OCT Image Dual Balanced Processing & Volume Generation

This project is the culmination of my acquired knowledge from SBME's Optical Linear Systems course and weeks of development and research. This project was presented as a term project and obtained a grade of 98% for exceeding expectations as it was a comprehensive, accurate, and generates viable retinal cross section volumes from raw OCT data obtained from VGH's Clinical Trial OCT System. I would like to thank UBC's Professor Myeong Jin Ju, and Nueralink Engineer Andrew Chen as they have taught me a great deal about the subject of OCT image signal processing and have written many sections and functions this code relies on.

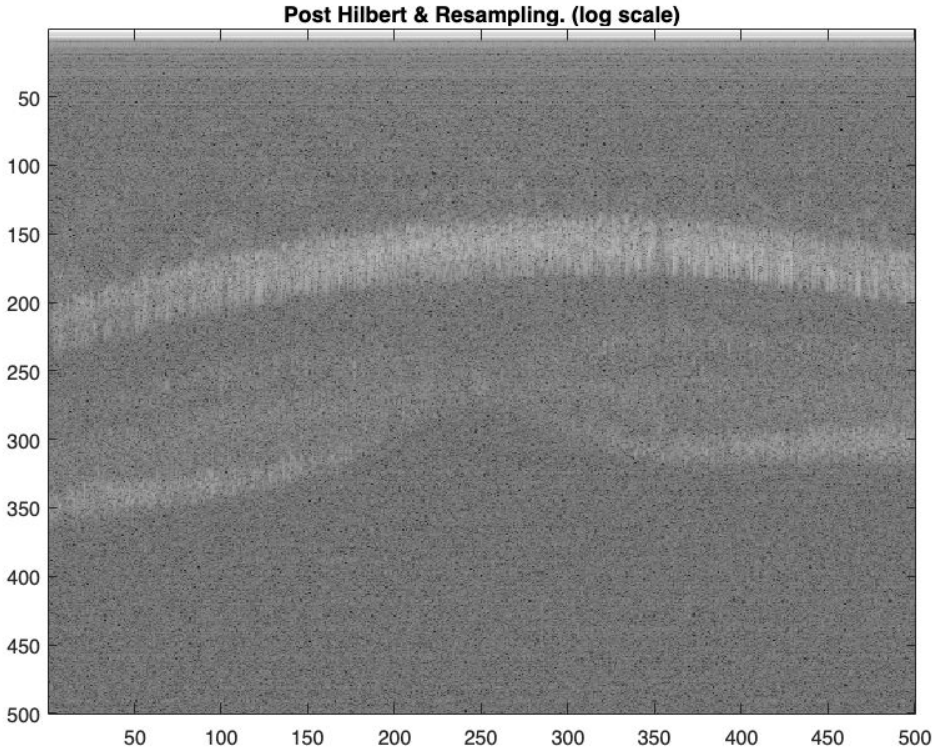
```
%% Setting Up and Importing Data
clear all; clc; close all;

load(fullfile('./RawOCT_A.mat'));
load(fullfile('./RawOCT_B.mat'));

fileID = fopen(fullfile('./LUT_A.bin'),'r'); LUT_A = fread(fileID,'double');
fclose(fileID);
fileID = fopen(fullfile('./LUT_B.bin'),'r'); LUT_B = fread(fileID,'double');
fclose(fileID);
```

Step 1: Processing a reference B-Scan.

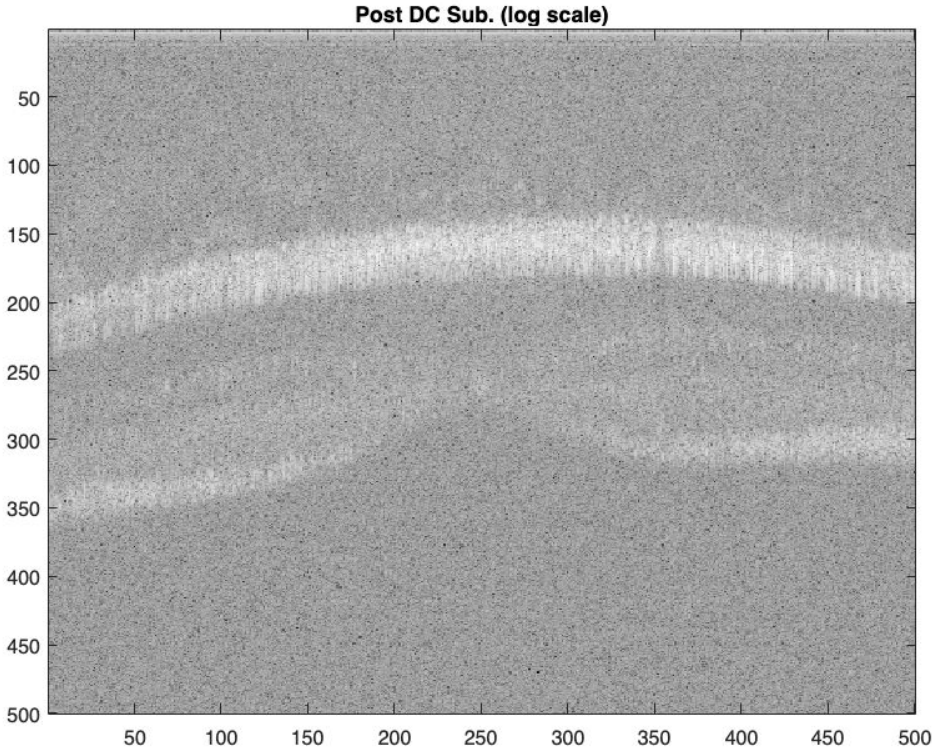
First we prove the efficacy of the processing algorithm by performing the individual steps one by one and examining scan for changes at every step.

The processes used on the B-Scans are: Hilbert Transform→K-Linear Resampling→DC Fixed Pattern Noise Subtraction→Dispersion Estimation and Compensation→Hanning Windowing.

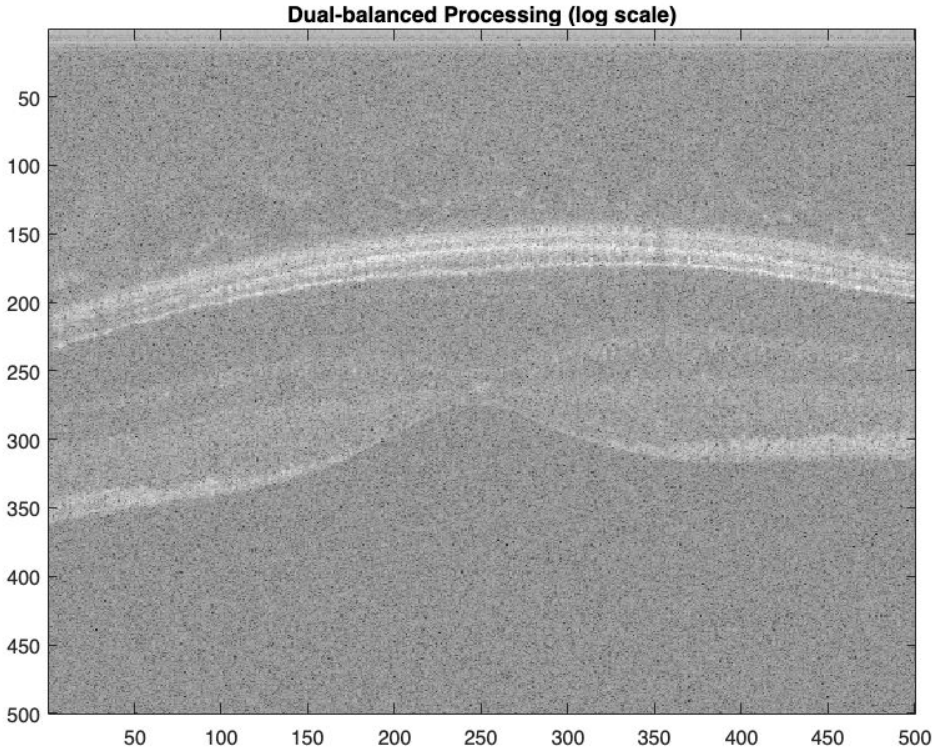
```
%% Hilbert Transform and K-Linear Resampling
CplxRawData_A = hilbert(RawOCT_A(:,:,round(end/2)));
CplxRawData_B = hilbert(RawOCT_B(:,:,round(end/2)));

CplxRawData_Rescaled_A = reSampling_LUT(CplxRawData_A, LUT_A);
CplxRawData_Rescaled_B = reSampling_LUT(CplxRawData_B, LUT_B);
CplxRawData_Rescaled_AB = CplxRawData_Rescaled_A - CplxRawData_Rescaled_B;

FFTData_Rescaled_AB = fft(CplxRawData_Rescaled_AB);
OCTimg = 20.*log10(abs(FFTData_Rescaled_AB(1:500,:)));
imagesc(OCTimg); colormap("gray"); title('Post Hilbert & Resampling. (log
scale)');
```



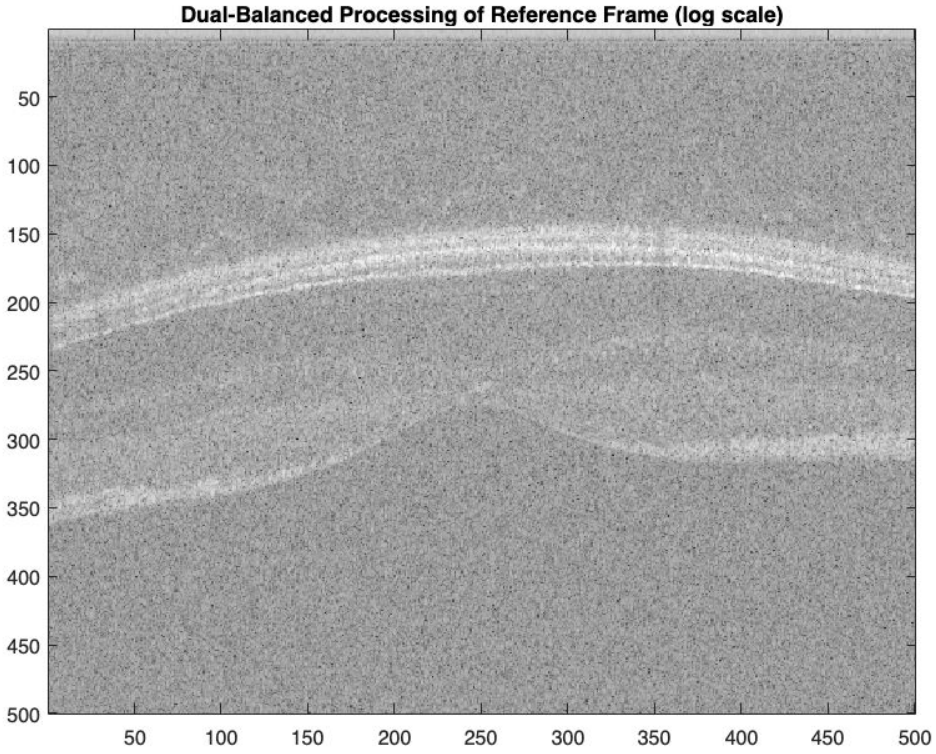

```
%% DC Fixed Pattern Noise Subtraction
CplxRawData_DCSub_AB = CplxRawData_Rescaled_AB
- (repmat(median(real(CplxRawData_Rescaled_AB),2),
[1,size(CplxRawData_Rescaled_AB,2)])
+1j.*repmat(median(imag(CplxRawData_Rescaled_AB),2),
[1,size(CplxRawData_Rescaled_AB,2)]));

FFTData_DCSUB_AB = fft(CplxRawData_DCSub_AB);
OCTimg = 20.*log10(abs(FFTData_DCSUB_AB(1:500,:)));
imagesc(OCTimg); colormap("gray"); title('Post DC Sub. (log scale)');
```




```
%% Dispersion Compensation
% Set dispersion estimation parameters %
dispMaxOrder = 4;
coeffRange = 20;
depthROI = [14, 590];

% Dispersion estimation %
[dispCoeffs, arrDispCoeff, arrCost] =
setDispCoeff(CplxRawData_DCSub_AB,depthROI,dispMaxOrder,coeffRange);

% Dispersion correction %
CplxRawData_DispCorr_AB =
compDisPhase(CplxRawData_DCSub_AB,dispMaxOrder,dispCoeffs);

FFTData_DispCorr_AB = fft(CplxRawData_DispCorr_AB);
imagesc(20.*log10(abs(FFTData_DispCorr_AB(1:500,:))));
colormap("gray"); title('Dual-balanced Processing (log scale)');
```




```
%% Hanning Windowing
CplxRawData_win =
CplxRawData_DispCorr_AB.*repmat(hann(size(CplxRawData_DispCorr_AB,1)), [1
size(CplxRawData_DispCorr_AB,2)]);

%% Final Reference Image.
Ref_FFTData_Final = fft(CplxRawData_win);
Ref_OCTimg_Final = 20.*log10(abs(Ref_FFTData_Final(1:500,:)));
imagesc(Ref_OCTimg_Final); colormap("gray"); title('Dual-Balanced Processing of
Reference Frame (log scale)');
```



Step 2: Volume Generation.

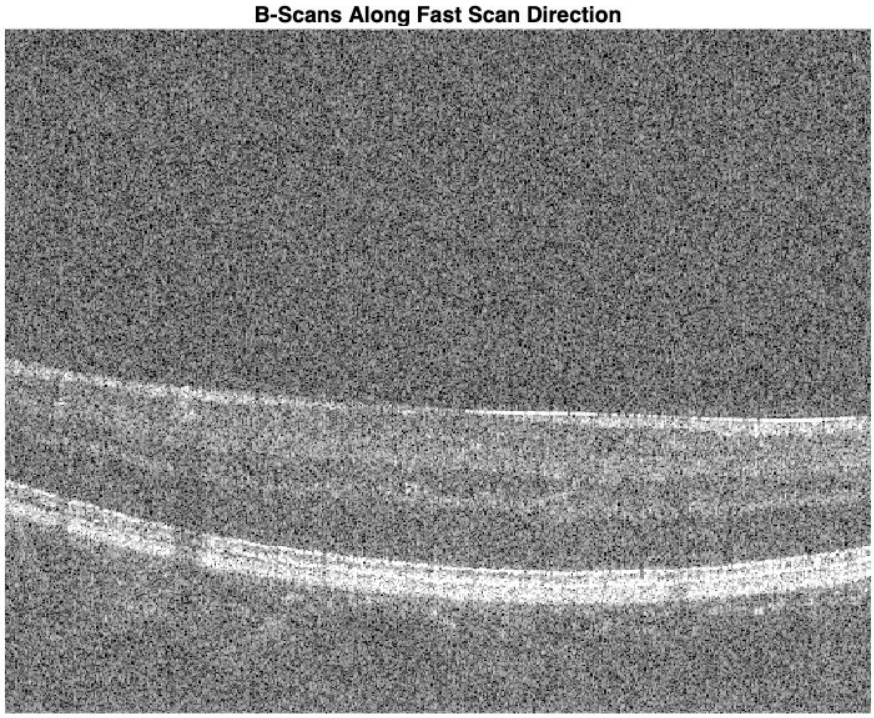
Now to generate the volume we have a mock run to verify the code, and precalculate the disperssion coefficants. Although the dispersion coefficients can be imported from a saved file, in order to make the codes self sufficient and only dependent on the raw files for allowed interchangability of source data and future proofing the code.

```
clear all; clc;

load(fullfile('./RawOCT_A.mat'));
load(fullfile('./RawOCT_B.mat'));

fileID = fopen(fullfile('./LUT_A.bin'),'r'); LUT_A = fread(fileID,'double');
fclose(fileID);
fileID = fopen(fullfile('./LUT_B.bin'),'r'); LUT_B = fread(fileID,'double');
fclose(fileID);

% Mock run to find dispersion coeeficiants once to speed up processing
CplxRawData_A = hilbert(RawOCT_A(:,:,round(end/2)));
CplxRawData_B = hilbert(RawOCT_B(:,:,round(end/2)));
CplxRawData_Rescaled_A = reSampling_LUT(CplxRawData_A, LUT_A);
CplxRawData_Rescaled_B = reSampling_LUT(CplxRawData_B, LUT_B);
CplxRawData_Rescaled_AB = CplxRawData_Rescaled_A - CplxRawData_Rescaled_B;
CplxRawData_DCSub_AB = CplxRawData_Rescaled_AB
- (repmat(median(real(CplxRawData_Rescaled_AB),2),
[1,size(CplxRawData_Rescaled_AB,2)])
+1j.*repmat(median(imag(CplxRawData_Rescaled_AB),2),
[1,size(CplxRawData_Rescaled_AB,2)]));
```

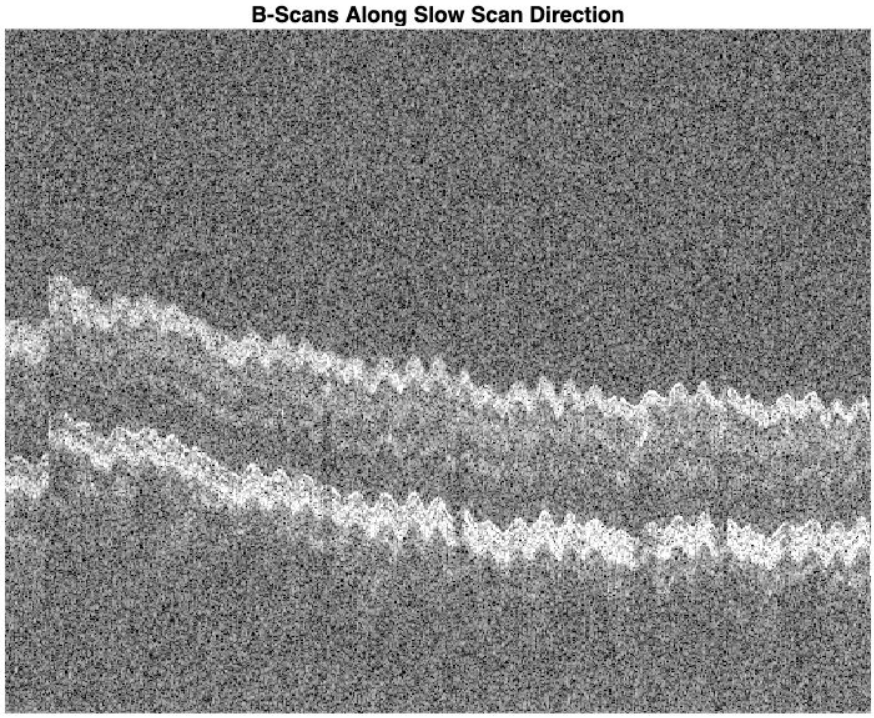
```
dispMaxOrder = 4; coeffRange = 20; depthROI = [14, 590];
[dispCoeffs, arrDispCoeff, arrCost] =
setDispCoeff(CplxRawData_DCSub_AB,depthROI,dispMaxOrder,coeffRange);

for FrameNum = 1:size(RawOCT_A,3)
    CplxRawData_A = hilbert(RawOCT_A(:,:,FrameNum));
    CplxRawData_B = hilbert(RawOCT_B(:,:,FrameNum));
    CplxRawData_Rescaled_A = reSampling_LUT(CplxRawData_A, LUT_A);
    CplxRawData_Rescaled_B = reSampling_LUT(CplxRawData_B, LUT_B);
    CplxRawData_Rescaled_AB = CplxRawData_Rescaled_A - CplxRawData_Rescaled_B;
    CplxRawData_DCSub_AB = CplxRawData_Rescaled_AB
- (repmat(median(real(CplxRawData_Rescaled_AB),2),
[1,size(CplxRawData_Rescaled_AB,2)])
+1j.*repmat(median(imag(CplxRawData_Rescaled_AB),2),
[1,size(CplxRawData_Rescaled_AB,2)]));
    CplxRawData_DispCorr_AB =
compDisPhase(CplxRawData_DCSub_AB,dispMaxOrder,dispCoeffs);
    CplxRawData_win =
CplxRawData_DispCorr_AB.*repmat(hann(size(CplxRawData_DispCorr_AB,1)), [1
size(CplxRawData_DispCorr_AB,2)]);
    Ref_FFTData_Final = fft(CplxRawData_win);
    OCT_Vol_Uncorrected(:,:,FrameNum) =
flipud(Ref_FFTData_Final(depthROI(1):depthROI(2),:));
end

save(fullfile("./OCT_Vol_Uncorrected"), 'OCT_Vol_Uncorrected', '-v7.3');

%% Save .tiff stack - Uncomment for uncorrected tiff stack.
% for i = 1:size(OCT_tcorr,3)
%     img = imadjust(mat2gray(20.*log10(abs(OCT_tcorr(:,:,i)))));
%     imwrite(img, 'OCT_Vol_Uncorrected.tiff', 'WriteMode', 'append',
'Compression','none');
% end
```

Pre Correction Volume Examination

```
clear all; clc; close all;
load(fullfile("./OCT_Vol_Uncorrected"));

% B-Scans along fast direction
for i = 1:size(OCT_Vol_Uncorrected,3)
    imagesc(imadjust(mat2gray(20.*log10(abs(OCT_Vol_Uncorrected(:,:,i))))));
    colormap('gray');axis off; title('B-Scans Along Fast Scan Direction');
    pause(0.05);
end
```




```
% B-Scans along Slow Direction
for i = 1:size(OCT_Vol_Uncorrected,2)

imagesc(imadjust(mat2gray(20.*log10(abs(squeeze(OCT_Vol_Uncorrected(:,i,:)))))));
    colormap('gray'); axis off; title('B-Scans Along Slow Scan Direction');
    pause(0.05);
end
```



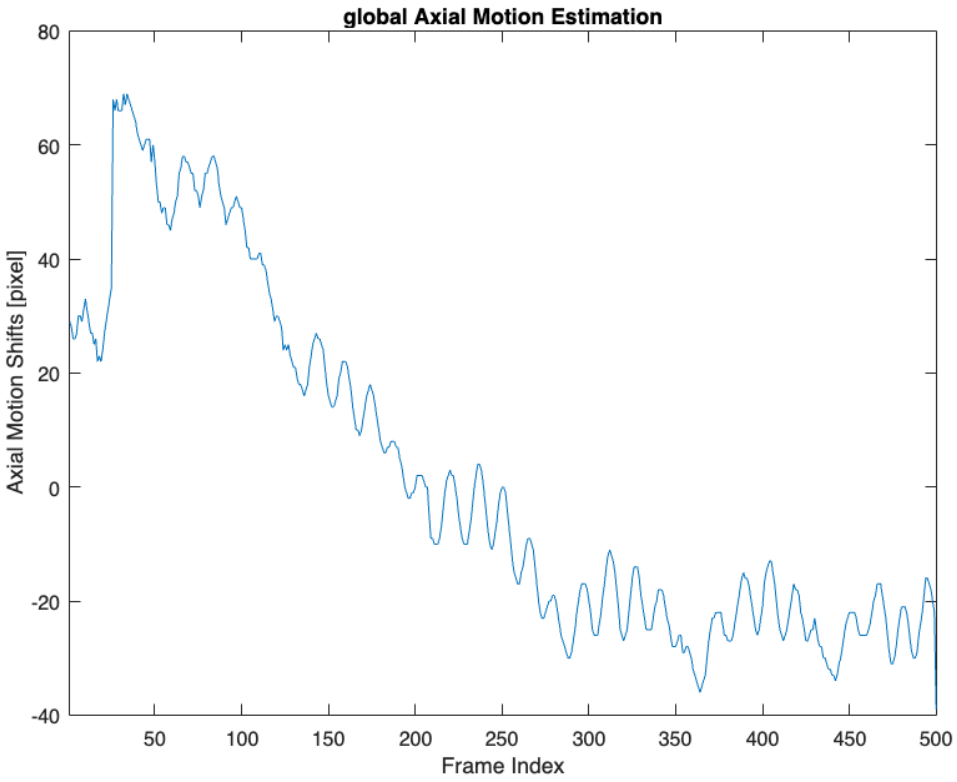
Step 3: Correcting the 3D volume.

After generation of the volume we have to correct for axial misallignment. First we perform global axial motion correction, then we perform global axial tilt correction. This combination of corrections generate a coherent and well aligned 3D volume of which en-face scans can be generated. This processing allows us to go from raw OCT data to usable, manipulable, and diagnosis ready retinal volume model.

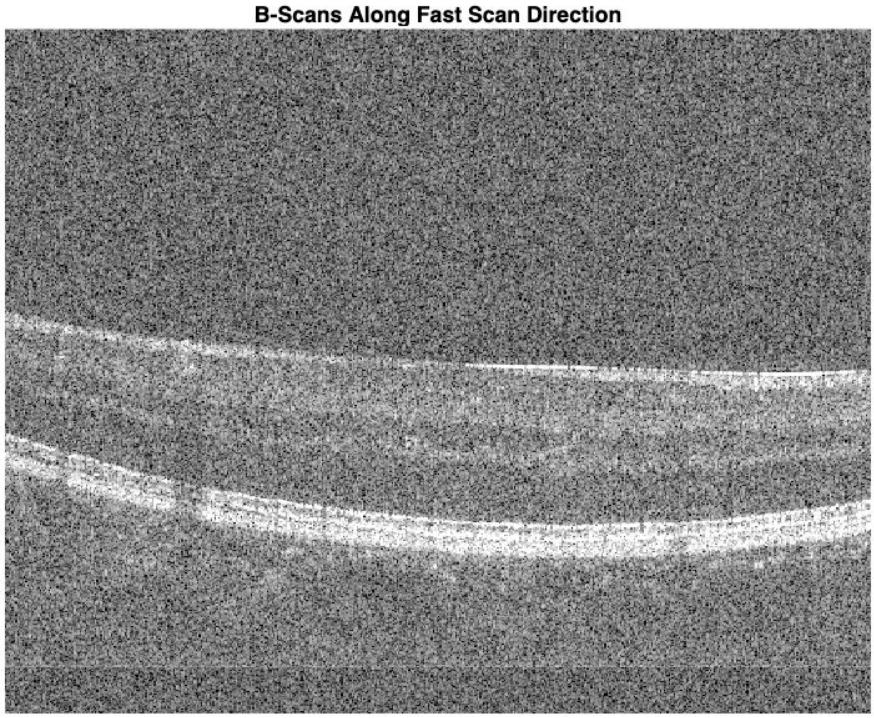
Global Axial Motion Correction

```
usfac = 1;
numFrames = size(OCT_Vol_Uncorrected, 3);
global_axial_motion = zeros([numFrames, 1]);
OCT_mcorr = OCT_Vol_Uncorrected;

for I = 1:numFrames
    % Every 'for' loop, reference frame will be the middle frame
    [output, ~] = dftregistration(fft2(20.*log10(abs(OCT_Vol_Uncorrected(:, :,
round(numFrames./2))))),fft2(20.*log10(abs(OCT_Vol_Uncorrected(:, :, I)))),
usfac);
    % Assign and save the shifting value for axial (yShift)
    global_axial_motion(I) = round(output(3));
    OCT_mcorr(:, :, I) = circshift(OCT_Vol_Uncorrected(:, :, I), [output(3), 0]);
end

plot(global_axial_motion);
xlim([1 500]); title('global Axial Motion Estimation');
xlabel('Frame Index');
```

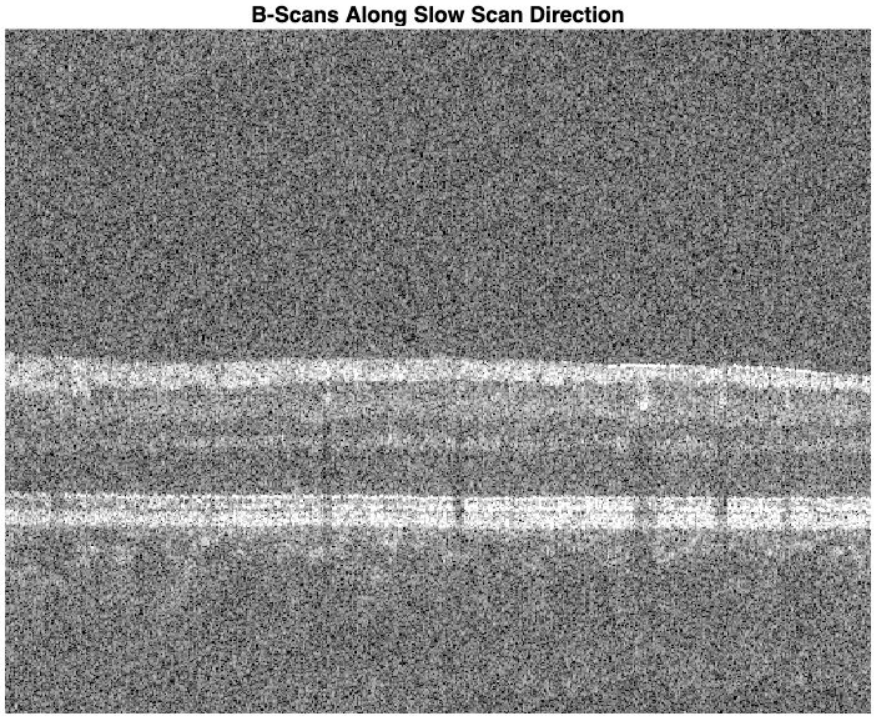
```
ylabel('Axial Motion Shifts [pixel]');
```




```
% B-Scans along fast direction
for i = size(OCT_mcorr,3)
    imagesc(imadjust(mat2gray(20.*log10(abs(OCT_mcorr(:,:,i))))));
    colormap("gray");axis off; title('B-Scans Along Fast Scan Direction');
    pause(0.05);
end
```

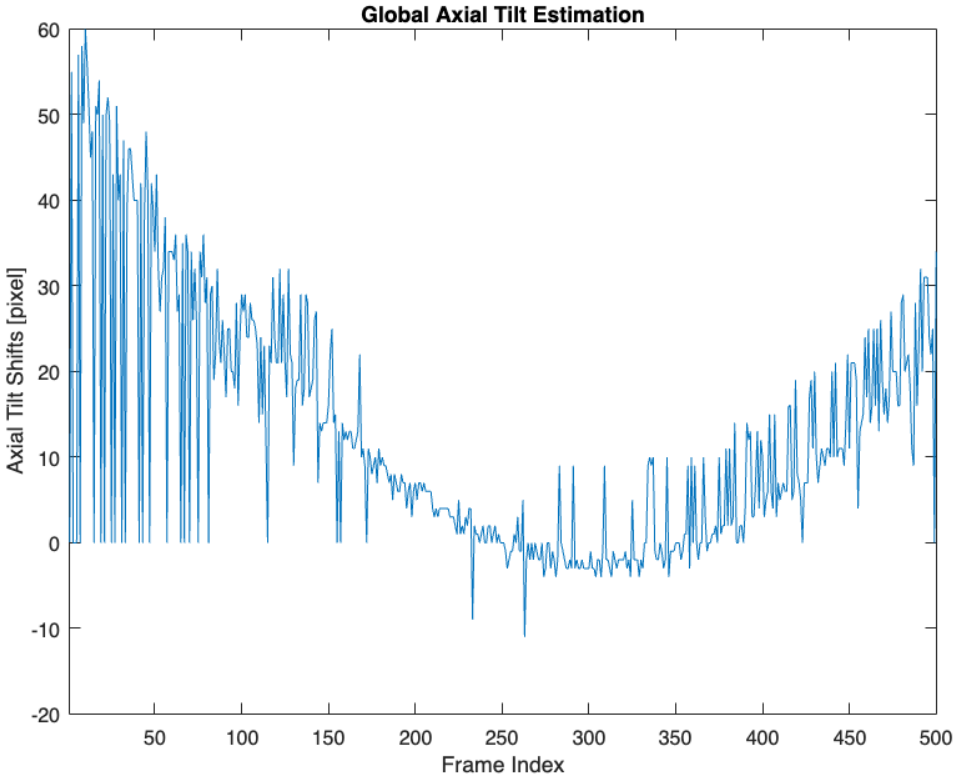



```
% B-Scans along slow direction
for i = size(OCT_mcorr,2)
    imagesc(imadjust(mat2gray(squeeze(20.*log10(abs(OCT_mcorr(:,i,:)))))));
    colormap("gray");axis off; title('B-Scans Along Slow Scan Direction');
    pause(0.005);
end
```

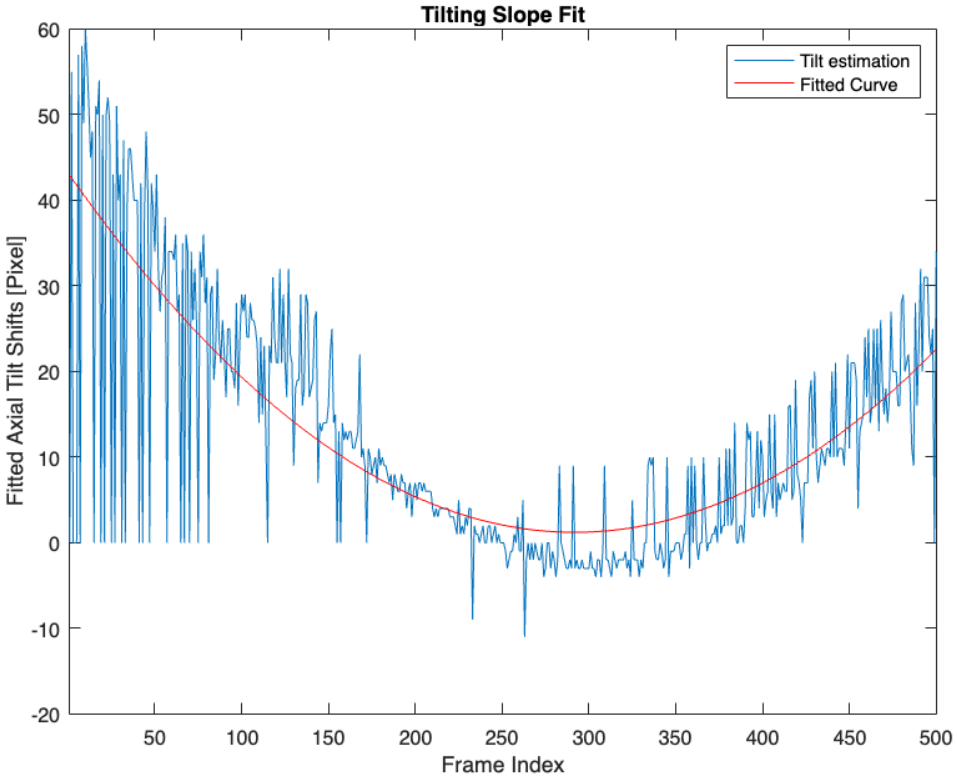


Global axial tilt correction

```
numLines = size(OCT_mcorr, 2);
global_axial_tilt = zeros([numLines 1]);

for I = 1:numLines
    % Every 'for' loop, reference frame will be the middle frame %%%
    [output, ~] = dftregistration(fft2(squeeze(OCT_mcorr(:,
round(numLines./2), :))),...
    fft2(squeeze(OCT_mcorr(:, I, :))), usfac);
    % Assign and save the shifting value for lateral (xShift) and axial (yShift)
%%%
    global_axial_tilt(I) = round(output(3));
end


plot(global_axial_tilt);
xlim([1 500]);
title('Global Axial Tilt Estimation');
xlabel('Frame Index');
ylabel('Axial Tilt Shifts [pixel]');
```

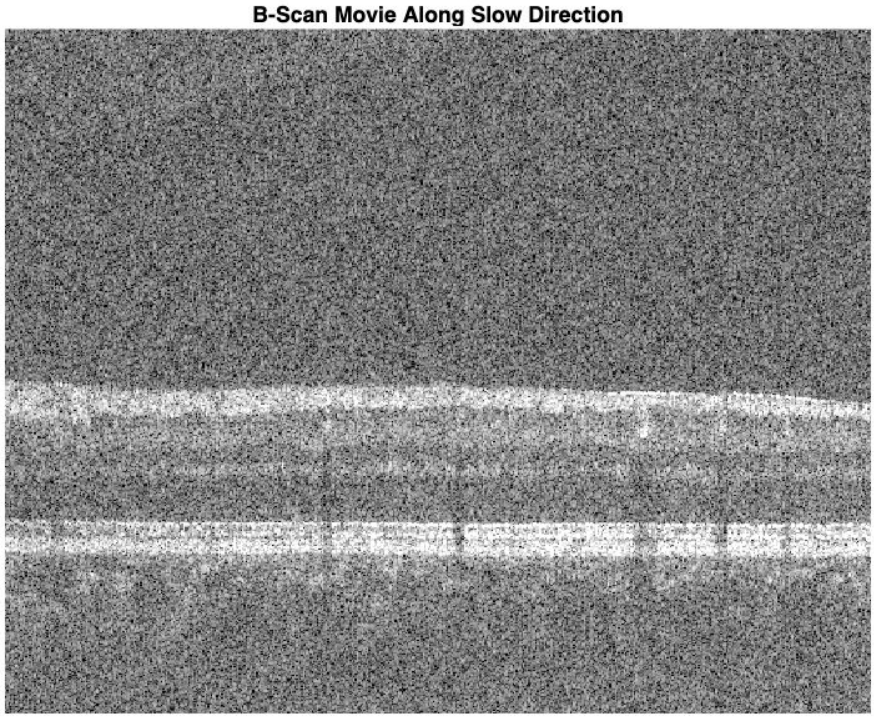



```
% Polynomial fitting the tilt curve
x = (1:numLines)';
cx = polyfit(x,global_axial_tilt,2);
global_axial_tilt_fit = polyval(cx, x);


plot(global_axial_tilt); hold on;
plot(global_axial_tilt_fit, 'r'); hold off;
legend('Tilt estimation', 'Fitted Curve');
xlim([1 numLines]);
title('Tilting Slope Fit');
xlabel('Frame Index');
ylabel('Fitted Axial Tilt Shifts [Pixel]');
```



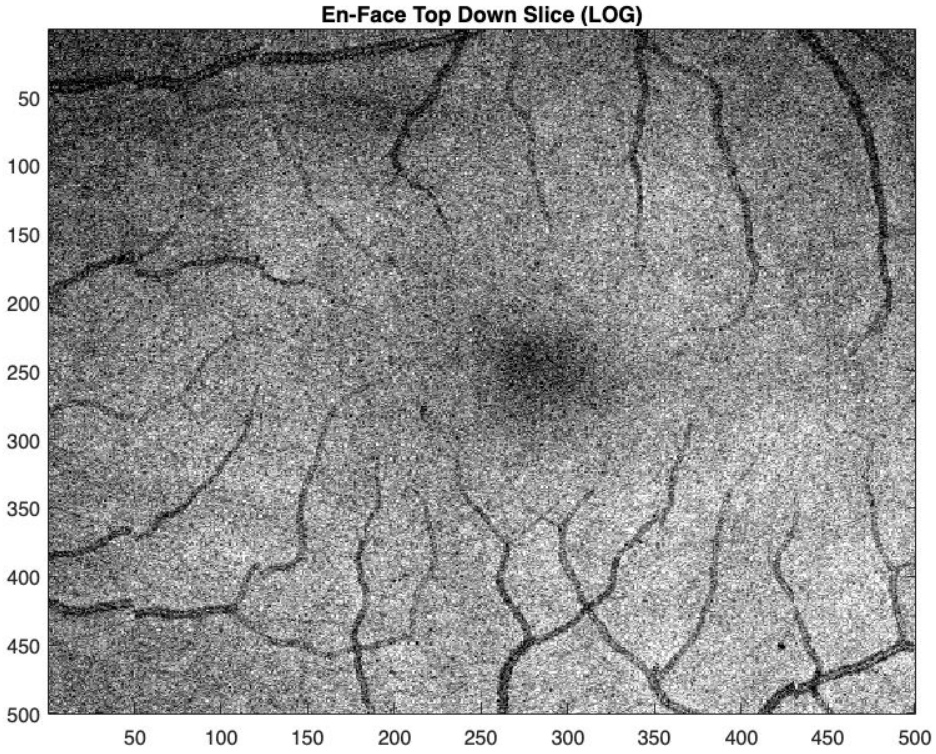

```
% Apply the fitted tilt estimation
OCT_tcorr_fit = OCT_mcorr;
for I = 1:numLines
    OCT_tcorr_fit(:, I, :) = circshift(OCT_mcorr(:, I, :),
[round(global_axial_tilt_fit(I), 0)]);
end

% B-Scans along fast direction
for i = 1:size(OCT_tcorr_fit,3)
    imagesc(imadjust(mat2gray(20.*log10(abs(OCT_tcorr_fit(:,:,i))))));
colormap("gray"); axis off;  title('B-Scans Along Fast Scan Direction');
    pause(0.005);
end
```


```
% B-Scans along Slow direction
for i = 1:size(OCT_tcorr_fit,2)
    imagesc(imadjust(mat2gray(20.*log10(abs(squeeze(OCT_tcorr_fit(:,i,:)))))));
colormap("gray"); axis off; title('B-Scan Movie Along Slow Direction');
    pause(0.005);
end
```

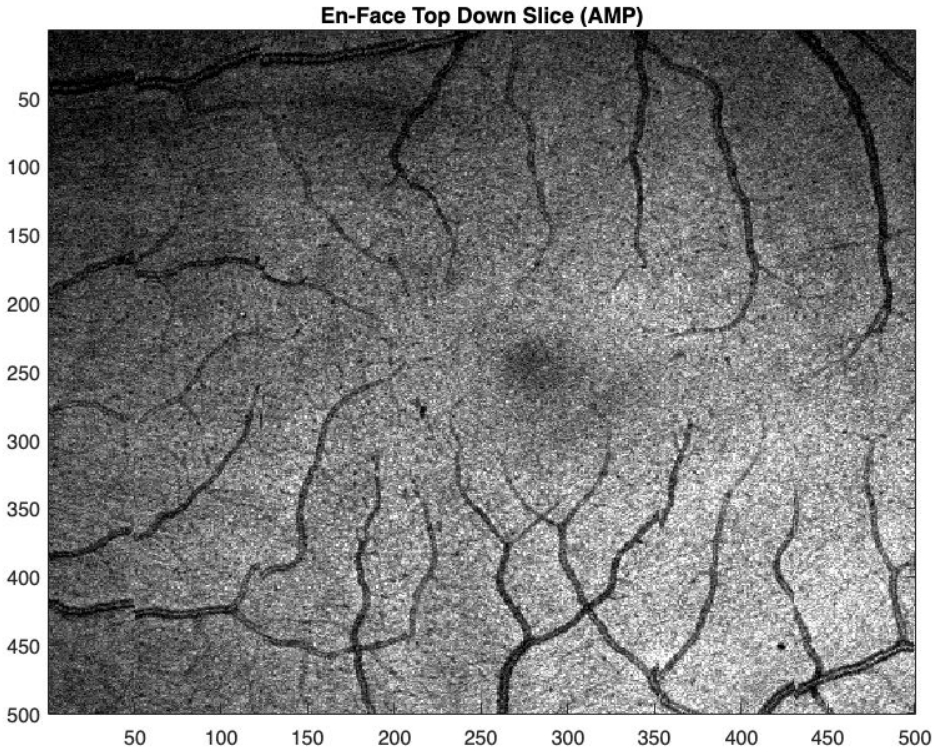


Results: Examining Generated En-Face data and Generate Final Coreected Volume.

```
% En-face Image Generation.
depthROI = [14, 577];
OCT_Vol_Log = 20.*log10(abs(OCT_tcorr_fit));
OCT_Vol_Amp = (abs(OCT_tcorr_fit));

Enface_Img_Log = squeeze(mean(OCT_Vol_Log(depthROI(1):525,:,:),1));
Enface_Img_Amp = squeeze(mean(OCT_Vol_Amp(depthROI(1):depthROI(2),:,:),1));


figure()
imagesc(imadjust(mat2gray(Enface_Img_Log))); colormap("gray"); title('En-Face
Top Down Slice (LOG)');
```




```
figure()
imagesc(imadjust(mat2gray(Enface_Img_Amp))); colormap("gray"); title('En-Face
Top Down Slice (AMP)');
```



```
% Generate Final tiff Stack Of The Corrected Volume.
for i = 1:size(OCT_tcorr_fit,3)
    img = imadjust(mat2gray(20.*log10(abs(OCT_tcorr_fit(:,:,i)))));
    imwrite(img, 'OCT_Corrected.tiff', 'WriteMode', 'append',
'Compression','none');
end

% Generate Final .mat Volume.
save(fullfile("./OCT_Vol_Corrected"), 'OCT_tcorr_fit', '-v7.3');
```