UNIVERSITY OF CALIFORNIA
RIVERSIDE

Progress Towards Detecting Neural Activity in Optical Coherence Tomography Using Phase

A Thesis submitted in partial satisfaction
of the requirements for the degree of

Master of Science

in

Bioengineering

by

Jamsine Kavan Shah

December 2017

Thesis Committee:
Dr. Hyle Park, Chairperson
Dr. William Grover
Dr. Jin Nam

The Dissertation of Jasmine Kavan Shah is approved:

_____

_____

_____

Committee Chairperson

University of California, Riverside

ABSTRACT OF THE THESIS

Progress Towards Detecting Neural Activity in Optical Coherence Tomography Using Phase

by

Jasmine Kavan Shah

Master of Science, Graduate Program in Bioengineering
University of California, Riverside December 2017
Dr. Hyle Park, Chairperson

Optical Coherence Tomography (OCT) is non-invasive, real time optical imaging method based on low coherence interferometry with high resolution. It is capable of imaging microstructure by measuring the light backscattered from the sample. Under the AGI grant, I have built a Swept Source OCT (SS-OCT) / phase resolved OCT to detect the nanometer(nm) scale changes in the cell membrane that occur during membrane depolarization and Ion influx. This type of small changes can be seen using phase. Intensity is robust but is less sensitive to the small-scale changes whereas phase is highly sensitive. The noise level for phase difference quantification depends on the lateral motion, triggering of the wavelength mismatch and signal to noise ratio (SNR). To detect the nm scale changes phase noise should be minimum. The phase noise is inversely proportional to SNR. The SNR of the system should be maximum. The SS-OCT has swept source with sweep rate 100 kHz and it is not phase stable. It needs post processing algorithm to match the

triggering wavelength to stabilize the phase. I wrote the code in GPU to reduce the computation time for post processing algorithm. Due to a few setbacks related to moving the system to a collaborating lab, the biological portion of my thesis work, trying to find the neural activity in the walking leg nerve of the Lobster using phase, was done with a spectral domain OCT (SD-OCT). To see any action potential change in the nerve I realigned the SD-OCT system with center wavelength to increase the SNR of the system and the sample arm with the nerve chamber (3D printed) to stimulate the nerve at one end and take the OCT image at the other end to see the action potential.

Keywords: swept source OCT, spectral domain OCT, neural activity, phase sensitivity, lateral motion, signal to noise ratio.

Table of Contents

List of Figures

Chapter 1

Chapter 2

Chapter 4

Chapter 1. Introduction to OCT

1.1. Introduction

Optical Coherence Tomography (OCT) is non-invasive imaging technique capable of high resolution images of tissues. The first biological application of OCT was reported by Huang et al. in 1988, for the measurement of the axial length of the eye. OCT performs cross - sectional imaging by measuring the magnitude and echo time delay of backscattered light from the tissues. Figure 1.1. illustrates how OCT compares in resolution versus imaging depth to other common imaging techniques used in medical and research fields. The standard clinical imaging techniques like functional Magnetic Resonance Imaging (fMRI),

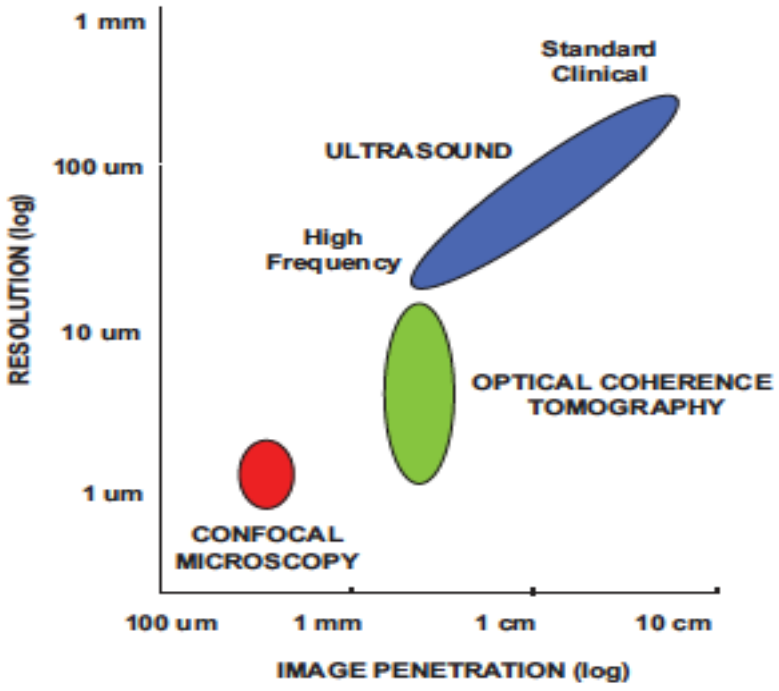


Figure 1.1. Comparison of resolution and imaging depth for standard clinical imaging(fMRI), ultrasound, OCT, confocal microscopy. [1]

Computed Tomography (CT-scan) have greater depth penetration (centimeters) and can image entire body, but has limited spatial resolution (millimeters). The standard ultrasound can image deep structures because of the minimal absorption in tissues. The high frequency ultrasound helps to image only few millimeters in the body with the low resolution (15μm -20μm) because higher frequencies are highly attenuated in biological tissues which reduces the penetration depth of the imaging. Confocal microscopy has extremely high resolutions(~1μm). It is typically used for *en face* imaging because resolution is determined by the diffraction limit of light [1]. The gap between ultrasound and confocal microscopy is filled by OCT. OCT provides the axial resolution from 1 μm -15μm and imaging depth of 2-3mm. It has ability to penetrate enough and create a depth resolved images non-invasively.

1.2. Principle of OCT

OCT is a method based on low-coherence interferometry. Interferometry is a powerful technique for measuring the light that is backscattered from the tissue and the light that has traveled a known distance or time delay through a reference path. It is used to detect intensity and optical path delay in OCT. Interferometry generates interference patterns between light reflected back from the known reference path and backscattered from the tissue. OCT is based on Michelson-Interferometer. In Michelson Interferometer, light from the source is divided into the reference

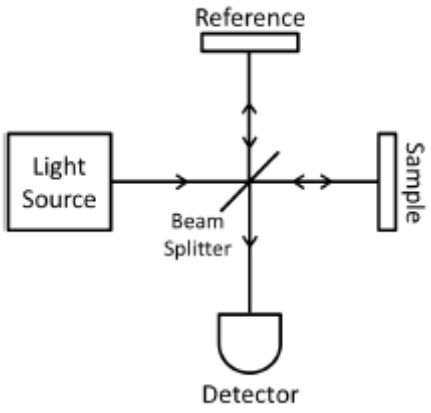


Figure1.2 Schematic of Michelson Interferometer used in OCT

path $E_R(t)$ and the sample path $E_S(t)$. There is always a small difference in the path length for reference arm and the sample arm. Before reflecting back, light is interfered and then sent to the detector. The output of the interferometer at the detector is the sum of the sample and the reference fields. Interferometer measures the field of light rather than its intensity [1]. The intensity (I_O) at the detector is proportional to the square of the total field:

$$I_O \sim |I_R|^2 + |I_S|^2 + 2\sqrt{I_r I_s}\ cos(2k\Delta L)$$

k is the wavenumber of the wavelength coming from the source and ΔL is the path difference between the sample and the reference path. The coherence is the constant phase difference at the same frequency. In OCT, optical echoes are detected at low coherence (broad-bandwidth) light source. To ensure the interference or phase discontinuities, path difference should match the coherence length which is inversely proportional to the frequency bandwidth of the light source.

1.3. Optical Coherence Tomography

Time Domain-OCT (TD-OCT) was the first-generation OCT. In TD-OCT, reference arm is scanned to obtain the depth profile (A-scan) from the sample. By transverse scanning where the

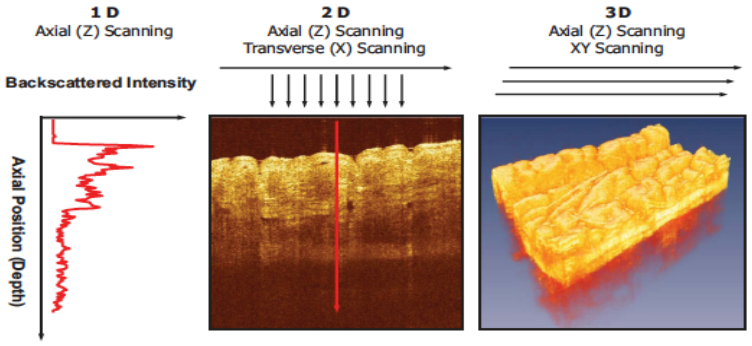
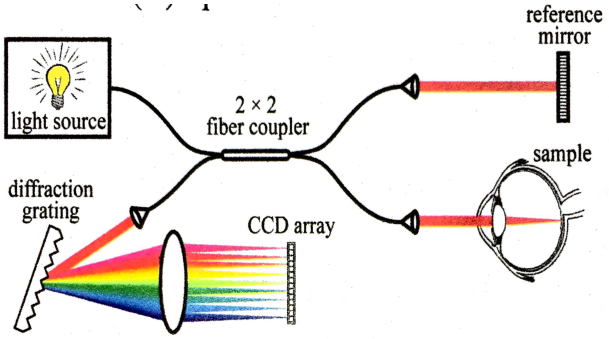


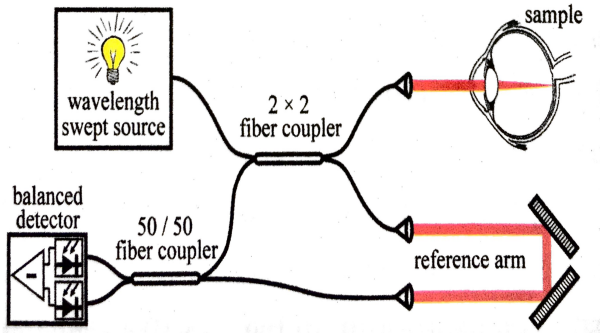
Figure1.3 OCT images and volume construction **(a)** Backscattered Intensity (dB) Versus Axial position graph. It shows the amount of light penetrating to certain depth. (**b)** 2D scan (Transverse scan) is cross sectional image consisting specific number of A-lines. **(c)** 3D scan (Raster Scan) is the volume of consecutive cross-sectional images.

incident beam across the sample and collecting different depth profiles 2D cross sectional images are performed. 3D imaging is performed using raster scanning where the beam across the sample takes the continuous 2D images (Figure1.3). However due to limited sensitivity and physical limitations of the mechanical scanning of the reference mirror, making it difficult to take phase sensitivity data. The next generation Fourier domain OCT (FDOCT) offers significantly improved sensitivity and imaging speed compared to TD-OCT. FD-OCT detection can be performed in two ways: Spectral Domain OCT (SD-OCT), which uses broadband light source and a spectrometer with line scan cameras and Swept Source OCT (SS-OCT), using a broadband narrow pulse swept source laser [2]. SS-OCT

is also known as Optical Frequency Domain Imaging (OFDI). In SD-OCT the photodetector of the TD –OCT is replaced by spectrometer as shown in figure 1.4. The spectrometer uses a diffraction grating for an angular separation of wavelengths and a lens is used to image them as multiple small wavelength bands on a charge coupled device (CCD) detection array.



(a) Spectral-Domain OCT



(b) Swept Source OCT

Figure 1.4. (a) Schematic of spectral domain OCT setup which uses spectrometer for the detection of spectrally resolved interference signals. **(b)** Schematic of swept source OCT setup which uses rapidly wavelength swept light source and balance detector for the detection of spectrally resolved interference signals [5].

In SD-OCT, an entire A-scan measurement is acquired in “Single Shot” on the full CCD array. Whereas in SS-OCT, the light source of TD-OCT is replaced with a narrow bandwidth laser source that is rapidly swept in time over broad spectral bandwidth. In the figure 1.4. a generic setup of SS-OCT is shown that incorporates reference arm transmission and a balance detector. In SS-OCT balance detector is highly advantageous to remove the common mode signals, in particular the relative intensity noise of the swept source, which is in general much stronger than for SD-OCT light sources. In SS-OCT an entire A-scan is acquired by “a single wavelength sweep” over the full spectral bandwidth of swept source [5]. The applications related to OCT mostly involves extracting only the amplitude of the interference signal that is intensity, to obtain the resolution on
the order of microns. The phase information is also available, but it is highly susceptible to small changes. It is possible to record nm-scale changes in the tissues after achieving sufficient phase stability. The phase difference between successive A-lines, it is essential that the phase measurement is repeatable from one A-line to the next. The changes in the measured phase results from the systematic or interferometric instabilities increase the phase noise of the system. In SD-OCT, the inherent of the source, interferometer and spectrometer enables highly repeatable phase measurement and correspondingly high sensitivity. In OFDI, variations in the timing of the wavelength for each sweep in the swept source relative to the acquisition electronics can induce variations in the phase measurement that degrades the sensitivity [3]. Because of the superior phase stability and can detect sub-nanometer range optical path length change SD-OCT was used to measure the transient structural changes (fast (ms) and small (nm) that are directly related to the

Action Potential(AP) at all depth points from nerve bundles dissected from lobster legs using phase interferometry [4].

Chapter 2. Swept Source OCT/ OFDI

2.1. Light source

The light source in the range of 1060nm shows the better penetration in the deeper retinal layers due to lower scattering at the slightly reduced axial resolution. The anterior segment of the eye is easily accessible for OCT measurement and is minimally affected by light absorption due to aqueous content in ocular. But the posterior segment of the eye is mostly aqueous which attenuates the OCT light by water absorption. The deep penetration of 1060nm light allows to image choroidal blood vessel network. The other reason for choosing specifically this wavelength is with 800nm wavelength will stimulate the photoreceptors as it is visible light and 1310 nm wavelength will easily absorb in water and posterior segment of eye is aqueous. This is of particular interest in diagnosing ocular diseases like Age related macular degeneration, Glaucoma [7-9]. The OFDI system uses swept source laser (Axsun Technologies Inc., MA, USA) with center wavelength 1060nm and wavelength tuning range 106 nm. The depth resolution for this light source is 8.4μm. The source is unidirectional (short to long wavelengths) occurs at sweep rate 100KHz and sample duty cycle of 50%. After every sweep rate, there is dead time and during that laser gain element is turned off to prevent wavelength sweep in the opposite direction as the source scans back to its initial position. The A-scan rate is determined by the sweep speed of the swept source laser. The average output power for this laser source is 15mW.An electronic sweep trigger is applied by the laser to trigger the acquisition of individual A-lines [5]. The mechanical wavelength tuning in laser source creates the small variations in

wavelength sweeps, triggering time and sampling which adversely affects the reproducibility of interference fringes.

2.2. Detector

In swept source OCT, point scan detection (m-scan) method is used where OCT will scan at the one fix position over a time also known as m-scan. With this point scanning technique and sweeping different range of wavelengths over a fix position gives explicit information at different depths for that position which helps in diagnosing various diseases. Unlike the SD-OCT in swept source OCT dual balance detection is used instead of spectrometer. The dual balance detection increases dynamic range and cancel the common mode noise. Figure 2.1 is the schematic of phase–resolved OCT system consisting of swept source laser coupled with fiber based interferometer. Light emitted from the swept source laser is split into a sample arm and a reference arm by a 25/75 fiber-coupler. The sample arm light is directed to the tip-tilt mirror in a lens to image the posterior segment of the human eye. A 90/10 fiber-coupler is used to further split the 90% of the reference arm light to the reference arm board through an air gap to match the optical dispersion of the sample arm. A diaphragm was placed in the air gap to control the amount of reference light in

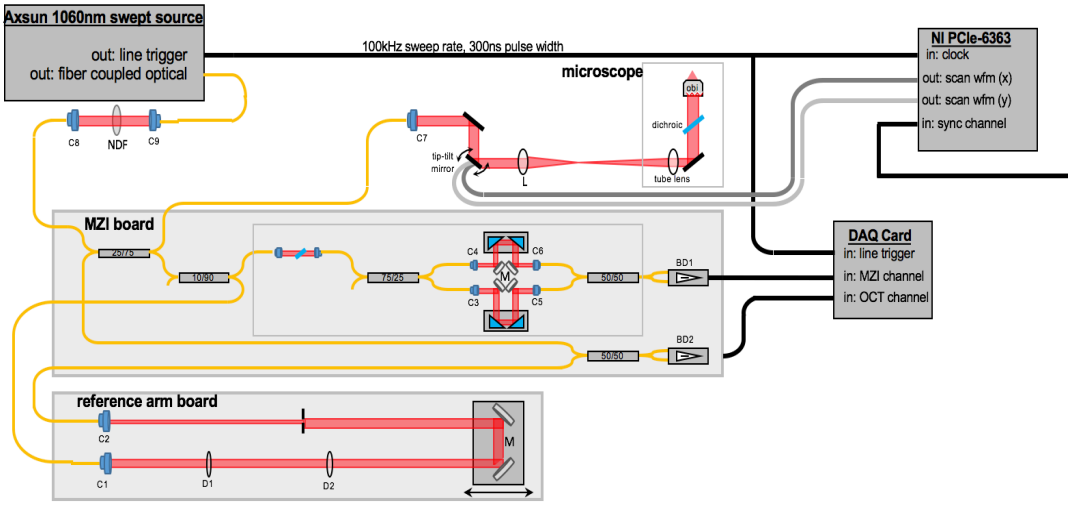
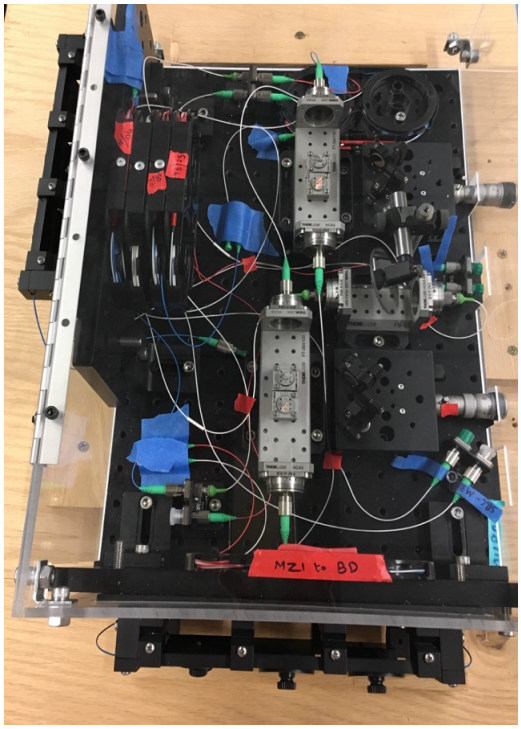
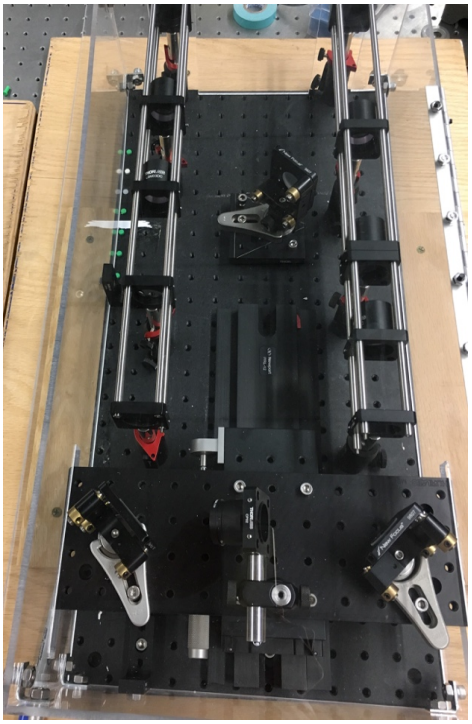


Figure2.1. Experimental setup for the phase resolved OCT system.C1-C9: Collimator, M: Mirror, BD1-BD2: Balance detector, D: Diaphragm, NDF: Neutral Density Filter.



MZI board



Reference arm

detection. The 10% of the reference light is coupled to the Mach-Zehnder interferometer (MZI) which creates a stable interference fringe to measure non-linear wavenumber sampling and to phase stabilize each laser sweep [5]. The reference and sample arms are recombined with 50/50 fiber-coupler for dual balance detection of the interference signals by dual balance photo receiver (1837 Nirvana Auto-Balanced Fiber Optic Receiver, 900-1650nm). In MZI light was split using 75/25 fiber coupler into two arms each passing light through air gap before the light was detected at the 50/50 coupler and auto balance photo receiver. There is small optical path difference in MZI to create an interference.

2.3. Characterization for phase instability

Instabilities in the swept source laser, the interferometer and data acquisition hardware result in a shift in the wavenumber (k-space) for subsequent A-lines [6]. In order to obtain phase stable data, a signal resampling procedure is used. To stabilize the phase from the MZI phase– stabilization algorithm was written in Matlab. The basic structure of resampling wavenumber is extracting raw data from MZI to clean the noise from the spectra and do the Fourier transform to find the peak. Apply inverse Fourier transform and unwrap the phase angle for all the A-lines. The corrected unwrapped phase shows the phase stabilization between A-lines. Figure 3.2. shows the exaggerated example using 10 spectra of the MZI that light source of this SS-OCT before and after the correction using the phase stabilization algorithm. It can be clear seen from those graphs how instable swept source is and it can be stabilized to take the phase data.

In this SS-OCT system the wavenumber for individual A-line is simultaneously recorded by Mach-Zehnder Interferometer (MZI). Figure 2.3 shows schematic diagram of MZI which is similar to the Michaelson interferometer except in MZI light pass through both the reference and sample arm only once. There is a small path difference in both the arms of the MZI to create an interference. This path difference will help to calibrate the phase shift in each wavenumber. MZI creates a stable interference fringe to measure the non-linear wavenumber (K-space) and to phase stabilize each laser sweep. The fix path difference in the MZI can detect the wavenumber for specific wavelength Phase stabilization for large set of A-lines is achieved by using the reference MZI signal with respect to all other A-lines.

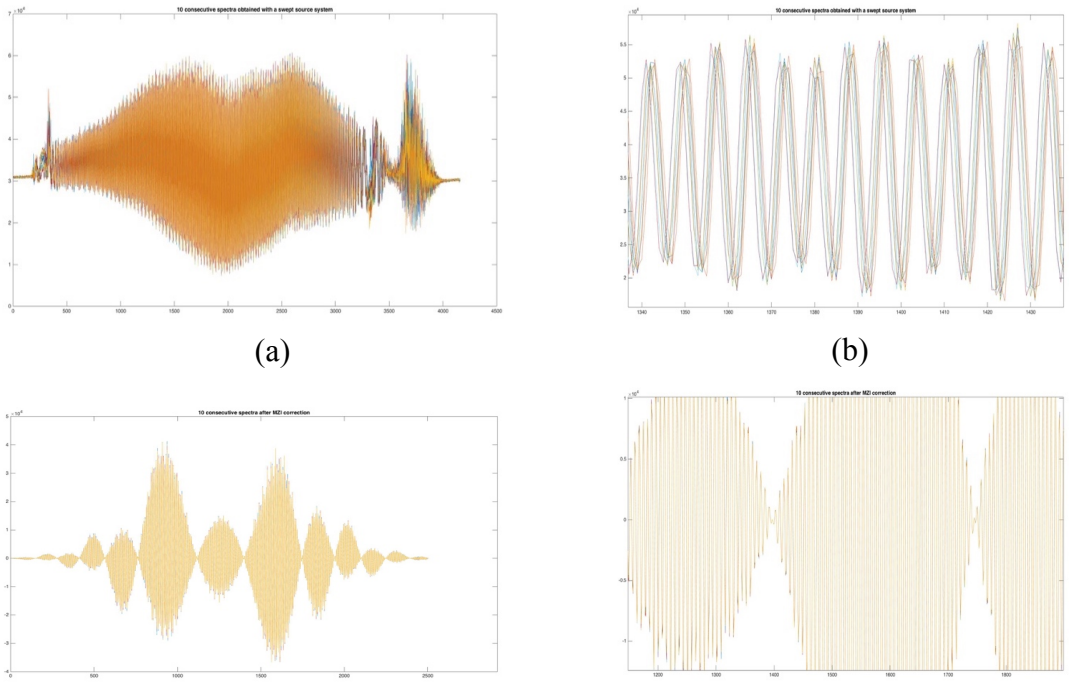


Figure 2.2. (a) 10 consecutive spectra from MZI before any phase correction **(b)** Shows the zoom in of those 10 consecutive spectra which are not phase stable. (c) 10 consecutive spectra from MZI after the phase correction using post processing algorithm **(d)** Shows the zoom in 10 consecutive spectra from MZI which are now phase stable after correction using post processing algorithm

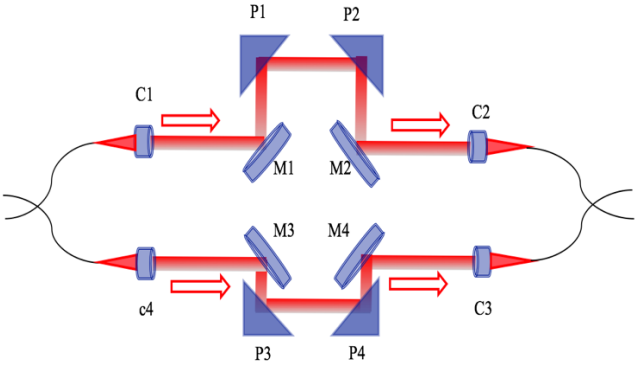
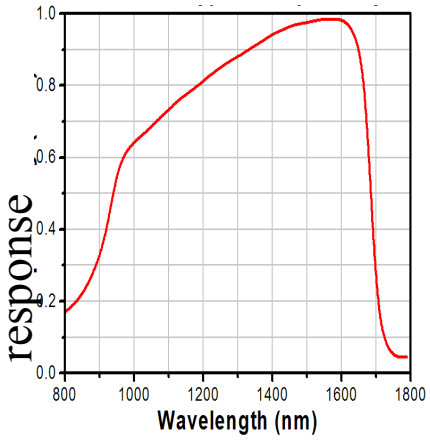
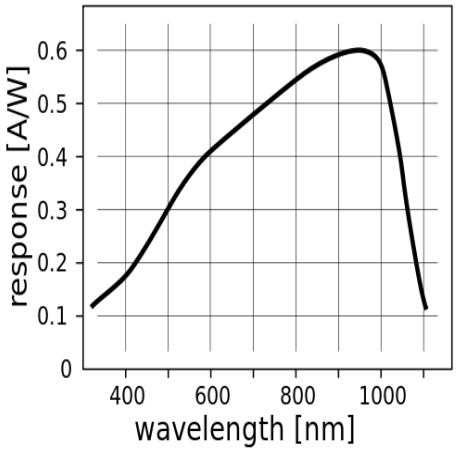


Figure 2.3. Schematic for Mach-Zehnder Interferometer.C1-C4: Collimators, M1-M4: Mirrors, P1-P4: Prisms.

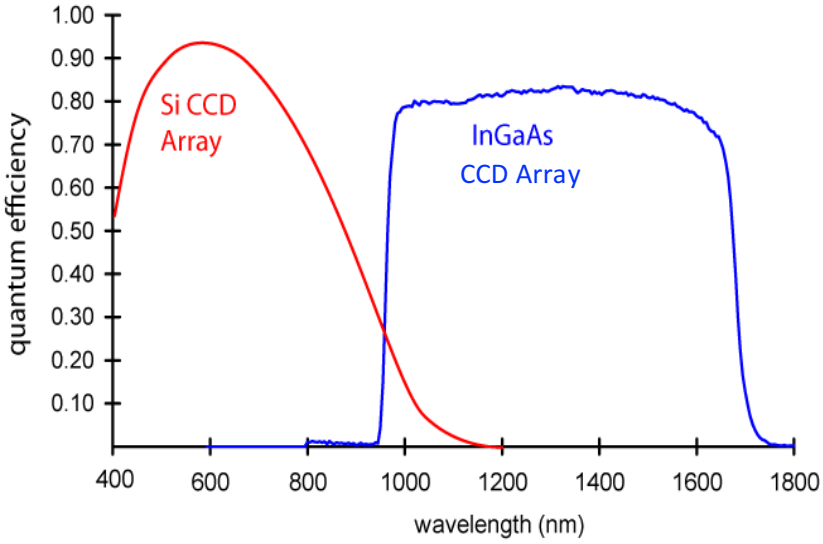
In the phase stabilities discussion of SD-OCT it was shown that spectrometer with line scan camera and diffraction grating is more stable. The reason for using dual balance detector in SS-OCT is line scan cameras are only available for certain range as shown in the figure 2.4. Fundamentally, a charge coupled device (CCD) is an integrated circuit etched onto a silicon surface forming light sensitive elements called pixels. Photons incident on this surface generate charge that can be read by electronics and turned into a digital copy of the light patterns falling on the device. The line scan cameras come with the two types of surface elements to detect the light silicon (Si) and Indium Gallium Arsenide (InGaAs). The CCD array with Si detector can detect from 400nm -1000 nm of light and after 1000 nm response of the wavelength starts dropping. InGaAs detector of CCD array can detect from 800nm-1800nm of light. Even when InGaAs has the required wavelength range it is varying not stable. From the figure2.4. (c) it can be clearly seen the range of the swept source (985nm-1200nm) has the least quantum efficiency.



(a) InGaAs photodetector



(b) Si photodetector



(c) Quantum efficiency of Si and InGaAs photodetector

Figure2.4. (a) Graph showing the response of InGaAs photodetector for the range of wavelengths(b) Graph showing the response of Si photodetector for the range of wavelengths. (c) Graph showing the quantum efficiency of InGaAs and Si photodetector for the range of wavelengths.

2.4. Post Processing Algorithm in Graphics Processing Unit (GPU)

With the advances in hardware the line acquisition rates of the swept source system have reached to 2.5 million Hz [13]. However, the heavy computational load is required to process the acquired real time data if the system has to be used for clinical purpose. A GPU has many stream processors that allows programmers to exploit the inherent parallelism. A GPU is specially designed electronic circuit for rapidly manipulating and altering memory to accelerate the building of images. The term GPU was popularized by NVIDIA in 1999, who marketed the GeForce 256. Figure 3.5 is an illustration of the difference between CPU and GPU [14]. A CPU is composed of only few cores with lots of cache memory that can handle few software threads at a time. On the contrary GPU is composed of hundreds of cores that can handle hundreds and thousands of threads simultaneously. GPU is specialized for highly parallel computation while a CPU has few cores only optimized for serial processing. Figure 2.5. shows the structure difference between CPU and GPU.

I was using Tesla K20c GPU with 13 Multiprocessor, MaxThreadPerBlock [1024 1024 64] and total memory of 4.9556e+09. I wrote the post processing code in GPU to reduce the computational time. Even after converting the code to GPU it was showing more time compare to CPU. I found that nested for loop structure was one of the reasons of taking more time for GPU because GPU does parallel processing. To eliminate this reason for more computational time I wrote the testing code with only two for loops to check with various A-lines and varying number of loops to check

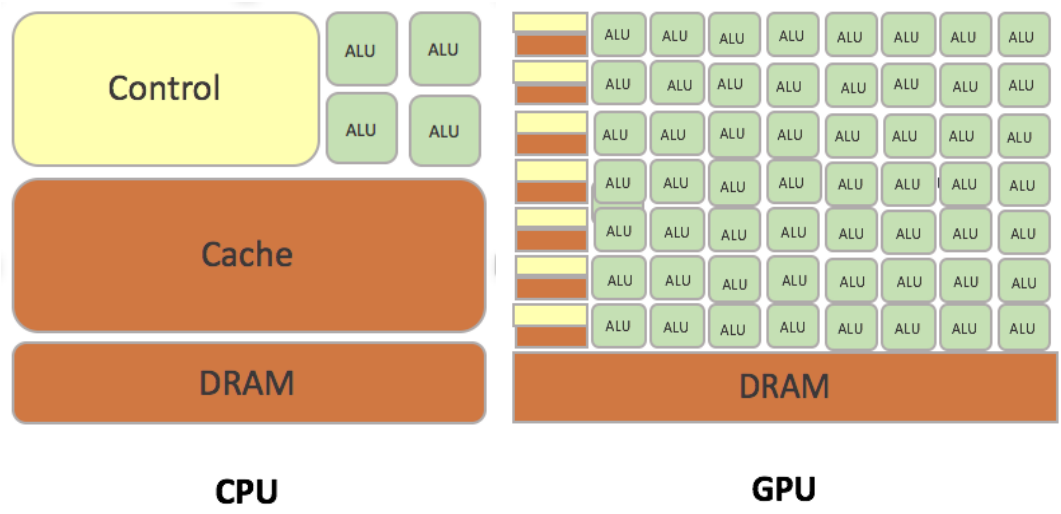


Figure 2.5. Structure difference between CPU and GPU. ALU: Arithmetic Logic Unit, DRAM: Dynamic Random-Access Memory, Cache: smaller and faster memory to store data, Control: directs the operation of processor

the threshold of the GPU when it's compiling time starts dropping compare to CPU. But that was also not the reason. Figure2.6 shows the comparison of GPU time and CPU time. Here this graph shows only ten loops but even after repeating the loop thousand and more than thousand times the looks same.

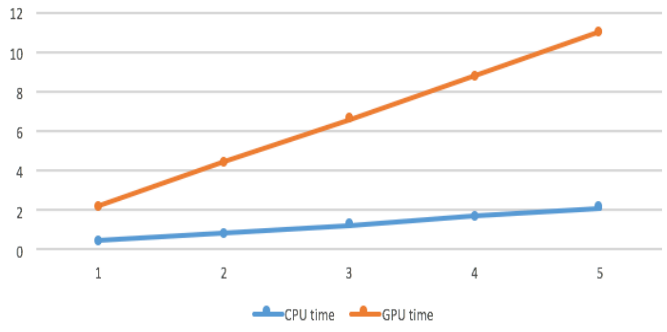


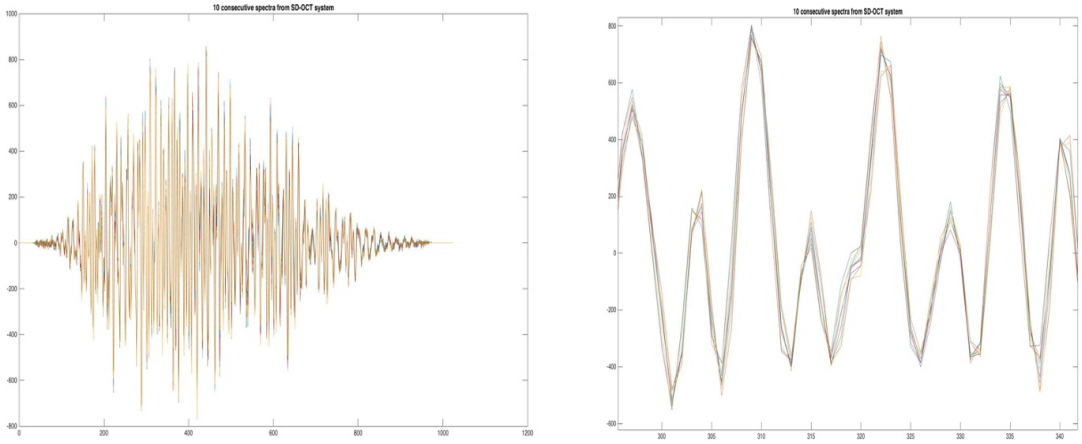
Figure 2.6. Comparison of GPU time and CPU time

So, the Final conclusion was it was not worth using GPU if the computation is not lengthy enough because transferring time from CPU to GPU and back to the CPU take sufficient amount of time which cannot be ignored. As this project was in collaboration with Stanford University and UC Berkeley under AGI grant it was moved to Stanford University. After moving the system over there it has few problems so I have to redirect my focus for MS research to look for phase noise sources and trying to see the action potential change in walking leg nerve of the Lobster and for that I have used 1310 nm Spectral domain OCT.

Chapter 3. Spectral Domain OCT

3.1. Light source

A superluminescent diode (SLD) is highly stable edge emitting semiconductor light source based on super luminescence or Amplified Spontaneous Emission (ASE). The center wavelength of this laser source is 1310nm. To illustrate the stability of the SLD laser source figure 3.1 shows the 10 spectra before any correction. From graph, it is seen that all the spectra are in phase.



(a) **(b)**

Figure 3.1. (a) 10 spectra before any correction (b) Expanded spectra to check the phase stability.

3.2. Spectrometer

The experimental setup of spectrometer based SD-OCT system as shown in figure3.2. This setup used the SLD laser. This system was based on Michaelson Interferometry. A 2x1 beam coupler was used to direct the light from the source to split it in the reference arm

and sample arm. The optical circulator was used in the setup to separate the signals travelling back in the opposite directions. It is designed in such a way that light entering any port exits from the other. This means that if light enters port 1 it is emitted from port 2, but if some of the emitted light is reflected back to the circulator, it does not come out of port 1 but instead exits from port 3. At the detector end two lines scan camera (SUI) with 1024 pixels and 400kHz acquiring speed. The polarization controller was used to adjust the spectral power. I build the sample holder to hold and scan the

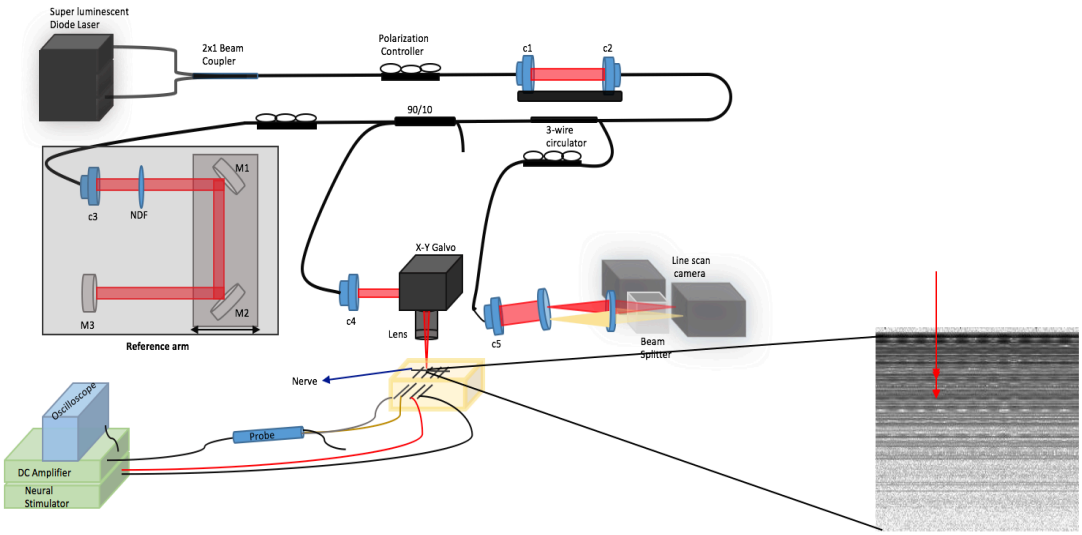


Figure 3.2. Schematic of spectral domain OCT system. c1-c5: collimator, NDF: Neutral Density Filter, M1-M2: Mirror. Image shows the nerve bundle.

Walking leg nerve of the Lobster. This sample holder consists of nerve chamber, neural stimulator, DC Amplifier and oscilloscope. To dissect the walking leg nerve of the Lobster I used Furusawa et al method. Figure 3.3 shows the walking leg nerve of the lobster was 1 inch long and diameter less than 1mm. Figure 3.4 shows the two nerve chambers I made.

Initially I made the nerve chamber from the Plexi glass which was (4x2x2) inch and water proof. But I have to make another nerve chamber which was smaller than the Plexi glass (3x1x1.5) inch and I made with the 3D printer using Solidworks to design it because walking nerve was 1inch long with diameter less than 1mm.

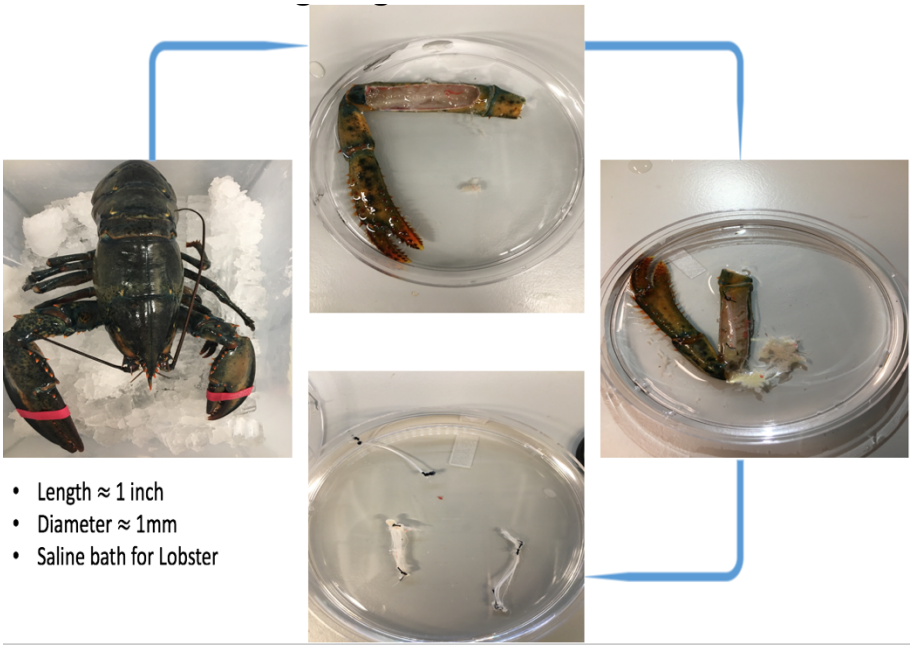


Figure 3.3. Dissection of the walking leg nerve of the Lobster

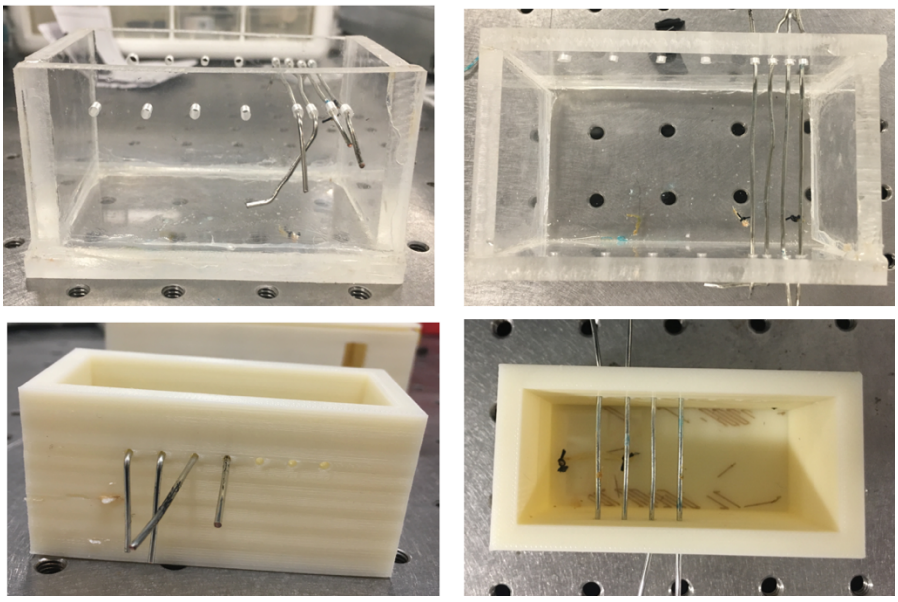


Figure 3.4. (a) Chamber at the Top is made using Plexi glass material and the copper wire with silver coating as electrodes. (b) Bottom chamber is 3D printed with water proof material and the same copper wire with silver coating as electrodes

To keep the nerve alive after dissecting and suture saline solution (525 nM $NaCl_2$, 13.3 mM KCl, 12.4 mM $CaCl_2$, 24.8 mM $MgCl_2$ and 5 mM dextrose) is used[10]. Electrodes in the chamber were copper wire with silver coating used as electrodes to stimulate walking leg nerve of lobster. The isolated nerve helps to see the action potential. The interference detected at the spectrometer after the light travelling back from the sample and reference arm has the information in the form of phase and intensity. The intensity information is available in the envelope of the interference and phase information is inside the envelope.

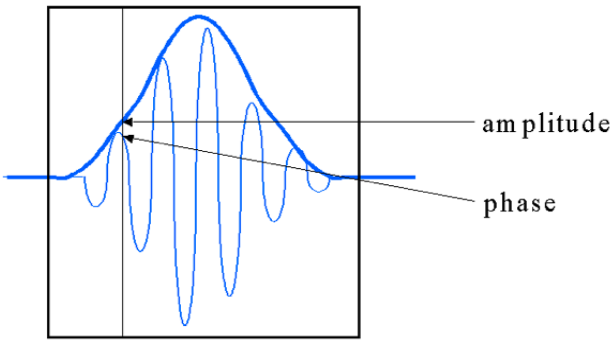


Figure 3.5. Image shows the Interference with Intensity and phase information. Intensity information is in envelope and phase information is inside the envelope.

The phase- difference values were calculated by subtracting a time –shifted version of the data from the original data and formula is as given below,

$$\Delta\emptyset = \emptyset(z_1, t) - \emptyset(z_2, t)$$

The phase noise is inversely proportion to the square root of SNR. To reduce the phase noise SNR should be high.

$$\Delta\emptyset_m = \sqrt{\frac{1}{SNR}}$$

The phase noise is determined by the standard deviation of the phase difference values [15].

$$\sigma_{\Delta\emptyset} = \sqrt{\frac{1}{2SNR(z_1,t)} + \frac{1}{2SNR(z_2,t)}}$$

Intensity is the function of wavenumber and square of electric power. It is obtained from the electric field power of the reference arm and sample arm power [4].

$$E_T = E_r\, e^{i(kz_r - wt)} + E_s\, e^{i(kz_s - wt)}$$

$$I_o(k) = I_r(k) + I_s(k) + 2\sqrt{I_r(k) I_s(k)}\cos 2\pi k(z_r - z_s)$$

AC term

3.3. Phase stability in SD-OCT

Intensity is very robust and has high sensitivity whereas phase is very sensitive and it is hard to get the high sensitivity. For the phase stability, the SD-OCT has very stable light source. It has the continuous output of broadband light source. The spatial separation of the frequency because of diffraction grating which can specially assign the wavelength to the pixels of the CCD array in the

line scan camera (1024 pixels) using post processing code. When light is normally incident on grating the diffracted light will have maxima at angles θ_m given by.

$$d\left(\sin\theta_i + \sin\theta_t\right) = m\lambda$$

where d is the slit spacing in diffraction grating , m is the order of diffraction, λ is the wavelength, θ_i is the incident angle of light and θ_t is the transmission angle of light.

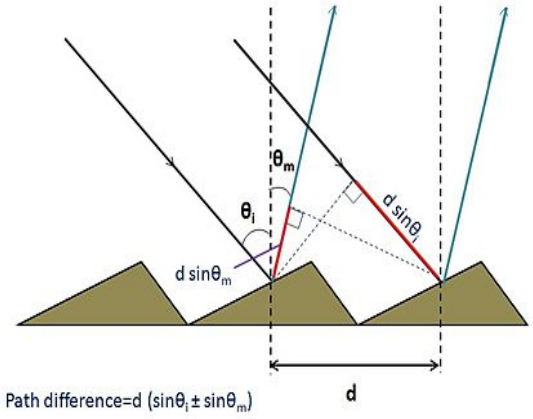


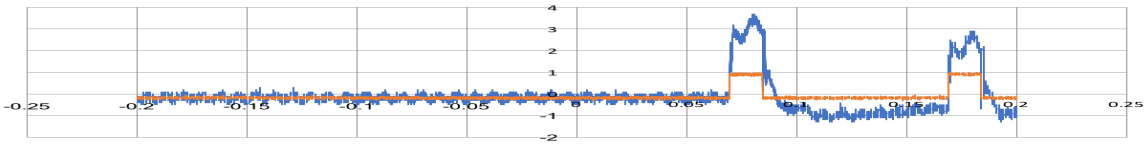
Figure 3.6. Diffraction grating

However, the relationship between the angles of the diffracted beams, the grating spacing and the wavelength of the light apply to any regular structure of the same spacing, because the phase relationship between light scattered from adjacent elements of the grating remains the same.

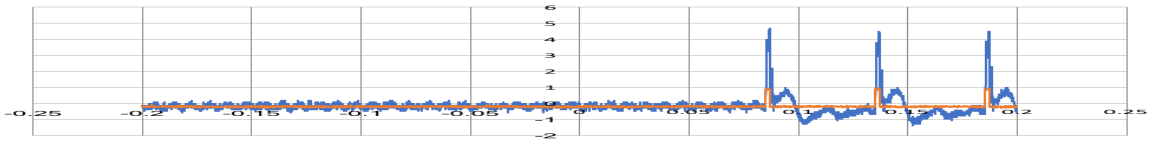
Chapter 4. Experimental Data

4.1 Electrical data

Before taking the any optical data of the neural activity I stimulated the nerve externally using nerve stimulator (A&M systems 1800) on the nerve chamber I made with the Plexi glass and 3D printed chamber. The other end of the nerve was on the electrodes from where the action potential was send to the DC amplifier (A& M Systems 1800) whose output was connected to the oscilloscope to compare this electrical data with OCT data. I took multiple set of data by changing the pulse duration and keeping pulse period maximum. The few others by changing the amplitude of the baseline voltage. Also took the data when nerve is dead to compare and see action potential change. Figure 4.1 From the graphs, it was clearly seen that action potential is seen when nerve is alive and there is no action potential when nerve is dead.



(a) Electrical data from oscilloscope when nerve is dead.



(b) electrical data from oscilloscope when nerve is alive.

Figure 4.1. (a) Electrical data from oscilloscope when nerve is dead where x-axis represent the time in seconds and Y-axis is the voltage in volts. (b) Electrical data from oscilloscope when nerve is dead where x-axis represent the time in seconds and Y-axis is the voltage in volts. The blue waveform is the amplified output from the nerve activity and orange waveform is the triggering pulse given to the nerve for external stimulation.

4.2. Sources of phase noise in SD-OCT system

The phase stability is analyzed by measuring the phase difference and the removal of fixed pattern noise. The phase difference values follow a Gaussian probability distribution in which the standard deviation $\sigma_{\Delta\phi}$ depends only on SNR [15]. From the formula given below it was is clear that phase noise is inversely proportional to SNR. It is that neural activity can be as small as 2nm. As shown in figure 4.2. it was demonstrated in our lab to check the effect of neural activity piezo wire was already used to see the thickness change (nm) with time. It was understood that phase noise has to be very small to see any major change in the thickness of piezo wire or neural activity

$$\sigma_{\Delta\phi} = \sqrt{\frac{1}{SNR}}$$

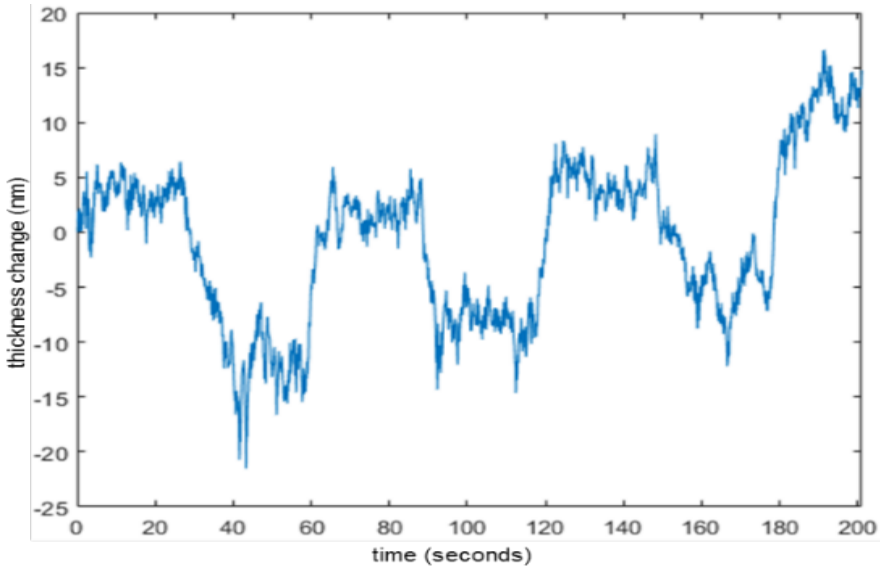
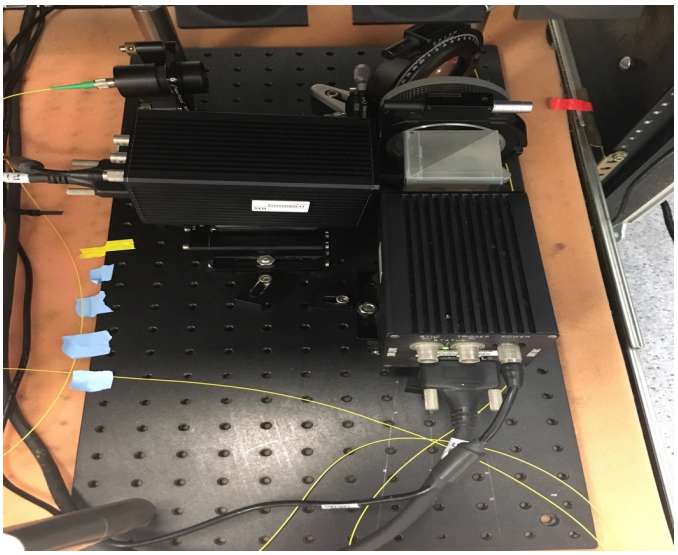
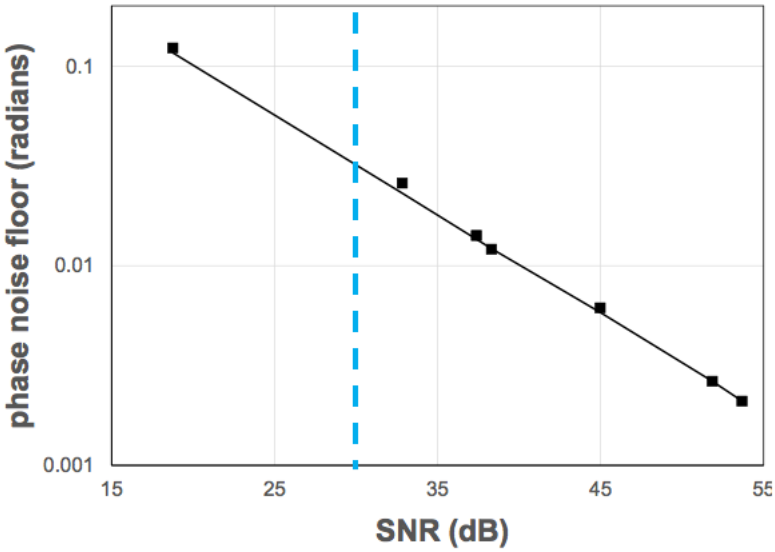


Figure4.2. Thickness change in piezo wire with time.

To increase the SNR suffiently to see the neural activity I realigned the spectrometer of the SD-OCT system as shown in the figure4.3.



(a) Setup of the Spectrometer in SD-OCT sysyem



(b) Graph shows the phase noise Vs SNR

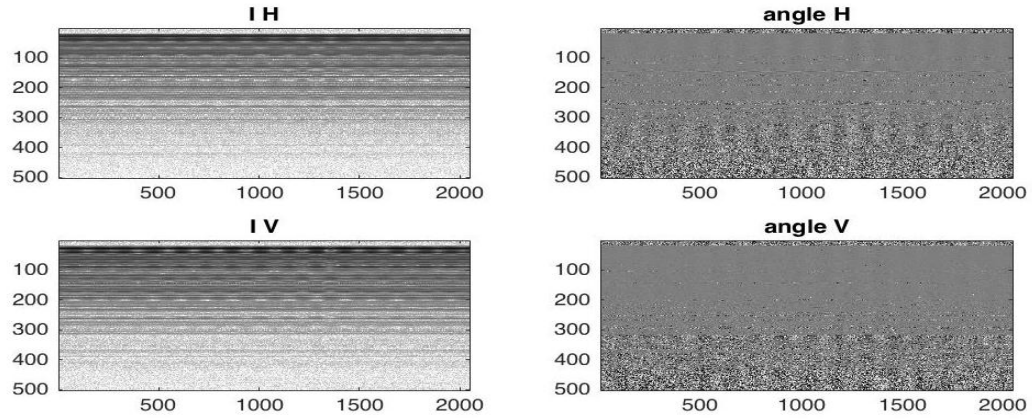
Figure4.3. (a) Images shows the realigned spectrometer (b) Graph of the phase noise Vs SNR and blue dotted line shows the SNR before the realignment 30dB.

After aligning the system, I was able to get SNR around 33dB which was sufficient to see small changes up to 2nm in the nerve the phase noise occurs may occur because of the lateral motion also. Here it can be assumed that both the A-lines are obtained from the same location in the nerve but which may not be true when lateral scanning is used to the cross-sectional images (B-scan) [16]. The difference in measured sample structure causes an additional phase noise term $\sigma_{\Delta x}$ that is based on random phases at a particular point in the sample and the displacement of the beam compared to its spot size [16].

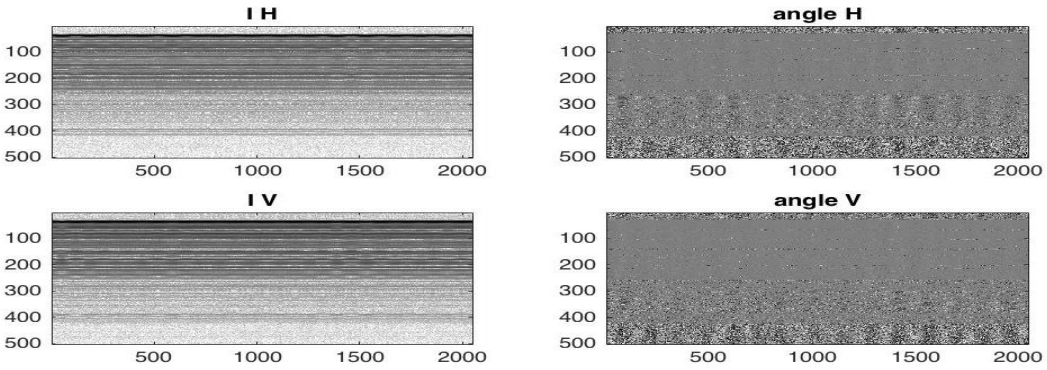
$$\sigma_{\Delta x} = \sqrt{\frac{4\pi}{3}\left(1 - e^{-2\left(\frac{\Delta x}{d}\right)^2}\right)}$$

Here d is the beam diameter or the spot size on the nerve and Δx is the displacement in the spot in-between the successive A-lines. In this SD-OCT setup first I suspected that as I am taking the cross –sectional images this noise patterns can be because of the Galvo jittering. I assumed that beam is hitting the same spot for every A-lines but it might be so there can be small jitter in Galvo and that may cause the beam to hit the different position on nerve. Figure 4.4shows the nerve data of walking leg nerve of the lobster where I am externally stimulating the nerve and trying to see any action potential change on the other end of the nerve. Due to some kind of phase noise in the background it was not possible to see the any phase change in the nerve. So, I took more data set with the by turning the power OFF

of the Galvo actually because I assumed that noise might be because of the jitter in the Galvo. For this data set I also kept the coverslip on the top of walking leg nerve as reference surface to see any change in depth due to action potential.



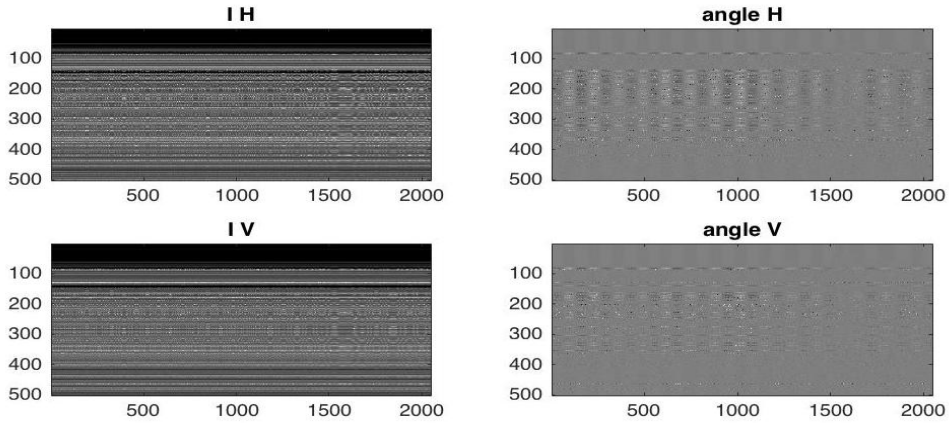
(a) Galvo On and No AP



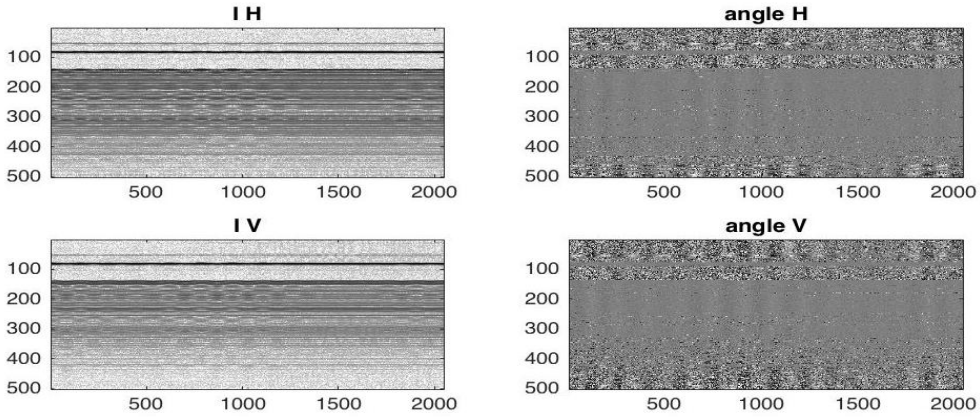
(b) Galvo ON and AP

Figure 4.4. (a) Data set of the walking leg nerve of lobster when Galvo is ON and No AP (b) Data set of the walking leg nerve of lobster when Galvo is ON and AP data set on the left IH and IV are the intensity were nerve bundles are clearly seen but cannot see the action potential change in image (b) therefore phase difference is used to see the small activity in nerve but in this angle H and angle V image also any change in nerve is not seen because there is still some kind of phase noise in the background.

Figure 4.5 shows the data with the Galvo OFF and coverslip on the top of the nerve and there is still the same phase noise in the background which doesn't allow to change any small change due to action potential in nerve. To eliminate the noise due to Galvo jitter I calculated the displacement or lateral motion using the above given formula. The beam



(a) Galvo Off, CS and No AP



(b) Galvo Off, CS and No AP

Figure 4.5. (a) Data set of the walking leg nerve of lobster when Galvo is ON and No AP with coverslip on top of the nerve (b) Data set of the walking leg nerve of lobster when Galvo is ON and No AP and coverslip on top of the nerve.

diameter for my setup was 30 μm and phase difference from the post processing code was 0.14 rads which gave the lateral motion displacement Δx od 1.5 μm. The conclusion was this phase noise is due to some other reason and not the Galvo jitter because 1.5 μm of displacement is too big for Galvo.

Conclusion

Even after increasing the SNR to 33dB it was not sufficient to see the such a small change in the walking leg nerve of lobster. There is some unknown noise of around 60Hz in the image that can be clearly seen even the Galvo is OFF. From image, it was calculated that 2000 A-lines takes 0.5sec because camera speed is 4000KHz which means 4000 A-lines per sec the noise image has approximately 30 oscillations half a second which is 60Hz of phase noise per sec. The reason for this noise might be the air current in the lab space still need to figure it out. This can be potential future goal for looking at the phase change in nerves. As the electrical stimulation is given to the nerve so frequently one more thing has to be checked is there any thermal change in the nerve because of this frequent stimulation.

References

1. J. Fujimoto, W. Drexler (2008). *Optical Coherence Tomography Technology and Applications*. Heidelberg: Springer.

2. R. K. Manapuram, V. G. R. Manne, K. V. Larin (2009). Phase-sensitive swept source optical coherence tomography for imaging and quantifying of microbubbles in clear and scattering media. *Journal of Applied Physics,* 105.

3. B. J. Vakoc, S. H. Yun, J. F. de Boer, G. J. Tearney, B. E. Bouma (2005). Phase-resolved optical frequency domain imaging. *Optics Express,* 5486

4. T. Akkin, C. Joo, and J. F. de Boer (2007). Depth-Resolved Measurement of Transient Structural Changes during Action Potential Propagation. *Biophysical Journal,* 1347–1353

5. B. Baraf (2015). *Angiography and Polarimetry of the posterior eye with functional Optical Coherence Tomography*. Netherlands: Ipskamp Drukkers

6. B. Braaf,1, K. A. Vermeer, V. D.P. Sicam, E.V Zeeburg, J. C. van Meurs, J. F. de Boer (2011). Phase-stabilized optical frequency domain imaging at 1-μm for the measurement of blood flow in the human choroid. *Optical Society of America.*

7. A. Unterhuber, B. Povazay, B. Hermann, H. Sattmann, A. Chavez-Pirson, and W. Drexler, (2005). In vivo retinal optical coherence tomography at 1040 nm - enhanced penetration into the choroid. *Optics Express* 13(9), 3252–3258.

8. E. C. Lee, J. F. de Boer, M. Mujat, H. Lim, and S. H. Yun, (2006). In vivo optical frequency domain imaging of human retina and choroid, *Optics Express* 14(10), 4403–4411.

9. Y. Chen, D. L. Burnes, M. de Bruin, M. Mujat, and J. F. de Boer, (2009). Three-dimensional pointwise comparison of human retinal optical property at 845 and 1060 nm using optical frequency domain imaging. *Journal of Biomedical Optics,* 14(2), 024016

10. F. A. Wininger, J. L. Schei, D. M. Rector (2009). Complete Optical Neurophysiology: Toward Optical Stimulation and Recording of Neural Tissue. *Applied Optics*

11. https://en.wikipedia.org/wiki/Photodiode

12. https://www.thorlabs.com/newgrouppage9.cfm?objectgroup_ID=4047

13. http://www.nvidia.com/object/IO_20020111_5424.html

14. B. Baraf, K.V. Vienoia (2012). Real-time eye motion correction in phase-resolved OCT angiography with tracking SLO. *Biomedical Optics Express.*

15. B. J. Vakoc, S. H. Yun, J. F. de Boer, G. J. Tearney, B. E. Bouma (2005). Phase-resolved optical frequency domain imaging. *Optics Express* 5483.

16. B. H. Park, M. C. Pierce, B. Cense, S.H. Yun, M. Mujat, G.J. Tearney, B.E. Bouma, J.F. de Boer (2005). Real-time fiber-based multi-functional spectral domain optical coherence tomography at 1.3μm. *Optics Express* 3931.

Appendix

The GPU code I wrote fast processing in Matlab has multiple functions and main code using GPU commands is given below:

Main code for SS_OCT

```
close all

%% Create Parameters in GPU

pdLineGPU          = gpuArray(pdLine);
dSlopeGPU          = gpuArray(dSlope);
bShowResultsGPU     = zeros(1,'gpuArray');
m_nRawLineLength    = 3000;
m_nRawLineLengthGPU = gpuArray(m_nRawLineLength);
m_pdMZIParameters   = [16384, 180, 2415, 64, 8192, 4, 50, 700, 64, 44, 32, 0.3, -1010, 1950, 64];
pdMZIParametersGPU  = gpuArray(m_pdMZIParameters);
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
m_strFile = 'C:\Users\hpark\Desktop\SS_code\Ref_02';
[pdRawOCTGPU, pdRawMZIGPU] = ReadRawOCTGPU(m_strFile, m_nRawLineLengthGPU);
[pdRefOCTGPU] = RegularizeMZIGPU(pdRawOCTGPU, pdRawMZIGPU, pdMZIParametersGPU, pdLineGPU, dSlopeGPU,1);
pdAvgRefGPU = mean(pdRefOCTGPU, 2);

m_strFile = 'C:\Users\hpark\Desktop\SS_code\Sample_02';
[pdRawOCTGPU, pdRawMZIGPU] = ReadRawOCTGPU(m_strFile, m_nRawLineLengthGPU);
[pdOCTGPU] = RegularizeMZIGPU(pdRawOCTGPU, pdRawMZIGPU, pdMZIParametersGPU, pdLineGPU, dSlopeGPU);
pdCleanOCTGPU = pdOCTGPU - repmat(pdAvgRefGPU, [1,size(pdOCTGPU,2)]);


scrsz = get(groot,'ScreenSize');
nY = 3;  nBottom = 50;  nTop = 90;  nX = 3;  nLeft = 10;  nRight = 10;
nHeight = scrsz(4)-nBottom;  nWidth = scrsz(3)-nLeft;
fProfile = figure('Position',[nLeft+0*nWidth/nX nBottom+2*nHeight/3 nWidth/nX-nRight nHeight/nY-nTop]);
% fAngle   = figure('Position',[nLeft+1*nWidth/nX nBottom+2*nHeight/3 nWidth/nX-nRight nHeight/nY-nTop]);


pdFFTGPU= fft(pdCleanOCTGPU);
figure(fProfile), plot(20*log10(abs(pdFFTGPU))); title('Final OCT profile');
hold on;
pdMeanProfileGPU = mean(abs(pdFFTGPU).^2, 2);
plot(10*log10(pdMeanProfileGPU), 'k', 'LineWidth', 2);
hold off;
xlim([1,round(0.5*size(pdFFTGPU,1))]);

% pdAngle = angle(pdFFT);
% pdAngle1 = pdAngle(nDepth1,:);
% pdAngle2 = pdAngle(nDepth2,:);
% pdDiff = pdAngle1 - pdAngle2;
% pdStep = floor(pdDiff / (1*pi));%% why r we dividing by pi?
% pdDiff = pdDiff - 1*pi * pdStep;
% figure(fAngle), plot(pdDiff);
% title(sprintf('the:%3.2fmrad; exp:%3.2fmrad', 1000*sqrt(dVar), 1000*std(pdDiff)));

% profile off
% p=profile('info');
```

Function to read raw data:

```
function [pdOCT, pdMZI] = ReadRawOCTGPU (strFile, nRawLineLength)

if (nargin < 2)
   nRawLineLength = 3000;

end % if (nargin < 2)

fp = fopen(strFile, 'r', 'l');
pdRawLine =fread(fp, 'uint16');
% pdRawLineGPU = gpuArray(pdRawLine);
pdOCTLine = pdRawLine(1:2:end);
pdMZILine = pdRawLine(2:2:end);

junk = fclose(fp);
clear fp pdRawLine pdRawLineGPU junk;

nNumberLines = floor(length(pdOCTLine) / nRawLineLength);
nFullLength = nNumberLines * nRawLineLength;
pdOCT = reshape(pdOCTLine(1:nFullLength), nRawLineLength, []);%% change the size of the matrix while keeping same number of
elements in it.
pdMZI = reshape(pdMZILine(1:nFullLength), nRawLineLength, []);

pdOCT=gpuArray(pdOCT);
pdMZI=gpuArray(pdMZI);
clear nNumberLines nFullLength pdOCTLine pdMZILine;
```

Function to clean the MZI data:

```
function [pdCleanedMZI] = CleanMZIGPU(pdRefRawMZI, nLeft, nRight, nRound, dLevel)

nLineLength  = size(pdRefRawMZI,1);
nNumberLines = size(pdRefRawMZI,2);

pdMask                    = zeros([nLineLength, 1],'gpuArray');
pdMask(nLeft:nRight)          = 1.0;
pdMask(nLeft:nLeft+nRound-1)   = 0.5*(1+cos((nRound-1:-1:0)'*(pi/(nRound-1))));
pdMask(nRight-nRound+1:nRight) = 0.5*(1+cos((0:nRound-1)' *(pi/(nRound-1))));

pdCleanedMZI = zeros(([nLineLength,nNumberLines]),'gpuArray');


for nLine = 1 : nNumberLines
   pdCleanedMZI(:,nLine) = pdMask .* (pdRefRawMZI(:,nLine) - dLevel);
end

clear nLineLength nNumberLines pdMask nLine;
```

Function for Zero padding of the data

```
function [pdZP] = ZeroPadGPU(pd, nPaddingFactor)


nLineLength  = size(pd, 1);
nNumberLines = size(pd, 2);
```

```
nMidLength = nLineLength / 2 + 1;
% nMidLength = gpuArray(nMidLength1);

pdFFT = fft(pd);
pdFFT(nMidLength, :) = pdFFT(nMidLength, :) ./ 2.0;

pdPaddedFFT = zeros([nPaddingFactor*nLineLength, nNumberLines]);
pdPaddedFFT(1:nMidLength, :) = pdFFT(1:nMidLength, :);
pdPaddedFFT(end-nMidLength+2:end, :) = pdFFT(end-nMidLength+2:end, :);

pdZP = ifft(pdPaddedFFT) * nPaddingFactor;


clear nLineLength nNumberLines nMidLength pdFFT pdPaddedFFT;


Function to calculate the  regularization of MZI

function [pdLine, dSlope] = CalculateMZIRegularization(pdRefRawMZI, pdMZIParameters)

nRawLineLength       = size(pdRefRawMZI, 1);
nNumberLines         = size(pdRefRawMZI, 2);

dSaturation          = pdMZIParameters( 1);
nMZILeft             = pdMZIParameters( 2);
nMZIRight            = pdMZIParameters( 3);
nMZIRound            = pdMZIParameters( 4);
dMZILevel            = pdMZIParameters( 5);
nPaddingFactor       = pdMZIParameters( 6);
nPeakSearchLeftIndex  = pdMZIParameters( 7);
nPeakSearchRightIndex = pdMZIParameters( 8);
dPeakSearchLeftLevel  = pdMZIParameters( 9);
dPeakSearchRightLevel = pdMZIParameters(10);
nPeakRoundLength      = pdMZIParameters(11);
dAngleDiffThreshold   = pdMZIParameters(12);



pdCleanedMZI = CleanMZIGPU(pdRefRawMZI, nMZILeft, nMZIRight, nMZIRound, dMZILevel);
pdZPMZI = ZeroPadGPU(pdCleanedMZI, nPaddingFactor);
pdLineIndex = (0:(nRawLineLength-1))' - 0.5*(nMZILeft+nMZIRight);
pdZPLineIndex = repmat((0:(nPaddingFactor*nRawLineLength-1))' / nPaddingFactor - 0.5.*(nMZILeft+nMZIRight),
[1,nNumberLines]);%repmat(A,2(row),3(column)) & repmat(A,2) means 2 by 2 matrix
clear pdCleanedMZI;
pdResampledMZI = zeros([nRawLineLength, nNumberLines]);

for nLine = 1 : nNumberLines
   pdResampledMZI(:,nLine) = interp1(pdZPLineIndex(:,nLine), pdZPMZI(:,nLine), pdLineIndex);
end % for nLine = 1 : nNumberLines
clear nLine;

pdFFT                = fft(pdResampledMZI);
pdPeakSearch         = mean(abs(pdFFT), 2);
[dPeakMax, nPeakIndex] = max(pdPeakSearch(nPeakSearchLeftIndex:nPeakSearchRightIndex));
nPeakIndex           = nPeakIndex + nPeakSearchLeftIndex - 1; %% position for peak
pnSearchLeft         = find(pdPeakSearch(1:nPeakIndex) < 10.0^(0.1*dPeakSearchLeftLevel));
nLeftIndex           = pnSearchLeft(end);
pnSearchRight        = find(pdPeakSearch(nPeakIndex:nPeakSearchRightIndex) < 10.0^(0.1*dPeakSearchRightLevel));
nRightIndex          = pnSearchRight(1) - 1 + nPeakIndex;
pdCutPeak            = CleanMZI(pdFFT, nLeftIndex-nPeakRoundLength, nRightIndex+nPeakRoundLength, nPeakRoundLength, 0);
clear pdFFT pdPeakSearch dPeakMax nPeakIndex pnSearchLeft nLeftIndex pnSearchRight nRightIndex;

% Examine angle
pdReverse = ifft(pdCutPeak);
pdAngle = unwrap(angle(pdReverse));%% eliminates the discontinuites
pdStep = floor(pdAngle(nMZILeft+nMZIRound,:) / (2.0*pi));% round the values towards negative infinity.
```

```
for nLine = 1 : nNumberLines;
   pdAngle(:, nLine) = pdAngle(:, nLine)-(2.0*pi) * pdStep(:, nLine);
end

pdMeanAngle = mean(pdAngle, 2);
pdDiff = pdMeanAngle(2:end) - pdMeanAngle(1:end-1);
pnFitLimit = find(pdDiff > dAngleDiffThreshold);
pAngleLine = polyfit(pdLineIndex(pnFitLimit(1):pnFitLimit(end)), pdMeanAngle(pnFitLimit(1):pnFitLimit(end)), 1);
% dSlope = pAngleLine(1);
% pdLine = polyval(pAngleLine, pdLineIndex);% evaluates pAngleline at each pdLineIndex.
dSlope=gpuArray(gdSlope);
pdLine=gpuArray(gpdLine);


clear nRawLineLength nNumberLines;
clear dSaturation nMZILeft nMZIRight nMZIRound dMZILevel nPaddingFactor nPeakSearchLeftIndex;
clear nPeakSearchRightIndex dPeakSearchLeftLevel dPeakSearchRightLevel nPeakRoundLength dAngleDiffThreshold;
clear nFinalPullStart nFinalLineLength nFinalRoundLength;
```

Regularize MZI code

```
function [pdOCT] = RegularizeMZIGPU(pdRawOCT, pdRawMZI, pdMZIParameters, gpdLine, gdSlope, bShowResults)
nRawLineLength     = zeros((size(pdRawMZI, 1)));
nNumberLines       = zeros((size(pdRawMZI, 2)));
dSaturation          = pdMZIParameters( 1);
nMZILeft             = pdMZIParameters( 2);
nMZIRight            = pdMZIParameters( 3);
nMZIRound            = pdMZIParameters( 4);
dMZILevel            = pdMZIParameters( 5);
nPaddingFactor       = pdMZIParameters( 6);
nPeakSearchLeftIndex  = pdMZIParameters( 7);
nPeakSearchRightIndex = pdMZIParameters( 8);
dPeakSearchLeftLevel  = pdMZIParameters( 9);
dPeakSearchRightLevel = pdMZIParameters(10);
nPeakRoundLength      = pdMZIParameters(11);
dAngleDiffThreshold   = pdMZIParameters(12);
nFinalPullStart       = pdMZIParameters(13);
nFinalLineLength      = pdMZIParameters(14);
nFinalRoundLength     = pdMZIParameters(15);


if (nargin < 6)
   bShowResults = 0;
end % if (nargin < 6)
if (bShowResults == 1)
   scrsz = get(groot,'ScreenSize');
   nY = 3;  nBottom = 50;  nTop = 90;  nX = 5;  nLeft = 10;  nRight = 10;
   nHeight = scrsz(4)-nBottom;  nWidth = scrsz(3)-nLeft;
   fRawMZI       = figure('Position',[nLeft+0*nWidth/nX nBottom+2*nHeight/3 nWidth/nX-nRight nHeight/nY-nTop]);
%    fCleanedMZI   = figure('Position',[nLeft+1*nWidth/nX nBottom+2*nHeight/3 nWidth/nX-nRight nHeight/nY-nTop]);
%    fProfile      = figure('Position',[nLeft+2*nWidth/nX nBottom+2*nHeight/3 nWidth/nX-nRight nHeight/nY-nTop]);
   fAngle        = figure('Position',[nLeft+3*nWidth/nX nBottom+2*nHeight/3 nWidth/nX-nRight nHeight/nY-nTop]);
   fError        = figure('Position',[nLeft+4*nWidth/nX nBottom+2*nHeight/3 nWidth/nX-nRight nHeight/nY-nTop]);
%    fFixedMZI     = figure('Position',[nLeft+0*nWidth/nX nBottom+1*nHeight/3 nWidth/nX-nRight nHeight/nY-nTop]);
   fFinalMZI     = figure('Position',[nLeft+1*nWidth/nX nBottom+1*nHeight/3 nWidth/nX-nRight nHeight/nY-nTop]);
   fFinalProfile = figure('Position',[nLeft+2*nWidth/nX nBottom+1*nHeight/3 nWidth/nX-nRight nHeight/nY-nTop]);
   fRawOCT       = figure('Position',[nLeft+0*nWidth/nX nBottom+0*nHeight/3 nWidth/nX-nRight nHeight/nY-nTop]);
%    fCleanedOCT   = figure('Position',[nLeft+1*nWidth/nX nBottom+0*nHeight/3 nWidth/nX-nRight nHeight/nY-nTop]);
%    clear scrsz nY nBottom nTop nX nLeft nRight nHeight nWidth;
end % if (bShowResults == 1)

if (bShowResults == 1)
   figure(fRawMZI);
   plot(pdRawMZI);
   xlim([1, nRawLineLength]);
```

```
    ylim([0,dSaturation ]);
    title('raw MZI');
end % if (bShowResults == 1)

pdCleanedMZI      = CleanMZIGPU(pdRawMZI, nMZILeft, nMZIRight, nMZIRound, dMZILevel);
pdZPMZI           = ZeroPadGPU(pdCleanedMZI, nPaddingFactor);
pdLineIndex       = (0:(nRawLineLength-1))' - 0.5*(nMZILeft+nMZIRight);
pdZPLineIndex     = repmat((0:(nPaddingFactor*nRawLineLength-1))' / nPaddingFactor - 0.5*(nMZILeft+nMZIRight),
[1,nNumberLines]);
clear pdCleanedMZI;

for nRepetition= 1 : 18

     pdCutPeak=MZIPeakSearch(nRawLineLength,nNumberLines,pdZPLineIndex, pdZPMZI, pdLineIndex);
    [pdAngle]=PhaseCorrection(pdCutPeak,nRepetition,nNumberLines,nMZILeft,nMZIRight,nMZIRound);
    [pdError]=Errorcalculation(pdAngle,nRawLineLength,nPaddingFactor, nNumberLines,pdZPLineIndex,nMZILeft,nMZIRight);

     pdZPLineIndex = pdZPLineIndex + pdError / gdSlope;

    if (bShowResults == 1)
        figure(fAngle);
        plot(pdAngle);
        hold on;
        plot(gpdLine, 'k', 'LineWidth', 2);
        hold off;
        title('angle');

        figure(fError);
        plot(pdError);
        title('error');
        ylim([-5e-4, 5e-4]);
    end % if (bShowResults == 1)
    clear pdCutPeak pdAngle nLine pdMeanAngle pdDiff  pdTemp;
end % nRepetitionGPU = 1 : 20

pnFinalIndex = nFinalPullStart : nFinalPullStart + nFinalLineLength - 1;
pdMZIPull = zeros([nFinalLineLength, nNumberLines],'gpuArray');

for nLine = 1 : nNumberLines
    pdMZIPull(:,nLine) = interp1(pdZPLineIndex, pdZPMZI, pnFinalIndex);
end % for nLine = 1 : nNumberLines

pdFinalMZI = CleanMZIGPU(pdMZIPull, 1, nFinalLineLength, nFinalRoundLength, 0);

if (bShowResults == 1)
    figure(fFinalMZI);
    plot(pdFinalMZI);
    title('final MZI pull');

    pdFFT = fft(pdMZIPull);
    figure(fFinalProfile);
    plot(20*log10(abs(pdFFT)));
    hold on,
    plot(20*log10(mean(abs(pdFFT), 2)), 'k', 'LineWidth', 2);
    xlim([1, floor(0.5*1950)]);
    title('final MZI profile');
end % if (bShowResults == 1)


if (bShowResults == 1)
    figure(fRawOCT);
    plot(pdRawOCT);
    xlim([1, nRawLineLength]);
    ylim([0,dSaturation ]);
    title('raw OCT');
end % if (bShowResults == 1)
```

```
pdCleanedOCT = CleanMZIGPU(pdRawOCT, nMZILeft, nMZIRight, nMZIRound, dMZILevel);
pdZPOCT = ZeroPadGPU(pdCleanedOCT, nPaddingFactor);
pdResampledOCT = zeros([nFinalLineLength, nNumberLines]);


for nLine = 1 : nNumberLines
   pdResampledOCT(:,nLine) = interp1(pdZPLineIndex, pdZPOCT, pnFinalIndex);
end % for nLine = 1 : nNumberLines
pdOCT = CleanMZIGPU(pdResampledOCT, 1, nFinalLineLength, nFinalRoundLength, 0);

clear nRawLineLength nNumberLines;
clear dSaturation nMZILeft nMZIRight nMZIRound dMZILevel nPaddingFactor nPeakSearchLeftIndex;
clear nPeakSearchRightIndex dPeakSearchLeftLevel dPeakSearchRightLevel nPeakRoundLength dAngleDiffThreshold;
clear nFinalPullStart nFinalLineLength nFinalRoundLength;
if (bShowResults == 1)
   clear fRawMZI fCleanedMZI fProfile fAngle fError fFixedMZI fFinalMZI;
end % if (bShowResults == 1)
```

Code to compare the GPU and CPU time

```
close all;  clearvars;  clc;
% nLines = 256;
nLength= 2048;
% pdSpectrum = rand([nLength, nLines]);
% pdInterpolated = zeros([nLength, nLines]);
% pdWavelength = repmat(200*((1:nLength)'-1), [1,nLines]);
pdK = 190*((1:nLength)'-1)+1;
nIterations = 4;
% nR =8;

%% cpu version
disp('cpu version');
for nLines=[128,256,512,1024]
    pdSpectrum = rand([nLength, nLines]);
    pdInterpolated = zeros([nLength, nLines]);
    pdWavelength = repmat(200*((1:nLength)'-1), [1,nLines]);
    for nR=2:2:10
        dTotalTime = 0;
        for nIteration = 1 : nIterations;
            tic;
           for nAline = 1 : nLines
               for nInternal = 1 : nR;
                pdInterpolated(:,nAline) = interp1(pdWavelength(:,nAline),
pdSpectrum(:,nAline), pdK);
                   pdInterpolated(:,nAline) = fft(pdWavelength(:,nAline));
               end % for nInternal

           end % for nAline
           dLocalTime = toc;
           dTotalTime = dTotalTime + dLocalTime;
        end %% for nIteration
        dCPUTime(nR,nLines) = dTotalTime / nIterations;
        dTime4CPU=(dCPUTime)';
%         pdTable4CPU=table(dTime4CPU);
        pdTable4CPU=array2table(dTime4CPU(128:128:size(dTime4CPU,1),2:2:size(dTime4CPU,2)));
    end %% for nR
end %% for nLines
% % dCPUTime = dTotalTime / nIterations;
% keyboard;

%% gpu version
disp('gpu version');
gdK = gpuArray(pdK);
for nLines=[128,256,512,1024]
    pdSpectrum = rand([nLength, nLines]);
    pdInterpolated = zeros([nLength, nLines]);
    pdWavelength = repmat(200*((1:nLength)'-1), [1,nLines]);
    gdInterpolated = gpuArray(pdInterpolated);
    for nR=2:2:10
```

```
        dTotalTime = 0;
        for nIteration = 1 : nIterations;
            tic;
            gdWavelength = gpuArray(pdWavelength);
            gdSpectrum = gpuArray(pdSpectrum);
            for nAline = 1 : nLines;

                for nInternal = 1 : nR;
                  gdInterpolated(:,nAline) = interp1(gdWavelength(:,nAline),
gdSpectrum(:,nAline),gdK);
                    gdInterpolated(:,nAline) = fft(gdWavelength(:,nAline));
                end % for nInternal

           end % for nAline
           pdInterpolated = gather(gdInterpolated);
           dLocalTime = toc;
           dTotalTime = dTotalTime + dLocalTime;
       end % for nIteration
       dGPUTime(nR,nLines) = dTotalTime / nIterations;
       dTime4GPU=(dGPUTime)';
```

Code for testing GPU time

```
close all; clearvars;

nNumberLines = 4096;
nReps = 100;

pnLengths = [2, 4, 8, 26, 32, 64, 128, 256]; % , 512, 1024, 2048, 4096];

pdGTimes = 0*pnLengths;
for nI = 1 : length(pnLengths);
    nLineLength = pnLengths(nI);
    tic;
    for nRep = 1 : nReps;
        gArray = gpuArray(rand(nLineLength, nNumberLines, 'double'));
        gResult = fft(gArray);
    end
    x = toc;
    pdGTimes(nI) = x / nReps;
end
```

Using Parallel for loop

```
close all;  clearvars;  clc;
tic
m_nRawLineLength  = 3000;
m_pdMZIParameters = [16384, 180, 2415, 64, 8192, 4, 50, 700, 64, 44, 32, 0.3, -1010, 1950, 64];

m_strFile = '/Users/kavanshah/Documents/MATLAB/Ref_02';
[pdRefRawOCT, pdRefRawMZI] = ReadRawOCT(m_strFile, m_nRawLineLength);
m_strfile = '/Users/kavanshah/Documents/MATLAB/Sample_02';
[pdRawOCT, pdRawMZI]       = ReadRawOCT(m_strfile, m_nRawLineLength);

[pdLine, dSlope] = CalculateMZIRegularization(pdRefRawMZI, m_pdMZIParameters);
nRawLineLength        = size(pdRefRawMZI, 1);
nNumberLines          = size(pdRefRawMZI, 2);

dSaturation           = m_pdMZIParameters( 1);
nMZILeft              = m_pdMZIParameters( 2);
nMZIRight             = m_pdMZIParameters( 3);
nMZIRound             = m_pdMZIParameters( 4);
dMZILevel             = m_pdMZIParameters( 5);
nPaddingFactor        = m_pdMZIParameters( 6);
nPeakSearchLeftIndex  = m_pdMZIParameters( 7);
nPeakSearchRightIndex = m_pdMZIParameters( 8);
dPeakSearchLeftLevel  = m_pdMZIParameters( 9);
dPeakSearchRightLevel = m_pdMZIParameters(10);
nPeakRoundLength      = m_pdMZIParameters(11);
dAngleDiffThreshold   = m_pdMZIParameters(12);
nFinalPullStart       = m_pdMZIParameters(13);
nFinalLineLength      = m_pdMZIParameters(14);
nFinalRoundLength     = m_pdMZIParameters(15);


    scrsz = get(groot,'ScreenSize');
    nY = 3;  nBottom = 50;  nTop = 90;  nX = 5;  nLeft = 10;  nRight = 10;
    nHeight = scrsz(4)-nBottom;  nWidth = scrsz(3)-nLeft;
    fRawMZI       = figure('Position',[nLeft+0*nWidth/nX nBottom+2*nHeight/3 nWidth/nX-nRight
nHeight/nY-nTop]);
    fCleanedMZI   = figure('Position',[nLeft+1*nWidth/nX nBottom+2*nHeight/3 nWidth/nX-nRight
nHeight/nY-nTop]);
    fProfile      = figure('Position',[nLeft+2*nWidth/nX nBottom+2*nHeight/3 nWidth/nX-nRight
nHeight/nY-nTop]);
%     fAngle        = figure('Position',[nLeft+3*nWidth/nX nBottom+2*nHeight/3 nWidth/nX-nRight
nHeight/nY-nTop]);
%     fError        = figure('Position',[nLeft+4*nWidth/nX nBottom+2*nHeight/3 nWidth/nX-nRight
nHeight/nY-nTop]);
%     fFixedMZI     = figure('Position',[nLeft+0*nWidth/nX nBottom+1*nHeight/3 nWidth/nX-nRight
nHeight/nY-nTop]);
%     fFinalMZI     = figure('Position',[nLeft+1*nWidth/nX nBottom+1*nHeight/3 nWidth/nX-nRight
nHeight/nY-nTop]);
```

```
%     fFinalProfile = figure('Position',[nLeft+2*nWidth/nX nBottom+1*nHeight/3 nWidth/nX-nRight
nHeight/nY-nTop]);
%     fRawOCT       = figure('Position',[nLeft+0*nWidth/nX nBottom+0*nHeight/3 nWidth/nX-nRight
nHeight/nY-nTop]);
%     fCleanedOCT   = figure('Position',[nLeft+1*nWidth/nX nBottom+0*nHeight/3 nWidth/nX-nRight
nHeight/nY-nTop]);

%          pdnR={'2','4','6','8','10'};
       pdTable4CPU=array2table(dTime4GPU(128:128:size(dTime4GPU,1),2:2:size(dTime4GPU,2)));

    end %% for nR
end % for nLines
% dGPUTime = dTotalTime / nIterations;

%     clear scrsz nY nBottom nTop nX nLeft nRight nHeight nWidth;

    figure(fRawMZI);
    plot(pdRawMZI);
    xlim([1, nRawLineLength]);
    ylim([0, dSaturation]);
    title('raw MZI');

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~%
%USING PARFOR IN INNER LOOP

pdCleanedMZI = CleanMZI(pdRefRawMZI, nMZILeft, nMZIRight, nMZIRound, dMZILevel);
pdZPMZI      = ZeroPad(pdCleanedMZI, nPaddingFactor);
pdLineIndex  = (0:(nRawLineLength-1))' - 0.5*(nMZILeft+nMZIRight);
pdZPLineIndex= repmat((0:(nPaddingFactor*nRawLineLength-1))' / nPaddingFactor -
0.5*(nMZILeft+nMZIRight), [1,nNumberLines]);

for n=1:20
 pdResampledMZI=cell(1,nNumberLines);

 parfor nLine = 1 : nNumberLines
   pdResampledMZI {:,nLine}= zeros([nRawLineLength, nNumberLines]);
   pdResampledMZI {:,nLine}=interp1(pdZPLineIndex(:,nLine), pdZPMZI(:,nLine), pdLineIndex);
 end
  pdisNaNdata=pdResampledMZI(find(isnan(pdResampledMZI)));
  pdisNaNdata=0.0;
pdisNaN=cell2mat(pdResampledMZI);
  pdisNaN=pdResampledMZI{nNumberLines};
  pdisNaN=cell2mat(cellfun(@(x) cell2mat(x),pdResampledMZI,'Un',0));
        figure(fCleanedMZI);
        plot(pdLineIndex,pdisNaN);
        xlim([min(pdLineIndex), max(pdLineIndex)]);
        ylim([-dMZILevel, dSaturation-dMZILevel]);
        title('cleaned MZI');


    pdFFT = fft(pdisNaN);
    pdPeakSearch = mean(abs(pdFFT), 2);
    [dPeakMax, nPeakIndex] = max(pdPeakSearch(nPeakSearchLeftIndex:nPeakSearchRightIndex));
    nPeakIndex = nPeakIndex + nPeakSearchLeftIndex - 1;

    pnSearchLeft = find(pdPeakSearch(1:nPeakIndex) < 10.0^(0.1*dPeakSearchLeftLevel));
    nLeftIndex = pnSearchLeft(end);
    pnSearchRight = find(pdPeakSearch(nPeakIndex:nPeakSearchRightIndex) <
10.0^(0.1*dPeakSearchRightLevel));
    nRightIndex = pnSearchRight(1) - 1 + nPeakIndex;

    pdCutPeak = CleanMZI(pdFFT, nLeftIndex-nPeakRoundLength, nRightIndex+nPeakRoundLength,
nPeakRoundLength, 0);

    figure(fProfile);
    plot(20.0*log10(abs(pdCutPeak)));
    hold on;
    plot(20.0*log10(pdPeakSearch), 'k', 'LineWidth', 2);
    xlim([1, floor(0.5*nRawLineLength)]);
    hold off;
    title(sprintf('profile %2.1f dB', 20.0*log10(dPeakMax)))

      pdReverse = ifft(pdCutPeak);
```

```
      pdAngle = unwrap(angle(pdReverse));
   if (n == 1)
        pdStep = floor(pdAngle(nMZILeft+nMZIRound,:) / (2.0*pi));
   else
        nMid = round(0.5*(nMZILeft+nMZIRight));
        pdStep = round((pdAngle(nMid,:) - pdLine(nMid)) / (2.0*pi));
   end
   for nLine = 1 : nNumberLines;
       pdAngle(:, nLine) = pdAngle(:, nLine)-(2.0*pi) * pdStep(:, nLine);
   end

end %n=1:20

toc

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~%
USING PARFOR IN OUTER LOOP

pdCleanedMZI = CleanMZI(pdRawMZI, nMZILeft, nMZIRight, nMZIRound, dMZILevel);
pdZPMZI      = ZeroPad(pdCleanedMZI, nPaddingFactor);
pdLineIndex  = (0:(nRawLineLength-1))' - 0.5*(nMZILeft+nMZIRight);
parfor nRepetition = 1 : 1;
   nRepetition
   pdLine1=pdLine;
   pdZPLineIndex= repmat((0:(nPaddingFactor*nRawLineLength-1))' / nPaddingFactor -
0.5*(nMZILeft+nMZIRight), [1,nNumberLines]);
   pdResampledMZI = zeros([nRawLineLength, nNumberLines]);

   disp('done')
   for nLine = 1 : nNumberLines
       pdResampledMZI(:,nLine) =interp1(pdZPLineIndex(:,nLine), pdZPMZI(:,nLine),
pdLineIndex);
   end
%  if (bShowResults == 1)
        figure(fCleanedMZI);
        plot(pdLineIndex,pdResampledMZI);
        xlim([min(pdLineIndex), max(pdLineIndex)]);
        ylim([-dMZILevel, dSaturation-dMZILevel]);
        title('cleaned MZI');
 end % if (bShowResults == 1)

    pdFFT = fft(pdResampledMZI);
    pdPeakSearch = mean(abs(pdFFT), 2);
    [dPeakMax, nPeakIndex] = max(pdPeakSearch(nPeakSearchLeftIndex:nPeakSearchRightIndex));
    nPeakIndex = nPeakIndex + nPeakSearchLeftIndex - 1;

    pnSearchLeft = find(pdPeakSearch(1:nPeakIndex) < 10.0^(0.1*dPeakSearchLeftLevel));
    nLeftIndex = pnSearchLeft(end);
    pnSearchRight = find(pdPeakSearch(nPeakIndex:nPeakSearchRightIndex) <
10.0^(0.1*dPeakSearchRightLevel));
    nRightIndex = pnSearchRight(1) - 1 + nPeakIndex;

    pdCutPeak = CleanMZI(pdFFT, nLeftIndex-nPeakRoundLength, nRightIndex+nPeakRoundLength,
nPeakRoundLength, 0);

%   if (bShowResults == 1)
    figure(fProfile);
    plot(20.0*log10(abs(pdCutPeak)));
    hold on;
    plot(20.0*log10(pdPeakSearch), 'k', 'LineWidth', 2);
    xlim([1, floor(0.5*nRawLineLength)]);
    hold off;
    title(sprintf('profile %2.1f dB', 20.0*log10(dPeakMax)));
%   end % if (bShowResults == 1)

   pdReverse = ifft(pdCutPeak);
   pdAngle = unwrap(angle(pdReverse));
   if (nRepetition == 1)
        pdStep = floor(pdAngle(nMZILeft+nMZIRound,:) / (2.0*pi));
   else
        nMid = round(0.5*(nMZILeft+nMZIRight));
        pdStep = round((pdAngle(nMid,:) - pdLine1(nMid)) / (2.0*pi));
   end
```

```
%    pdAngle=cell(1,nNumberLines);
   for nLine = 1 : nNumberLines;
        pdAngle(:, nLine) = pdAngle(:, nLine) - (2.0*pi) * pdStep(:, nLine);
   end

     pdError = zeros([nRawLineLength*nPaddingFactor, nNumberLines]);
%    pdError = cell(1, nNumberLines);
  for nLine = 1 : nNumberLines;
        pdTemp = pdAngle(:,nLine) - pdLine1;
        pdError(:,nLine)= interp1(pdLineIndex, pdTemp, pdZPLineIndex(:,nLine));
  end
%     pdinterpError=pdError{nNumberLines};
    pdError(1:nMZILeft*nPaddingFactor, :) = repmat(pdError(nMZILeft*nPaddingFactor,:),
[nMZILeft*nPaddingFactor,1]);
    pdError(nMZIRight*nPaddingFactor:nRawLineLength*nPaddingFactor, :) =
repmat(pdError(nMZIRight*nPaddingFactor,:), [(nRawLineLength-nMZIRight)*nPaddingFactor+1,1]);

% if (bShowResults == 1)
    figure(fAngle);
        plot(pdAngle);
        hold on;
        plot(pdLine1, 'k', 'LineWidth', 2);
        hold off;
        title('angle');
%
        figure(fError);
        plot(pdError);
        title('error');
        ylim([-5e-4, 5e-4]);
% end % if (bShowResults == 1)
%
    pdZPLineIndex = pdZPLineIndex+pdError / dSlope;
%

end % nRepetition = 1 : 20
```