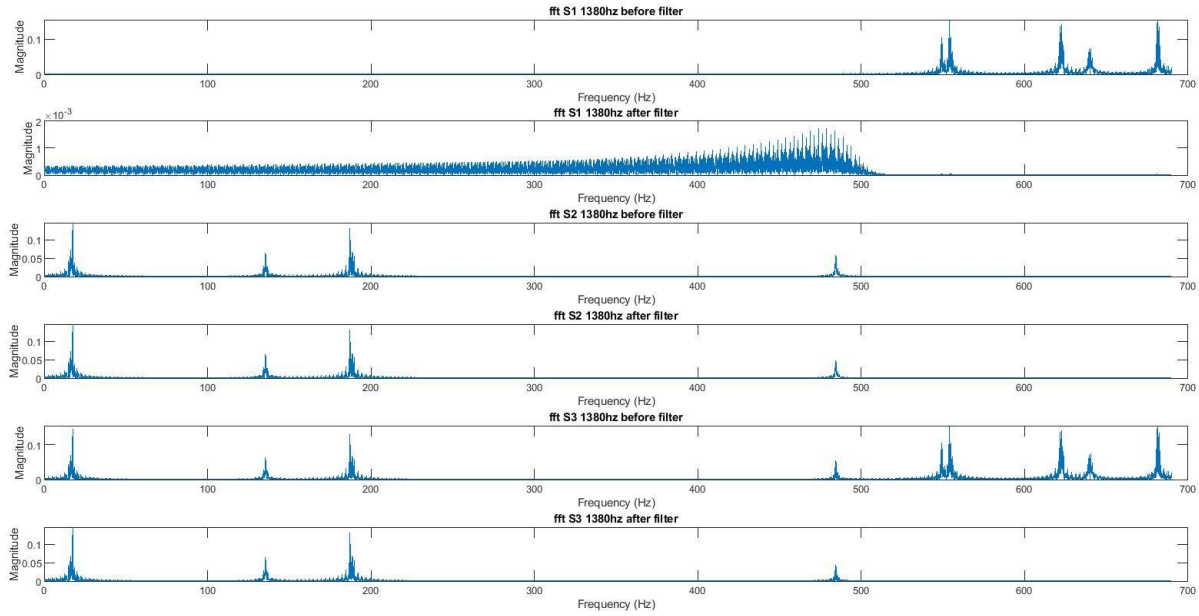

Figure 8 Figure showing the FFT transformed 1380Hz-sampled signals before and after applying FIR lowpass filter.

The steepness was adjusted to 0.8 to preserve the 'outlier' higher frequency in the second signal while removing those in the first signal. As can be seen in the FFT plot (fig 9), the filter applied to the second signal leaves it unchanged while the elements of the first signal are removed from the third signal. There is a small frequency remaining from the first signal, but it is negligible at the scale of 1x10-3. However, it is interesting to see the effect of the steepness on this negligible passing frequency range (filter is more sensitive at higher steepness, shown by the visual slope).

1.5.2 11025Hz

For the 11025Hz sampled signals, a high-pass filter was created (fig 10). This cut-off the frequencies under 1200hz. The steepness was adjusted to 0.8 to preserve the lower frequencies in the second signal.

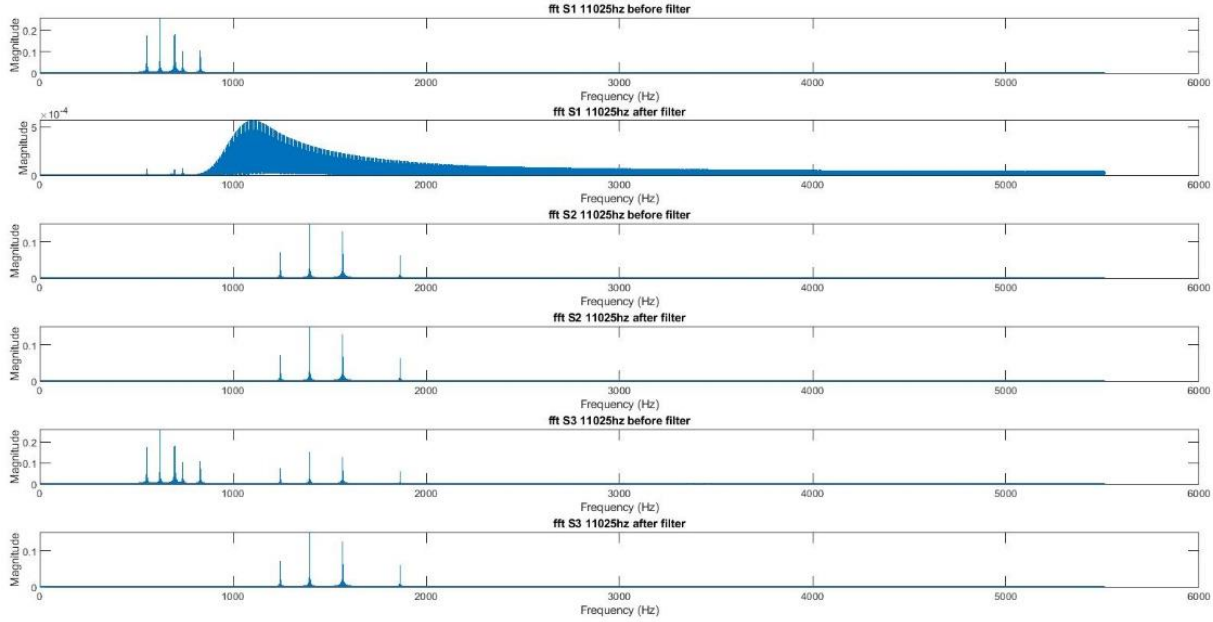

Figure 9 Figure showing the FFT transformed 11025Hz-sampled signals before and after applying FIR high pass filter.

2 Image Processing

The task was to develop an image processing pipeline to detect boundary pixels of plants. Four experimental pipelines were assessed.

2.1 Method 1: Canny/ Morphological Operations

The first technique was the use of contrast adjustment, filtering, and morphological operations with canny edge detection, which originally consisted of a trial-and-error approach.

The histogram of the greyscale image was analysed. It was found that focusing on the higher pixel values with a broad range increased the contrast of the plant region, whilst with a narrow range increased the contrast of the leaves themselves. A gaussian filter was used to reduce the impact of the noise brought into show from around the plant.

As the contrast adjustment was used to bring focus, binarization was performed using a high adaptive threshold (0.9) to bring these high contrast pixels to ‘white’ foreground binary values.

Smaller connected components (>70 pixels) were removed as these were generally noise objects that made their way through the binarization stage. The image was ‘closed’ such that leaves that became ellipses due to processing became more naturally oval-like (using a disk structural element of radius ‘2’). Any small holes in them were also filled.

Edges were found using the canny filter. Visually the detection seemed reasonable but is sensitive to noise connected to the leaf edges.

To get the connected components between the leaves and stalks/others, the binary leaf image was subtracted from the binary full plant image. Unfortunately, discrepancies in edges size (due to morphological operations), meant it is not a clean subtraction, and some outer leaf edges themselves are left. Eroding does not solve this, as the thin edges are sensitive to this operation and will remove all from the image.

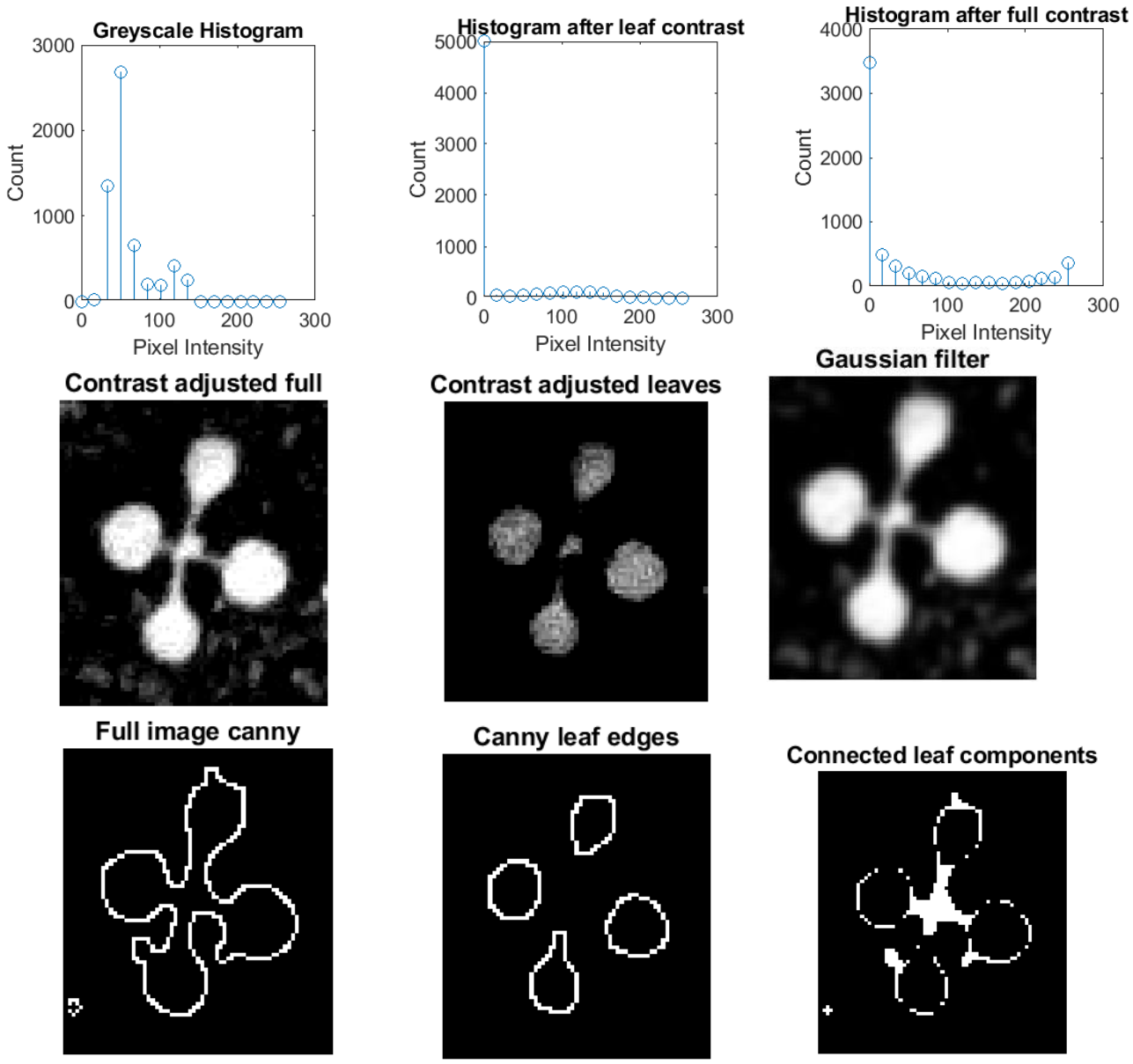

Figure 10 Showing histograms and outputs at various stages of the 'canny' pipeline.

2.2 Method 2: Fourier Transform

The second technique was the use of Fast Fourier Transform on the image to get the boundaries of the plant. The image was first brought into greyscale, contrast-adjusted and binarized in similar fashion to the last technique. A 2D Fast Fourier Transform was performed and visualised. It appeared that a high-pass filter with a cut-off frequency of 10 adequately started to show boundaries. To highlight them, the image was binarized once more. FFT seems to need that extra step and may not be practical

due to the addition of further noise that becomes evident on that last binarization (the new ‘cloudy’ image noise becomes sharp).

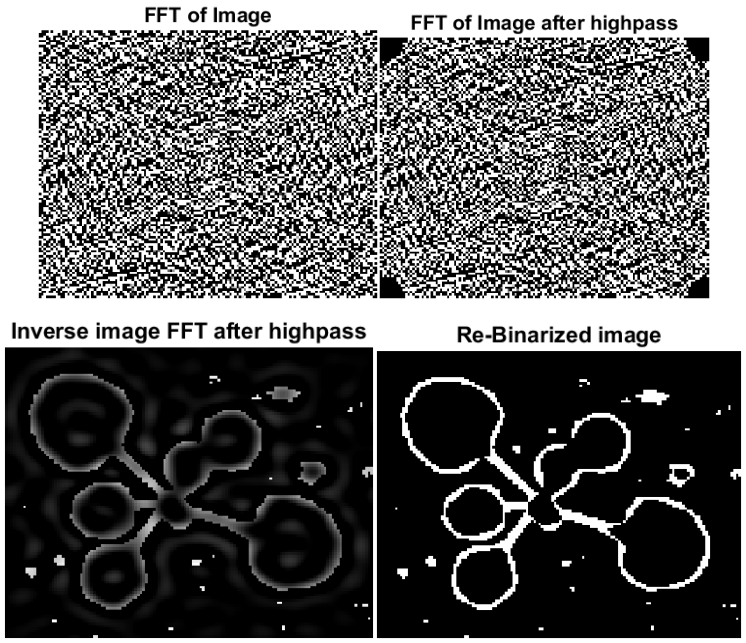

Figure 11 Using 2D Fast Fourier Transform to create a plant boundary.

2.3 Method 3: Hough Transform (Circles)

The third technique assessed was that of the Hough transform, in an attempt the find the leaf ‘circles’ to separate them. The image was converted to greyscale, contrast-adjusted, binarized and cleared of small components yet again. The optimum radii for leaves in the image appeared to be between 10 and 30 pixels. The transform was run with a sensitivity of 0.94, hoping to pick up circles that were not full (as leaves are not usually). The circles did seem to find the leaves, however, was subject visually to false positives. Furthermore, on analysis of the circles themselves, they tended to overlap, meaning that separating them from each other would be just as challenging as separating the leaves from each other in the first instance, if more so.

A more viable solution could be an analysis of a more general Hough transform which uses the spade-like shape of leaves as a mask. The challenge would still stand however on capturing the variety of plant leaves in their entirety.

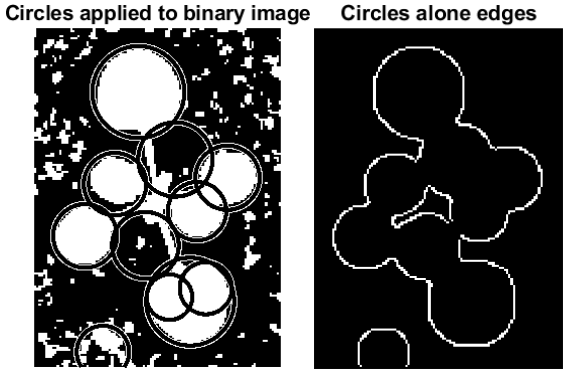

Figure 12 Circle Hough Transform performed on the plant dataset.

2.4 Method 4: Erosion subtraction

The final method began with converting the image to greyscale, adjusting the contrast and binarizing using an adaptive threshold. Small components were removed yet again, and small holes were filled. Items around the border were removed. The image was binarized.

The image was eroded with a very small disk structure element of radius 1. The eroded image was subtracted from the binarized image to produce an edge.

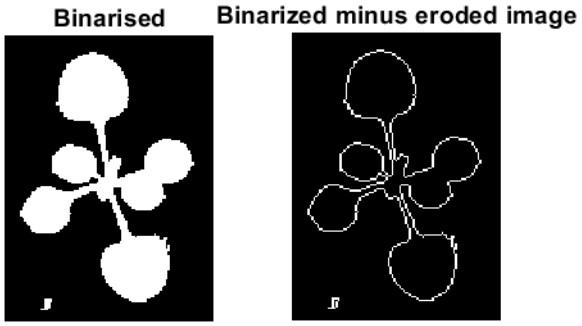

Figure 13 Creating boundaries using the erosion-subtraction technique.

2.5 Evaluation

The solutions were evaluated using precision, recall and F1 score (Appendix D), on a pixel comparison (against ground truth) basis. The solution was experimented with the two images seen in previous diagrams (plant005_rgb, plant002_rgb) but evaluated with the entire 50 image dataset. Unfortunately, the leaf connections were not successfully visualised and so their comparison will not occur, the Hough transform thereby being excluded also. The results were as follows.

The pipeline with the best recall (0.628) was that of the high-pass filter through Fast Fourier Transform. It correctly identifies most border pixels on average, although has a relatively higher standard deviation (0.152) than the other methods, indicating that it does not generalise well to all images.

The pipeline with the best precision (0.305) was the erosion subtraction technique without border clearing. It correctly identifies the most boundary pixels out of all the pixels labelled as boundary (considering false positives). This outperforms the high-pass filter as it considers the possible noise picked up as border. Perhaps border-clearing removes some components connected to the central plant due to inadequate noise removal. This method also has a next best recall under the high-pass filter. This leads the erosion subtraction solution to be the most efficient with an F1-score of 0.140.

It is also interesting to note that a canny filter is less accurate after gaussian smoothing. There may be a need to remove noise using other filtering techniques, as this problem seems to be the biggest challenge with its variance. A modern approach could be the use of deep learning. Semantic segmentation could be used to detect the plant in the image, while instance segmentation may detect individual leaves. Arising advances such as the use of U-Net (Ronneberger, Fischer, Brox 2015) or Transformer learning (Carion et al., 2020) may be of great aid.

3 References

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A. and Zagoruyko, S. (2020). End-to-End Object Detection with Transformers. [online] Available at: https://scontent-lhr8-1.xx.fbcdn.net/v/t39.2365-6/154305880_816694605586461_2873294970659239190_n.pdf?_nc_cat=108&ccb=1-7&_nc_sid=3c67a6&_nc_ohc=n6iMxGnIhxcAX9QLQqp&_nc_ht=scontent-lhr8-1.xx&oh=00_AfCO_u_I1AJW9I9eT07L5rezPopY-S5W9JRLx-6OJjRvhQ&oe=63CBD1C3 [Accessed 17 Jan. 2023].

MathWorks. (2022). Short-time Fourier transform - MATLAB stft - MathWorks United Kingdom. [online] Available at: https://uk.mathworks.com/help/signal/ref/stft.html#mw_f6bebd3d-2957-47ad-94bc-6982eed99a32_sep_mw_c056db1e-cade-47af-bf56-37cd76eee5db [Accessed 16 Jan. 2023].

Ronneberger O, Fischer P, Brox T (2015) U-Net: Convolutional Networks for Biomedical Image Segmentation. In: Navab N., Hornegger J., Wells W., Frangi A. (eds) Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015. MICCAI 2015. Lecture Notes in Computer Science, vol 9351.

4 Appendix

4.1 A. STFT @ Window size 1024

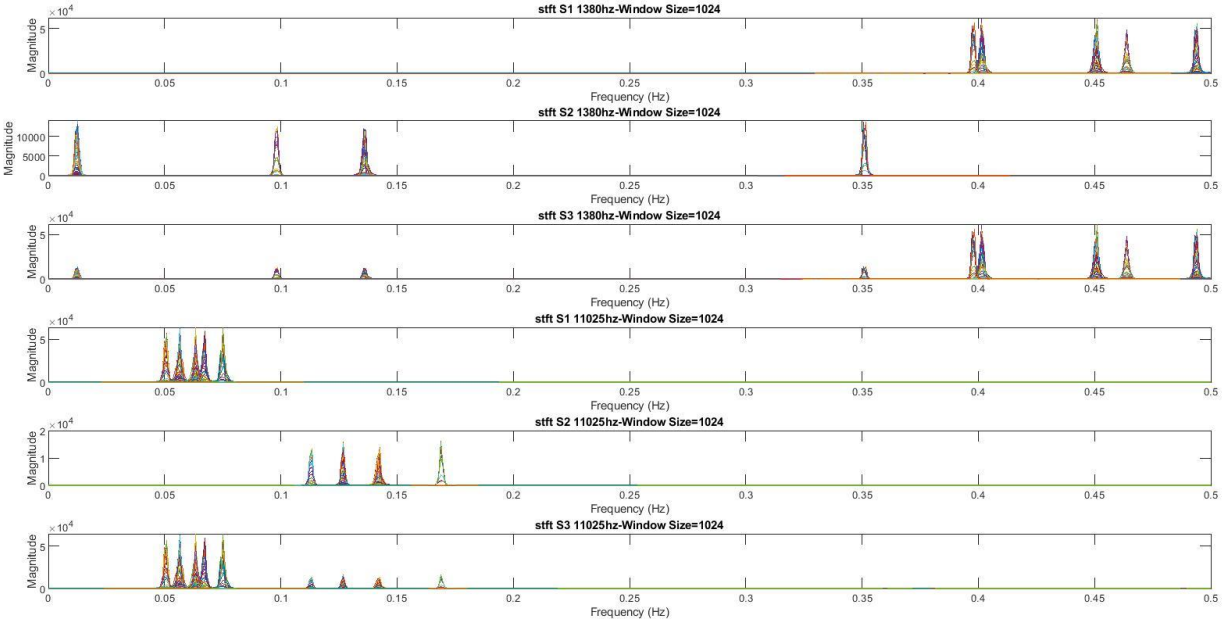

Figure 14 STFT @ Window size 1024

4.2 B. STFT @ Window size 2048

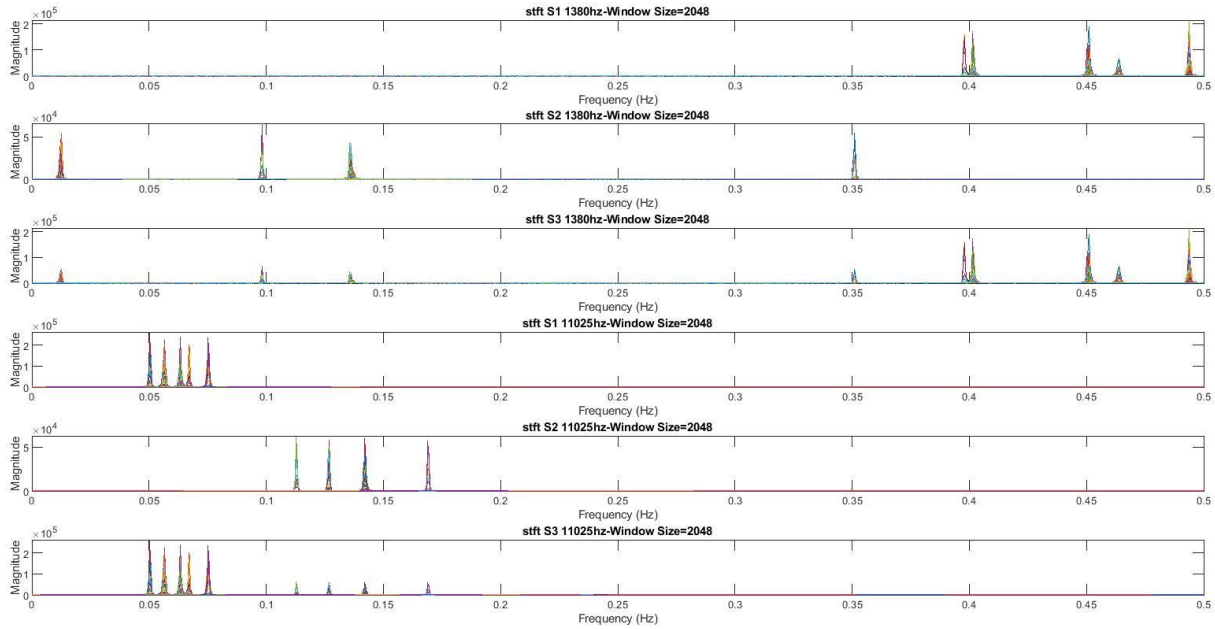

Figure 15 STFT @ Window size 2048

4.3 C. Scalogram of Signals using Amor, Bump Wavelets

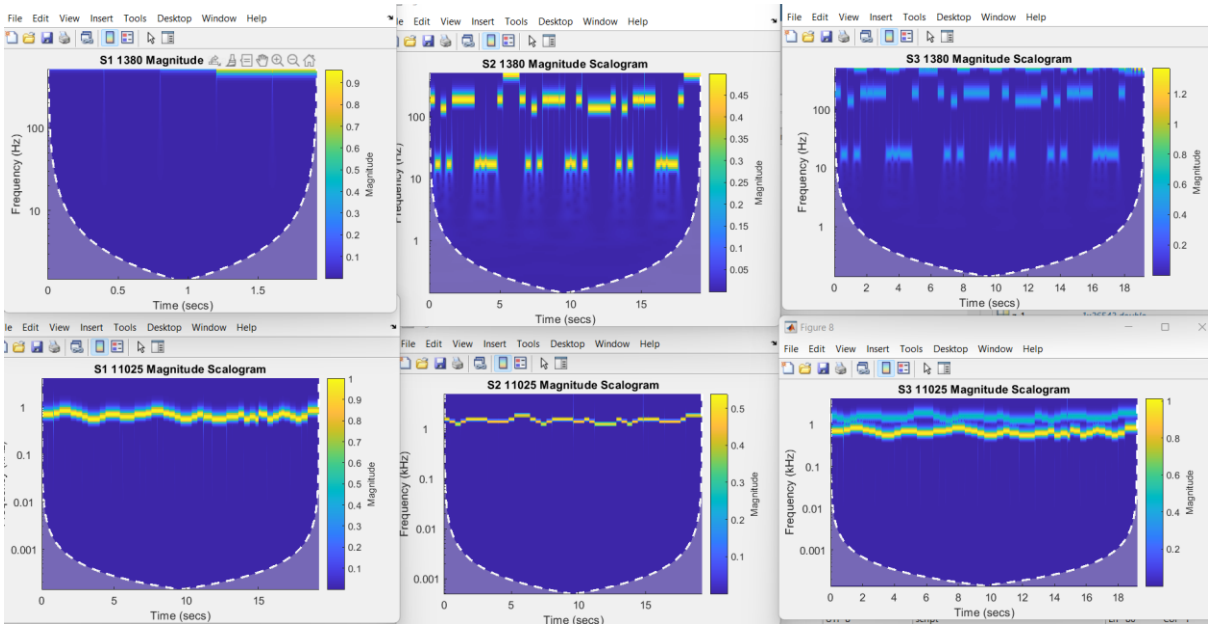

Figure 16 Scalogram of signals using Amor wavelet.

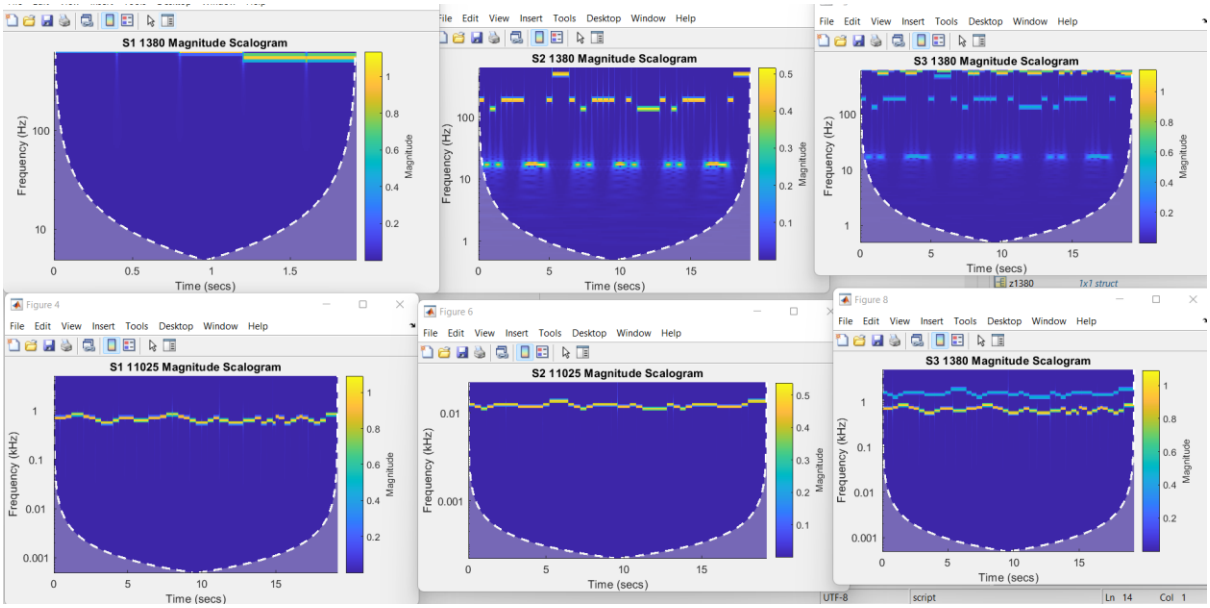

Figure 17 Scalogram of signals using Bump wavelet.

4.4 D. Evaluation Metrics of Different Image Processing Pipelines.

Table 1 Evaluation Metrics of Different Image Processing Pipelines.

Method	Mean Precision	Std. Deviation Precision	Mean Recall	Std. Deviation Recall	Mean F1-Score	Std. Deviation F1-Score
Canny & Gauss. Filter	0.179	0.109	0.312	0.106	0.222	0.114
Canny w/o smoothing	0.207	0.118	0.355	0.107	0.256	0.121
FFT high-pass filter	0.159	0.067	0.628	0.152	0.249	0.088
Erosion Subtraction, Border Clear	0.296	0.202	0.375	0.229	0.327	0.211
Erosion Subtraction, no B.C.	0.305	0.156	0.471	0.111	0.359	0.140

4.5 E. Plotting Signals

```
clc;clear;close all;

%This file plots signals and checks correlation

%First signal
x1380 = load("S1_1380.mat");
x11025 = load("S1_11025.mat");
x_1 = x1380.x_1;
x_2 = x11025.x_1;
%soundsc(x_1,1380);

%set analysis to L seconds of signal (0-19)
L = 2;

%first signal first hz
Fs = x1380.fs;
dt = 1/Fs;
t = 0:dt:L;

figure(1);
subplot(6,1,1);
plot(t, x_1(1:length(t)));
title('S1 1380hz');
xlabel('t');
ylabel('amplitude')

%first signal second hz
Fs = x11025.fs;
dt = 1/Fs;
t = 0:dt:L;

subplot(6,1,2);
plot(t, x_2(1:length(t)));
title('S1 11025hz');
xlabel('t');
ylabel('amplitude')

%second signal
y1380 = load("S2_1380.mat");
y11025 = load("S2_11025.mat");
y_1 = y1380.x_2;
y_2 = y11025.x_2;

%second signal first hz
Fs = y1380.fs;
dt = 1/Fs;
```

```
t = 0:dt:L;

subplot(6,1,3);
plot(t, y_1(1:length(t)));
title('S2 1380hz');
xlabel('t');
ylabel('amplitude')

%second signal second hz
Fs = y11025.fs;
dt = 1/Fs;
t = 0:dt:L;

subplot(6,1,4);
plot(t, y_2(1:length(t)));
title('S2 11025hz');
xlabel('t');
ylabel('amplitude')

%third signal
z1380 = load("S3_1380.mat");
z11025 = load("S3_11025.mat");
z_1 = z1380.x_3;
z_2 = z11025.x_3;

%third signal first hz
Fs = z1380.fs;
dt = 1/Fs;
t = 0:dt:L;

subplot(6,1,5);
plot(t, z_1(1:length(t)));
title('S3 1380hz');
xlabel('t');
ylabel('amplitude')

%third signal second hz
Fs = z11025.fs;
dt = 1/Fs;
t = 0:dt:L;

subplot(6,1,6);
plot(t, z_2(1:length(t)));
title('S3 11025hz');
xlabel('t');
ylabel('amplitude')
```

```
%check correlation of first 2 added signals and third (lower hertz)
pd = zeros([1 8]); %padding

y = [y_1 pd];
xy =  x_1 + y;

figure(2);

[c,lags] = xcorr(xy,z_1);
stem(lags,c)
title('1380hz autocorrelation');
xlabel('lag');
ylabel('autocorrelation');

display(isequal(xy,z_1)); %displays 1(true) therefore signal 1+2 = 3.

%check correlation of first 2 added signals and third (higher hertz)
pd = zeros([1 8]); %padding

y = [y_2 pd];
xy =  x_2 + y;

figure(3);

[c,lags] = xcorr(xy,z_2);
stem(lags,c)
title('11025hz autocorrelation');
xlabel('lag');
ylabel('autocorrelation');

display(isequal(xy,z_2)); %displays 1(true) therefore signal 1+2 = 3.
```

4.6 F. FFT function

```
classdef ft_functions
   methods
       %computes fft, single-sided spectrum P1 based on two-sides spectrum
       %P2 for plotting fast fourier transform
      function [f,P1] = fastf(obj,x,fs)
         Fs = fs;            % Sampling frequency
         T = 1/Fs;             % Sampling period
         L = length(x);      % Length of signal
         t = (0:L-1)*T;        % Time vector

         n = length(t);
         Y = fft(x, n);

         P2 = abs(Y/L);
         P1 = P2(1:floor(L/2+1));
         P1(2:end-1) = 2*P1(2:end-1);

         f = Fs*(0:(L/2))/L;
      end
   end
end
```

4.7 G. Fast Fourier Transform

```
clear;

%FFT of signals
%Note i included the FFT calculation in the ft_functions file for easier
%reuse.

%First signal
x1380 = load("S1_1380.mat");
x11025 = load("S1_11025.mat");
x_1 = x1380.x_1;
x_2 = x11025.x_1;
%soundsc(x_1,1380);

%second signal
y1380 = load("S2_1380.mat");
y11025 = load("S2_11025.mat");
y_1 = y1380.x_2;
y_2 = y11025.x_2;

%third signal
z1380 = load("S3_1380.mat");
z11025 = load("S3_11025.mat");
z_1 = z1380.x_3;
z_2 = z11025.x_3;


%Get functions from fourier transform object
ft_func = ft_functions;

%first signal first hz
[f,P1] = ft_func.fastf(x_1,x1380.fs);
subplot(6,1,1);
plot(f,P1);
title('fft S1 1380hz');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

%second signal first hz
[f2,P2] = ft_func.fastf(y_1,y1380.fs);
subplot(6,1,2);
plot(f2,P2);
title('fft S2 1380hz');
xlabel('Frequency (Hz)');
ylabel('Magnitude');


%third signal first hz
[f,P1] = ft_func.fastf(z_1,z1380.fs);
subplot(6,1,3);
plot(f,P1);
title('fft S3 1380hz');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

%first signal second hz
subplot(6,1,4);
[f,P1] = ft_func.fastf(x_2,x11025.fs);
plot(f,P1);
title('fft S1 11025hz');
xlabel('Frequency (Hz)');
ylabel('Magnitude');


%second signal second hz
subplot(6,1,5);
[f,P1] = ft_func.fastf(y_2,y11025.fs);
plot(f,P1);
title('fft S2 11025hz');
xlabel('Frequency (Hz)');
ylabel('Magnitude');


%third signal second hz
subplot(6,1,6);
[f,P1] = ft_func.fastf(z_2,z11025.fs);
plot(f,P1);
title('fft S3 11025hz');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
```

4.8 H. STFT Analysis

```
clc;clear;close all;

%stft at different window sizes

M = 4096; % window length
g = hann(M,"periodic"); % window
L = floor(0.75*M); % window Overlap
Ndft = M; % FFT Length


%First signal
x1380 = load("S1_1380.mat");
x11025 = load("S1_11025.mat");
x_1 = x1380.x_1;
x_2 = x11025.x_1;
%soundsc(x_1,1380);
figure(1);
s = strcat('-Window Size=',int2str(M)); %window size string

%second signal
y1380 = load("S2_1380.mat");
y11025 = load("S2_11025.mat");
y_1 = y1380.x_2;
y_2 = y11025.x_2;

%third signal
z1380 = load("S3_1380.mat");
z11025 = load("S3_11025.mat");
z_1 = z1380.x_3;
z_2 = z11025.x_3;

figure(1)

%first signal first hz
[st,ft,tt] = stft(x_1,x1380.fs, Window=g, OverlapLength=L, FFTLength=Ndft, FrequencyRange="onesided");
subplot(6,1,1);
plot(ft/x1380.fs,abs(st).^2);
title(strcat('stft S1 1380hz',s));
xlabel('Frequency (Hz)');
ylabel('Magnitude');

%second signal first hz
[st,ft,tt] = stft(y_1,y1380.fs, Window=g, OverlapLength=L, FFTLength=Ndft, FrequencyRange="onesided");
subplot(6,1,2);
plot(ft/y1380.fs,abs(st).^2);
title(strcat('stft S2 1380hz',s));
xlabel('Frequency (Hz)');
ylabel('Magnitude');

%third signal first hz
[st,ft,tt] = stft(z_1,z1380.fs, Window=g, OverlapLength=L, FFTLength=Ndft, FrequencyRange="onesided");
subplot(6,1,3);
plot(ft/z1380.fs,abs(st).^2);
title(strcat('stft S3 1380hz',s));
xlabel('Frequency (Hz)');
ylabel('Magnitude');

%first signal second hz
subplot(6,1,4);
[st,ft,tt] = stft(x_2,x11025.fs, Window=g, OverlapLength=L, FFTLength=Ndft, FrequencyRange="onesided");
plot(ft/x11025.fs,abs(st).^2);
title(strcat('stft S1 11025hz',s));
xlabel('Frequency (Hz)');
ylabel('Magnitude');


%second signal second hz
subplot(6,1,5);
[st,ft,tt] = stft(y_2,y11025.fs, Window=g, OverlapLength=L, FFTLength=Ndft, FrequencyRange="onesided");
plot(ft/y11025.fs,abs(st).^2);
title(strcat('stft S2 11025hz',s));
xlabel('Frequency (Hz)');
ylabel('Magnitude');


%third signal second hz
subplot(6,1,6);
[st,ft,tt] = stft(z_2,z11025.fs, Window=g, OverlapLength=L, FFTLength=Ndft, FrequencyRange="onesided");
plot(ft/z11025.fs,abs(st).^2);
title(strcat('stft S3 11025hz',s));
xlabel('Frequency (Hz)');
ylabel('Magnitude');
```

4.9 I. Spectrogram Analysis

```
clear;

M = 4096; % window length
g = hann(M,"periodic"); % window
L = 0.75*M; % window Overlap
Ndft = M; % FFT Length


%First signal
x1380 = load("S1_1380.mat");
x11025 = load("S1_11025.mat");
x_1 = x1380.x_1;
x_2 = x11025.x_1;
%soundsc(x_1,1380);

%second signal
y1380 = load("S2_1380.mat");
y11025 = load("S2_11025.mat");
y_1 = y1380.x_2;
y_2 = y11025.x_2;

%third signal
z1380 = load("S3_1380.mat");
z11025 = load("S3_11025.mat");
z_1 = z1380.x_3;
z_2 = z11025.x_3;

%first signal first hz
[sx,fx,tx] = spectrogram(x_1, g, L,Ndft, x1380.fs, "onesided");
subplot(3,2,1);
waterfall(tx,fx/x1380.fs, abs(sx).^2);
title('Spectrogram S1 1380hz');
xlabel('time[sec]')
ylabel('frequency[kHz]')
zlabel('Magnitude[dB]')

%first signal second hz
subplot(3,2,2);
[sx,fx,tx] = spectrogram(x_2, g, L,Ndft, x11025.fs, "onesided");
waterfall(tx,fx/x11025.fs, abs(sx).^2);
title('Spectrogram S1 11025hz');
xlabel('time[sec]')
ylabel('frequency[kHz]')
zlabel('Magnitude[dB]')

%second signal first hz
[sx,fx,tx] = spectrogram(y_1, g, L,Ndft, y1380.fs, "onesided");
subplot(3,2,3);
waterfall(tx,fx/y1380.fs, abs(sx).^2);
title('Spectrogram S2 1380hz');
xlabel('time[sec]')
ylabel('frequency[kHz]')
zlabel('Magnitude[dB]')

%second signal second hz
subplot(3,2,4);
[sx,fx,tx] = spectrogram(y_2, g, L,Ndft, y11025.fs, "onesided");
waterfall(tx,fx/y11025.fs, abs(sx).^2);
title('Spectrogram S2 11025hz');
xlabel('time[sec]')
ylabel('frequency[kHz]')
zlabel('Magnitude[dB]')



%third signal first hz
[sx,fx,tx] = spectrogram(z_1, g, L,Ndft, z1380.fs, "onesided");
subplot(3,2,5);
waterfall(tx,fx/z1380.fs, abs(sx).^2);
title('Spectrogram S3 1380hz');
xlabel('time[sec]')
ylabel('frequency[kHz]')
zlabel('Magnitude[dB]')

%third signal second hz
subplot(3,2,6);
[sx,fx,tx] = spectrogram(z_2, g, L,Ndft, z11025.fs, "onesided");
waterfall(tx,fx/z11025.fs, abs(sx).^2);
title('Spectrogram S3 11025hz');
xlabel('time[sec]')
ylabel('frequency[kHz]')
zlabel('Magnitude[dB]')
```

4.10 J. Continuous Wavelength Transform & Scalogram Analysis

```
clc;clear;close all;

%First signal
x1380 = load("S1_1380.mat");
x11025 = load("S1_11025.mat");
x_1 = x1380.x_1;
x_2 = x11025.x_1;

%second signal
y1380 = load("S2_1380.mat");
y11025 = load("S2_11025.mat");
y_1 = y1380.x_2;
y_2 = y11025.x_2;

%third signal
z1380 = load("S3_1380.mat");
z11025 = load("S3_11025.mat");
z_1 = z1380.x_3;
z_2 = z11025.x_3;

%soundsc(x_1,1380);

%first signal first hz


% We create a CWT filter bank that can be applied to the signal.
fb = cwtfilterbank(SignalLength=numel(x_1),SamplingFrequency=x1380.fs,Wavelet = "morse");
% the plot to confirm frequency range
figure(1)
freqz(fb);


figure(2)
cwt(x_1,FilterBank=fb);

title("S1 1380 Magnitude Scalogram")


%first signal second hz

% We create a CWT filter bank that can be applied to the signal.
fb = cwtfilterbank(SignalLength=numel(x_2),SamplingFrequency=x11025.fs,Wavelet = "morse");
%Need to determine frequency limits as per documentation.

% the plot to confirm frequency range
figure(3)
freqz(fb);

figure(4)
cwt(x_2,FilterBank=fb);
title("S1 11025 Magnitude Scalogram");


%second signal first hz
fb = cwtfilterbank(SignalLength=numel(y_1),SamplingFrequency=y1380.fs,Wavelet = "morse");

figure(5)
cwt(y_1,FilterBank=fb);
title("S2 1380 Magnitude Scalogram")

%second signal second hz
fb = cwtfilterbank(SignalLength=numel(y_2),SamplingFrequency=y11025.fs,Wavelet = "morse");

figure(6)
cwt(y_2,FilterBank=fb);
title("S2 11025 Magnitude Scalogram");

%third signal first hz
%second signal first hz
fb = cwtfilterbank(SignalLength=numel(z_1),SamplingFrequency=z1380.fs,Wavelet = "morse");

figure(7)
cwt(z_1,FilterBank=fb);
title("S3 1380 Magnitude Scalogram")

%third signal second hz
fb = cwtfilterbank(SignalLength=numel(z_2),SamplingFrequency=z11025.fs,Wavelet = "morse");

figure(8)
cwt(z_2,FilterBank=fb);
title("S3 11025 Magnitude Scalogram")
```

4.11 K. FIR Filter (1380Hz)

```
%tries lowpass with low frequency signals
clear;
% 1. Design a FIR filter
% 2. Cut-off all frequencies of the S1_1380 Signal
% 3. Doesn't cut off frequencies of S2_1380 Signal
% 4. Apply your filter to S3_1380 Signal
% 5. Plot frequencies of filtered signal
% 6. Repeat for 11025Hz


%Get functions from fourier transform object
ft_func = ft_functions;

%lowpass for 1380
lp = 470;

%steepness
stp = 0.8;

%First signal
x1380 = load("S1_1380.mat");
x_1 = x1380.x_1;

subplot(6,1,1);
[f,P1] = ft_func.fastf(x_1,x1380.fs);
plot(f,P1);
title('fft S1 1380hz before filter');
xlabel('Frequency (Hz)');
ylabel('Magnitude');


low1 = lowpass(x_1,lp,x1380.fs,'ImpulseResponse','fir','Steepness',stp);

subplot(6,1,2);

[f,P1] = ft_func.fastf(low1,x1380.fs);
plot(f,P1);
title('fft S1 1380hz after filter');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

%second signal
y1380 = load("S2_1380.mat");
y_1 = y1380.x_2;

subplot(6,1,3);
[f,P1] = ft_func.fastf(y_1,y1380.fs);
plot(f,P1);
title('fft S2 1380hz before filter');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

subplot(6,1,4);

low2 = lowpass(y_1,lp,y1380.fs,'ImpulseResponse','fir','Steepness',stp);
[f,P1] = ft_func.fastf(low2,y1380.fs);
plot(f,P1);
title('fft S2 1380hz after filter');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

%apply to third signal
z1380 = load("S3_1380.mat");
z_1 = z1380.x_3;

subplot(6,1,5);
[f,P1] = ft_func.fastf(z_1,z1380.fs);
plot(f,P1);
title('fft S3 1380hz before filter');
xlabel('Frequency (Hz)');
ylabel('Magnitude');


subplot(6,1,6);

low3 = lowpass(z_1,lp,z1380.fs,'ImpulseResponse','fir','Steepness',stp);
[f,P1] = ft_func.fastf(low3,z1380.fs);
plot(f,P1);
title('fft S3 1380hz after filter');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
```

4.12 L. FIR Filter (11025Hz)

```
%tries highpass with high frequency signals
clear;
% 1. Design a FIR filter
% 2. Cut-off all frequencies of the S1_1380 Signal
% 3. Doesn't cut off frequencies of S2_1380 Signal
% 4. Apply your filter to S3_1380 Signal
% 5. Plot frequencies of filtered signal
% 6. Repeat for 11025Hz

%Get functions from fourier transform object
ft_func = ft_functions;

%highpass for 11025
hp = 1200;

%steepness
stp = 0.7;

%First signal
x11025 = load("S1_11025.mat");
x_2 = x11025.x_1;

subplot(6,1,1);
[f,P1] = ft_func.fastf(x_2,x11025.fs);
plot(f,P1);
title('fft S1 11025hz before filter');
xlabel('Frequency (Hz)');
ylabel('Magnitude');


high1 = highpass(x_2,hp,x11025.fs,'ImpulseResponse','fir','Steepness',stp);

subplot(6,1,2);

[f,P1] = ft_func.fastf(high1,x11025.fs);
plot(f,P1);
title('fft S1 11025hz after filter');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

%second signal
y11025 = load("S2_11025.mat");
y_2 = y11025.x_2;

subplot(6,1,3);
[f,P1] = ft_func.fastf(y_2,y11025.fs);
plot(f,P1);
title('fft S2 11025hz before filter');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

subplot(6,1,4);

high2 = highpass(y_2,hp,y11025.fs,'ImpulseResponse','fir','Steepness',stp);
[f,P1] = ft_func.fastf(high2,y11025.fs);
plot(f,P1);
title('fft S2 11025hz after filter');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

%apply to third signal
z11025 = load("S3_11025.mat");
z_2 = z11025.x_3;

subplot(6,1,5);
[f,P1] = ft_func.fastf(z_2,z11025.fs);
plot(f,P1);
title('fft S3 11025hz before filter');
xlabel('Frequency (Hz)');
ylabel('Magnitude');


subplot(6,1,6);

high3 = highpass(z_2,hp,z11025.fs,'ImpulseResponse','fir','Steepness',stp);
[f,P1] = ft_func.fastf(high3,z11025.fs);
plot(f,P1);
title('fft S3 11025hz after filter');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
```

4.13 M. Morphological/Canny edge detection

```
clear; close all;

img = imread("DATASET/ara2013_plant005_rgb.png");

grey = rgb2gray(img);

subplot(3,6,1);
imshow(grey);
title('Greyscale');

[counts,x] = imhist(grey,16);

subplot(3,6,2);

stem(x,counts);
title('Greyscale Histogram');
xlabel('Pixel Intensity');
ylabel('Count');

%adjust contrast
sep = imadjust(grey,[0.4 0.6],[]);
[counts,x] = imhist(sep,16);
subplot(3,6,3);
imshow(sep);
title('Contrast adjusted leaves');

subplot(3,6,4);
stem(x,counts)
title('Histogram after leaf contrast');
xlabel('Pixel Intensity');
ylabel('Count');

J = imadjust(grey,[0.2 0.5],[]);

[counts,x] = imhist(J,16);
subplot(3,6,5);
imshow(J);
title('Contrast adjusted full');

J = imgaussfilt(J,0.9);

subplot(3,6,6);
imshow(J);
title('Gaussian filter');

subplot(3,6,7);
stem(x,counts)
title('Histogram after full contrast');
xlabel('Pixel Intensity');
ylabel('Count');


%binarize
T = adaptthresh(sep,0.99);%adapt threshold

BW = imbinarize(sep,T);
subplot(3,6,8);
```

```
imshow(BW);
title('Binarized leaves');

%clear noise
BWClean = bwareaopen(BW, 70);

subplot(3,6,9);
imshow(BWClean)
title('Leaves w/o small objects');


se = strel('disk', 2);

cl = imclose(BWClean,se);
subplot(3,6,10);
imshow(cl);
title('imClosed leaves');

subplot(3,6,11);
bw = imfill(cl, 'holes');
imshow(bw);
title('Leaves w/ filled holes');

%have this for leaves, full segment for stalks

subplot(3,6,12);
edg = edge(bw,'canny');
imshow(edg);
title('Canny leaf edges');

%no seperation
%binarize
T = adaptthresh(J,0.99);%adapt threshold

BW = imbinarize(J,T);


%clear noise
BWClean = bwareaopen(BW, 70);


subplot(3,6,13);
bwMerge = imfill(BWClean, 'holes');
bwMerge = imclearborder(bwMerge,4); %clear connections to border
title('Canny leaf edges');

edgMerge = edge(bwMerge,'canny');
imshow(edgMerge);
title('Full image canny');

subplot(3,6,14);
%minus images to get connections
Z=((1-bw) - (1-bwMerge));  %Preferred
imshow(Z);
title('Binarized full image minus leaf sections');

subplot(3,6,15);
%minus edges to get connecting components
cons = Z-edgMerge -edg;
```

```
imshow(cons);
title('Connected leaf components');

%erode
se = strel('disk',2);
conEr = imerode(cons,se);

subplot(3,6,16);
imshow(conEr);
title('Eroding connections');

%detect edge
conEdg = edge(conEr,'canny');


subplot(3,6,17);
imshow(conEdg);
title('Edge of connections');
```

4.14 N. Circle Hough Transform

```
%Attempt to split leaves using hough circles

clear; close all;

img = imread("DATASET/ara2013_plant002_rgb.png");

grey = rgb2gray(img);


%[r, c] = size(I_resized);

subplot(2,6,1);
imshow(grey);
title('Greyscale');

%adjust contrast
J = imadjust(grey,[0.2 0.8],[]);

subplot(2,6,2);
imshow(J);
title('Contrast adjust');

%binarize
T = adaptthresh(J,0.99);%adapt threshold

BW = imbinarize(J,T);
subplot(2,6,3);

imshow(BW);
title('Binarized');

%clear noise
BWClean = bwareaopen(BW, 70);

subplot(2,6,4);
imshow(BWClean)
title('Small items removed');

se = strel('disk', 1);
BW2 = imopen(BWClean,se);

subplot(2,6,5);
imshow(BW2);
title('After opening');

edges = edge(BW2,'canny');
```

```
subplot(2,6,6);
%output_image = imbinarize(output_image); % remove fft noise
imshow(edges);
title('After Canny with found circles');

[c1, r1] = imfindcircles(edges,[10,30], 'Sensitivity', 0.94);

viscircles(c1, r1,'EdgeColor','w');

%add circles to binary image to separate them

subplot(2,6,7);
imshow(BW);
title('Circles applied to binary image');
viscircles(c1, r1,'EdgeColor','black');

%create new image

%https://stackoverflow.com/questions/26906928/separate-two-overlapping-circles-in-
an-image-using-matlab
num_circles = numel(r1); %// Get number of circles
struct_reg = []; %// Save the shape analysis per circle / line here

%// For creating our circle in the temporary image
[X,Y] = meshgrid(1:size(BW,2), 1:size(BW,1));

%// Storing all of our circles in this image
circles_img = false(size(BW));

for idx = 1 : num_circles %// For each circle we have...
    %// Place our circle inside a temporary image
    r = r1(idx);
    cx = c1(idx,1); cy = c1(idx,2);
    tmp = (X - cx).^2 + (Y - cy).^2 <= r^2;

    % // Save in master circle image
    circles_img(tmp) = true;
end
subplot(2,6,8);
%now got leaves as circles image
imshow(circles_img);
title('Circles alone');

circEdge = edge(circles_img,'canny');
subplot(2,6,9);
imshow(circEdge);
```

```
title('Circles alone edges');

subplot(2,6,10);
fullEdge = edge(BWClean,'canny');
imshow(fullEdge);
title('Canny edge of original image');

subplot(2,6,11);
Z=1-((1-circEdge) & (1-fullEdge));  %Preferred
imshow(Z);
title('Union of circle edges and OG edges');

subplot(2,6,12);
%Z = imfill(Z,'holes');
se = strel('disk',1);
new = imopen(Z,se);
se = strel('disk',25);
new = imclose(new,se);
imshow(new);
title('Attempt at isolating leaf connections');
```

4.15 O. 2D FFT Boundary Detection

```
clear; close all;

img = imread("DATASET/ara2013_plant001_rgb.png");

grey = rgb2gray(img);

[r, c] = size(grey);

subplot(2,4,1);
imshow(grey);
title('Greyscale Image');

%adjust contrast
J = imadjust(grey,[0.2 0.5],[]);

subplot(2,4,2);
imshow(J);
title('Contrast-adjusted Image');

T = adaptthresh(J,0.99);
%binarize
BW = imbinarize(J,T);
subplot(2,4,3);

imshow(BW);
title('Binarised Image');

FT_img = fft2(double(BW));

subplot(2,4,4);
imshow(FT_img);
title('FFT of Image');

% Assign Cut-off Frequency
D0 = 10; % one can change this value accordingly

%https://www.geeksforgeeks.org/matlab-ideal-highpass-filter-in-image-processing/
% Designing filter
u = 0:(r-1);
idx = find(u>r/2);
u(idx) = u(idx)-r;
v = 0:(c-1);
idy = find(v>c/2);
v(idy) = v(idy)-c;

% MATLAB library function meshgrid(v, u) returns 2D grid
%  which contains the coordinates of vectors v and u.
% Matrix V with each row is a copy of v, and matrix U
% with each column is a copy of u
[V, U] = meshgrid(v, u);

% Calculating Euclidean Distance
D = sqrt(U.^2+V.^2);

% Comparing with the cut-off frequency and
% determining the filtering mask
H = double(D > D0);

% Convolution between the Fourier Transformed image and the mask
G = H.*FT_img;

subplot(2,4,5);
imshow(G);
title('FFT of Image after highpass');
% Getting the resultant image by Inverse Fourier Transform
% of the convoluted image using MATLAB library function
% ifft2 (2D inverse fast fourier transform)

output_image = real(ifft2(double(G)));


subplot(2,4,6);
imshow(output_image);
title('Inverse image FFT after highpass');

subplot(2,4,7);
output_image = imbinarize(output_image); % remove fft noise no threshold
imshow(output_image);
title('Re-Binarized image');

%imbinarize at different threshold to fully remove fft noise
subplot(2,4,8);
output_image_clean = bwareaopen(BW, 70); % remove fft noise
imshow(output_image_clean);
```

4.16 P. Erosion Subtraction Boundary

```
clc;clear;close all;

img = imread("DATASET/ara2013_plant002_rgb.png");

%greyscale
grey = rgb2gray(img);

subplot(2,3,1);
imshow(grey);
title('Greyscale');

subplot(2,3,2);
J = imadjust(grey,[0.2 0.5],[]);

imshow(J)
title('Contrast adjustment');
T = adaptthresh(J,0.99);


%binarize
BW = imbinarize(J,T);

%remove small objects
BW = bwareaopen(BW, 70);

%fill any holes
BW = imfill(BW, 'holes');


subplot(2,3,3);

BW =imclearborder(BW,4);%removes items connected to edge
imshow(BW);
title('Binarised');

s =strel('disk',1);

%erode
ime =imerode(BW,s);

subplot(2,3,4);
imshow(ime);
title('Eroded');

%minus erosion from original image to get edge
subplot(2,3,5);
imshow(BW-ime);
title('Binarized minus eroded image');
```

4.17 Q. Solution Evaluation

```
%read images and compare
clear; close all;

p = pwd;
d = pwd + "\DATASET";

file_list = dir(d);

bounds = {};
ims = {};

pre = []; %precision vals
re = []; %recall vals
f = []; %F1 vals

%filter to just images and add to list of images to be processed
for i = 1:numel(file_list)

    file = file_list(i);
    [filepath,name,ext] = fileparts(file.name);
    abs_path = fullfile(file.folder, file.name);

    %if file name contains boundary, add to boundary list
    if regexp(file.name, "[a-zA-Z]+2013_[A-Za-z0-9]+_boundaries\.png")
        I = imread(abs_path); % load image
        bounds{end+1} = I; % append to image array
    end

    %if file name rgb, add to image list
    if regexp(file.name, "[a-zA-Z]+2013_[A-Za-z0-9]+_rgb\.png")
        I = imread(abs_path); % load image
        ims{end+1} = I; % append to image array
    end

end

for l=1:length(ims)
    im = cell2mat(ims(l));
    grey = rgb2gray(im);



    %adjust contrast

    J = imadjust(grey,[0.2 0.5],[]);
% %%%%Erosion Method %%%%%
%     T = adaptthresh(J,0.99);
%
%     %binarize
%     BW = imbinarize(J,T);
%     BW = bwareaopen(BW, 70);
%     BW = imfill(BW, 'holes');
%     %BW =imclearborder(BW,4);
%
%     s =strel('disk',1);
%
%     ime =imerode(BW,s);
%
```

```
%     sol = BW-ime;

%%%%% Canny/Morphological Method
%     %J = imgaussfilt(J);
%
%     %no seperation
%     %binarize
%     T = adaptthresh(J,0.99);%adapt threshold
%
%     BW = imbinarize(J,T);
%
%     %remove connections to border
%     %BW = imclearborder(BW,1);
%     %clear noise
%     BWClean = bwareaopen(BW, 70);
%
%
%     bwMerge = imfill(BWClean, 'holes');
%
%     %bwMerge = imclearborder(bwMerge,4);
%
%     edgMerge = edge(bwMerge,'canny');

%%%% FFT high-pass method

    [r, c] = size(J);

    T = adaptthresh(J,0.99);
    %binarize
    BW = imbinarize(J,T);

    FT_img = fft2(double(BW));

    % Assign Cut-off Frequency
    D0 = 10; % one can change this value accordingly

    %https://www.geeksforgeeks.org/matlab-ideal-highpass-filter-in-image-
processing/
    % Designing filter
    u = 0:(r-1);
    idx = find(u>r/2);
    u(idx) = u(idx)-r;
    v = 0:(c-1);
    idy = find(v>c/2);
    v(idy) = v(idy)-c;

    % MATLAB library function meshgrid(v, u) returns 2D grid
    %  which contains the coordinates of vectors v and u.
    % Matrix V with each row is a copy of v, and matrix U
    % with each column is a copy of u
    [V, U] = meshgrid(v, u);

    % Calculating Euclidean Distance
    D = sqrt(U.^2+V.^2);

    % Comparing with the cut-off frequency and
    % determining the filtering mask
    H = double(D > D0);
```

```
    % Convolution between the Fourier Transformed image and the mask
    G = H.*FT_img;

    % Getting the resultant image by Inverse Fourier Transform
    % of the convoluted image using MATLAB library function
    % ifft2 (2D inverse fast fourier transform)

    output_image = real(ifft2(double(G)));

    output_image = imbinarize(output_image); % remove fft noise no threshold

    sol = output_image; %solution

    b = cell2mat(bounds(l));
    GT = b == 1;

    %imshowpair(edgMerge, bin_img);
    [rows, columns, numberOfColorChannels] = size(GT); % get rows and cols sizes

    TP=0;FP=0;TN=0;FN=0;
    % compare pixel values for "1" in segmented and GT images and add to
    % relevant metric
    for i=1:rows
        for j=1:columns
            if (sol(i,j) == 1 && GT(i,j) ==1)
                TP = TP+1;
            elseif(sol(i,j) == 1 && GT(i,j) ~= 1)
                FP=FP+1;
            elseif(sol(i,j) ~= 1 && GT(i,j) ~= 1)
                TN = TN +1;
            else
                FN = FN+1;
            end
        end
    end

    % calculate precision & recall
    Precision = TP/(TP+FP);
    Recall = TP/(TP+FN);
    Fone = 2 * ((Precision * Recall) / (Precision + Recall));

    pre = [pre Precision];
    re = [re Recall];
    f = [f Fone];

end


%create table showing metrics
Precision = [pre];
Precision(isnan(Precision))=0; %change NaNs to 0 if present
Recall = [re];
Recall(isnan(Recall))=0;
F1 = [f];
F1(isnan(F1))=0;

metrics = table(Precision,Recall,F1);
disp(metrics);
```

```
Mp = mean(Precision);
Sp = std(Precision);
Mr = mean(Recall);
Sr = std(Recall);
Mf = mean(F1);
Sf = std(F1);

sumMetrics = table(Mp,Sp,Mr,Sr,Mf,Sf);
disp(sumMetrics);
```