



SCSE19-0399

Visualization of Properties of Phase-Shifting Algorithms

Submitted in Partial Fulfillment of the Requirements for the
Degree of Engineering (Computer Science) of the Nanyang
Technological University

by

Edward Christopher (U1720912K)

Supervisor Dr. Qian Kemao

School of Computer Science and Engineering

2020

Abstract

The advancement of computer science, especially in conjunction with optical sciences, has led to a high demand for high precision optical testing. High precision optical testing is normally performed through a method known as interferometry, where surfaces of optical devices are mapped through the interference of 2 or more light waves. Phase Shifting Interferometry (PSI) is a one of the most widely used interferometry techniques for optical testing. PSI uses multiple interferograms introduced by means of a phase shift as opposed to a static interferogram in determining the phase value related to the surface of the tested optical devices. In order to find phase value, PSI uses PSI algorithms which utilizes the intensity values of from the interferograms, in order to reconstruct the tangent of the tested phase, which can be used to obtain phase value. Various PSI algorithms have been developed throughout the years as means to develop an algorithm resistant to various error sources. This project aims to develop a visualization application to illustrate PSI algorithm properties when exposed to different types of error. A web-based application consisting of a Backend REST API in conjunction with a Frontend Web Application is developed to support this aim. This web application is specially designed in order to be able to demonstrate an interactive and intuitive PSI 3D model for PSI phase errors. In the near future, this web application can be improved through incorporating better design ideas as well as providing integration with other platforms.

Keywords: Interferometry, Phase Shifting Interferometry, 3D model, Web Application, Phase Shifting Interferometry Algorithms

Acknowledgements

I would like express my special thanks and gratitude to Associate Professor Qian Kemao as well as Mr Yuchi Chen who had gave me the opportunity to work on this Final Year Project titled “Visualization of Properties of Phase-Shifting Algorithms”, as well as providing help and assistance in the learning process and development of this project.

I would also like to express my thanks to all my family, colleagues, and staffs for all the support provided until the completion of this project.

Table of Contents

Abstract.....	i
Acknowledgements.....	ii
Table of Contents.....	iii
List of Tables	v
List of Figures	vi
1. Introduction	1
1.1 Introduction	1
1.2 Scope and Limitations.....	2
2. Literature Review	4
2.1 Interferometry	4
2.2 Phase Shifting Interferometry	5
2.3 PSI Algorithms.....	7
2.3.1 Three-step Algorithm	7
2.3.2 Least-squares Algorithm (N-bucket Discrete Fourier Transform)	8
2.3.3 Family of Averaging Algorithms	9
2.3.4 Other PSI Algorithms	10
2.4 Error Analysis.....	11
2.5. Selected Error Representations.....	12
2.6 Development Platforms.....	14
2.6.1 REST API.....	14
3. Project Development.....	15
3.1 Overview	15
3.2 Early Development Phase	15
3.2.1 Overview	15
3.2.2 Development Phase Outline	16
3.2.3 Phase Issues	17
3.2.4 Conclusion.....	18
3.3 Backend Development	19
3.3.1 Overview	19

3.3.2 Phase Development Outline	19
3.3.3 Phase Issues	20
3.3.4 Conclusion	20
3.4 Frontend Development	21
3.4.1 Overview	21
3.4.2 Phase Development Outline	21
3.4.3 Phase Issues	22
3.4.4 Conclusion	23
3.5 Integration and Debugging	23
3.5.1 Overview	23
3.5.2 Phase Development Outline	24
3.5.3 Phase Issues	24
3.5.4 Conclusion	25
3.6 Refinement	25
3.6.1 Overview	25
3.6.2 Phase Development Outline	25
3.6.3 Phase Issues	26
3.6.4 Conclusion	26
4. Design and Implementation	27
4.1 Overview	27
4.2 Backend	28
4.3 Frontend	31
5. Conclusion	36
References	37

List of Tables

Table 1. Examples of N-bucket DFT PSI Algorithms.....	9
Table 2. Examples of Averaging Algorithms with 4-frame N-bucket DFT as base.	10
Table 3. 5 Project Development Iterations/Phases	15
Table 4. Early Development Phase Outline.....	17
Table 5. Backend Development Phase Outline	20
Table 6. Frontend Development Phase Outline	22
Table 7. Integration and Debugging Phase Outline	24
Table 8. Refinement Phase Outline	26
Table 9. Algorithm Data Structure	29

List of Figures

Figure 1. Basic Michelson Interferometer	4
Figure 2. Ways to perform phase shifting.....	5
Figure 3. Modern Interferometer	6
Figure 4. Backend REST API Software Architecture	28
Figure 5. Frontend Software Architecture	31
Figure 6. FormInput Component	32
Figure 7. Graph Component	33
Figure 8. Table Component	34
Figure 9. Details Component	35

1. Introduction

1.1 Introduction

The introduction of computers, its subsequent developments, and its applications for all other fields has led humans to the current state of the modern world. One of the advancements of computer science, came in the form of data analysis, and notably data visualization and modelling, has helped various other disciplines in understanding the nature and the behavior of respective fields. One of the disciplines particularly influenced with all these developments came in the form of optical sciences or in short optics, especially in trying to obtain high precision optical systems. High precision optical systems are normally obtained from a testing process called interferometry. Interferometry has been one of the dominant methods for high precision optical testing, with results from interferometers nowadays are digitized and results are obtained from computer data analysis.

The idea behind interferometry are generally the following, a light source is made to split using a beam splitter (half reflecting mirror) in order to travel to two surfaces, one surface is the tested surface (“unknown” surface), and the other is a reference surface (“known” surface). The beams reflected on these 2 surfaces would then recombine and directed towards the detector. Difference in travel length between 2 beams (due to different surfaces, etc.) would result into a phase difference, which from superposition would result in a series of fringes or an interference, which can be digitized and analyzed for determining the current state of the tested surface.

Earlier interferometry methods rely heavily on the idea of finding fringe centers for data. These methods usually caused multiple errors due to wrong identification of the fringe centers. The development of Phase Shifting Interferometry (PSI), first documented in 1966 (Carre, 1966), allows for intensity analysis without the need of fringe centers. In PSI, multiple interferograms recorded through the manipulation of phases by means of phase shifting are recorded. These interferograms then can be run through phase shifting algorithms in order to obtain information on the tested surface.

Development in both PSI along with the subsequent hardware developments over the last few years has led PSI to become a popular method in optical testing. Nowadays, there are various number of PSI algorithms, differing in both its type and the number of phase shifts. Algorithms such as the classic N-bucket (Bruning, 1974), averaging N+1 (Schwider, 1983), or Windowed Discrete Fourier Transform (Surrel, 1996) have been developed over the years with each getting more complex than its previous iterations. With its different form and complexities, these PSI algorithms generally has different features, with each having different sensitivities towards different errors.

There are various errors which could contribute to the accuracy of these PSI algorithms. Highlighting a few of them includes Gaussian noise, harmonics, and phase shift

miscalibrations. As mentioned before, different algorithms have different sensitivities towards these kinds of error. Furthermore, certain algorithms have different sensitivity not only on linear boundary, several PSI algorithms are known to also produce nonlinear errors, which must be mapped out for different number of orders. These establish a problem in trying to pick the most suitable and effective method in response to these different types of errors.

Current available research use simulation to simulate the sensitivity of different PSI algorithms to a certain type of error. After exposing, several different PSI algorithms to different thresholds of error, a graph can be drawn out based on the result of the simulation which shows different algorithms reaction to the increase or decrease of a certain type of error representation. However, in the case of many different algorithms tested, especially across many different of phase shifts, these methods shown to be inadequate, and not effective, especially in terms of its wide use within the industry. Another problem faced is there were not many alternatives to visualize PSI techniques in response to more than one type of error. From these issues, perhaps there is a demand to develop a new method of visualizing the sensitivity of different PSI techniques towards different type of errors.

In conclusion, this project aims to design and develop new method to visualize Phase Shifting Interferometry errors. More specifically, this project attempts to create an interactive and intuitive visualization application which analyzes many different PSI algorithms in response up to 3 different input of error representations, Additive White Gaussian Noise (AWGN), Harmonics, Phase Shift Miscalibration (PSMC). In general, this app allows for users to input 3 different types of error data based on industrial preferences, afterwards, the application would generate a visualization in the form of a 3D graph which in general displays the amount of error caused by the following input supported with a table and details surrounding the PSI algorithms.

This paper serves as a report for the development of this whole project. The first section of this paper introduces the project along with description of Phase Shifting Interferometry techniques. This followed by the second section focuses details surrounding the development and implementation of the project. Finally, the final section serves to conclude the report along with providing suggestion for future improvements.

1.2 Scope and Limitations

This development of this project is restricted by the following scope and limitations:

- Project is a data-based software, therefore, no specific hardware related to interferometry (such as interferometer) are used.
- Simulation of PSI algorithms are focused on to three error representations. This is intended to focus on major error representations as well as to provide better visualization.
 - Additive white Gaussian noise (AWGN)

- Harmonics (Harmonics)
 - Phase Shifting Miscalibration (PSMC)
- Types of PSI algorithms used are focused to the following chosen PSI algorithms. This allows more flexibility in visualization methods, along with a more focused emphasis on widely used PSI algorithms.
 - N-bucket Discrete Fourier Transform (Bruning et al., 1974) (code: N)
 - Averaging N+1 Algorithm (Schwider et al., 1983) (code: AVG)
 - N+1 Bucket Type A (Larkin and Oreb, 1992) (code: PO)
 - Windowed Discrete Fourier Transform (Surrel, 1996) (code: WDFT)
 - N + 3 bucket type (Hibino, 1997) (code: H)
- Maximum frame number (or number of phase shifts) is 20. This is because most application of PSI normally use less than 20 frames as well as to ease up visualization.

2. Literature Review

2.1 Interferometry

Interferometry is a group of techniques which relies on the interference of light waves (or waves in general) reflected from different surfaces in order to produce a series of fringes which in general can be used to depict the current state and shape of the tested surface. Interferometry has been used in many different scientific fields over the years. These includes optical science, astronomy, topography, seismology, quantum physics, and many other fields. In optical interferometry, using visible light waves which relatively have small wavelength, tiny changes in the surface can easily be detected through the interference pattern caused by the slight travel distance of the reflected light waves (Hariharan, 2007)

In general, interferometry processes are normally done using a device called interferometer. Figure 1 shows one of the early versions of the interferometer, the Michelson interferometer (LIGO, n.d.).

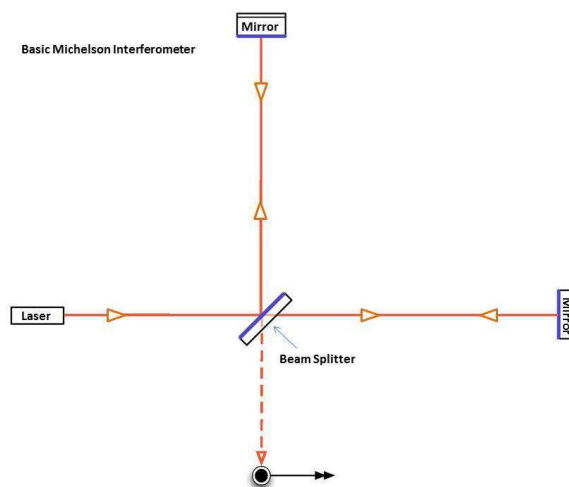


Figure 1. Basic Michelson Interferometer. From “What is an Interferometer?” (n.d.), Caltech/MIT/LIGO Laboratory, <https://www.ligo.caltech.edu/page/what-is-interferometer>

The interferometer works as follows, first, a light source (in recent times, a laser) are fired into a beam splitter. The beam splitter (normally in the form of a half reflective mirror), would then reflect or refract the light wave into two direction. The light wave would then travel towards mirrors located certain distances from the beam splitter. Mirrors (one tested surface, and one reference surface) would then reflect the light waves and let the reflection of both waves rejoined (around the beam splitter) and gets refracted on to a detector. Difference in the surface or distance between the 2 mirrors would translate into difference in terms of phase difference between the 2 rejoined waves. These phase difference would create an intensity pattern which can be captured by the detector (Hariharan, 2007).

Results from the interference of the 2 light waves would show in the form of a series of fringes which can be further translated into interferograms. The intensities recorded within the interferograms can be analyzed to determine the state of the tested surface. In the early days of optical interferometry, static interferograms are widely used in interferometry. This often produces significant problems upon analysis due to the need to find correct fringe centers in order to accurately determine the amount of intensities. The invention of Phase Shifting Interferometry in 1960s successfully mitigate this problem. The use of multiple interferograms through introducing phase shifts in Phase Shifting Interferometry as opposed to static interferograms removes the need to find fringe centers.

2.2 Phase Shifting Interferometry

Phase shifting interferometry is a famous technique for high precision characterization of surfaces that relies on the use of multiple interferograms where a controlled phase shift is introduced (de Groot et al., 2011). Phase shifts can be introduced into an interferometer through various methods. Methods include movement of mirror, tilting glass plates, diffraction grating, rotating half wave plate or analyzer, using optic modulator, or using a Zeeman laser. One of the common methods for a phase-shifting is using piezo-electric transducer in order to push a reference mirror or surface. PZTs normally can move a mirror over a 1 micrometer range (Creath, 1988). An illustration to summarize ways to do phase shifting can be seen in Figure 2.

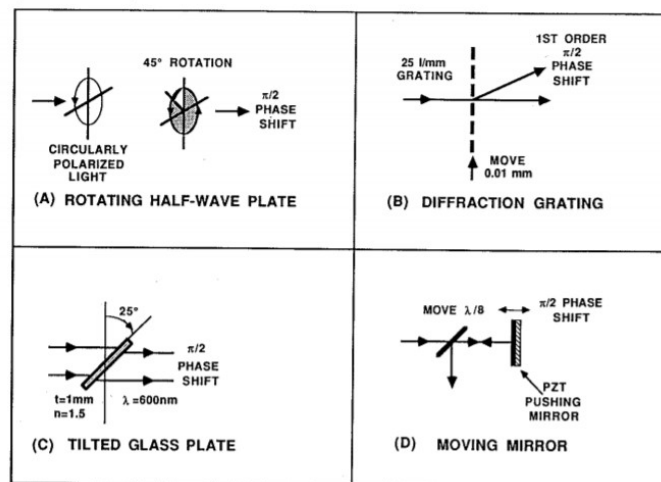


Figure 2. Ways to perform phase shifting. From Creath, K. (1988). *V Phase-Measurement Interferometry Techniques*. Progress in Optics, 349–393. doi: 10.1016/s0079-6638(08)70178-1

Methods on collecting phase data can be done using interferometers like the Michelson's interferometer shown in Figure 2.1. In recent times, however, due to the recent advancements in technology, more sophisticated interferometers have been designed with improved accuracy and design. An example of a more sophisticated interferometer can be seen in Figure 3 (de Groot et al., 2011).

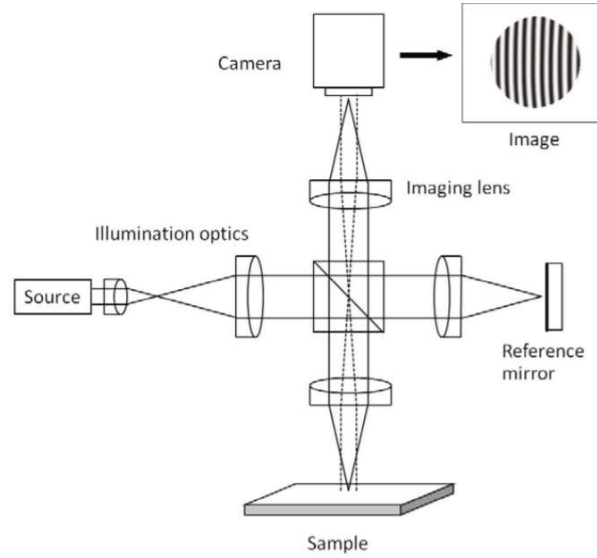


Figure 3. Modern Interferometer. From de Groot P. (2011) *Phase Shifting Interferometry*. In: Leach R. (eds) *Optical Measurement of Surface Topography*. Springer, Berlin, Heidelberg

Intensity values need to be calculated upon obtaining image or interferogram from the camera detector. There are mainly 2 techniques for calculating intensity levels based on the interferograms. The first technique uses the idea of stepping phases up to a determined value in between recording intensity levels, the other technique, named integrating-bucket technique, measures the intensity level through integration of intensity values across a range of phase during the phase shifting process (Creath, 1988).

Intensity values obtained from the interferometer (for Phase Shifting Interferometry) normally can be mapped into equation 2.1.

$$I_i = I_{DC} + I_{AC} \cos(\varphi) \quad (2.1)$$

where I_i represents the intensity value or level (recorded from interferogram, at a certain location), I_{DC} and I_{AC} represents dc and ac intensity, which normally have the property $I_{DC} > I_{AC}$, φ represents the wavefront phase, which normally in terms of PSI can be equated in the form of $\varphi = \theta + \Delta\theta$, where θ is the original phase, while $\Delta\theta$ refers to the phase shift value (de Groot et al., 2011).

From equation 2.1, there exist at least three unknowns, I_{DC} , I_{AC} , and the phase θ . The phase θ normally acts as the main objective in an PSI process. The value I_i is at least known from the interferometry process. Using algebra, the three unknowns would then

need at least 3 equations in order to determine their values. The 3 equations can be obtained through running the interferometer up to 3 phase shifts. This will generate 3 intensity values, which can be translated into three forms of equations. The three-step algorithm is one of the early versions of PSI algorithm utilizing the 3 phase shifts. Over the years, more techniques for PSI have been developed with varying forms and complexities.

2.3 PSI Algorithms

The following serves as a short list and brief description of different types of PSI Algorithms.

2.3.1 Three-step Algorithm

As discussed in the previous section, the three unknowns in Eq. 2.1, requires a minimum 3 measurements of intensity in order to reconstruct the original phase θ . A general case can be formed by means of using a phase shift value α (Malacara, 2007).

Assume the following,

$$\Delta\theta = [-\alpha, 0, \alpha] \quad (2.2)$$

Therefore, rewriting equation 2.1, the following equations can be obtained,

$$I_1 = I_{DC} + I_{AC} \cos(\theta - \alpha) \quad (2.3)$$

$$I_2 = I_{DC} + I_{AC} \cos(\theta) \quad (2.4)$$

$$I_3 = I_{DC} + I_{AC} \cos(\theta + \alpha) \quad (2.5)$$

Using trigonometric identities and algebra, the value of θ can be obtained,

$$\theta = \tan^{-1} \left(\left[\frac{1 - \cos(\alpha)}{\sin(\alpha)} \right] \frac{I_1 - I_3}{2I_2 - I_1 - I_3} \right) \quad (2.6)$$

In order to simplify equation 2.6, $\pi/2$ can be exchanged for α , this results in the following,

$$\theta = \tan^{-1} \left(\frac{I_1 - I_3}{2I_2 - I_1 - I_3} \right) \quad (2.7)$$

On the other hand, inputting the value of $2\pi/3$ as α , would give the following result,

$$\theta = \tan^{-1} \left(\sqrt{3} \frac{I_1 - I_3}{2I_2 - I_1 - I_3} \right) \quad (2.8)$$

Equation 2.7 and 2.8 shows that using PSI, using at least three phases, it is generally possible to reconstruct the unknown phase value of the interferograms using three equations. The three-step algorithm is perhaps the algorithm which requires the least

amount of data and is the least complex algorithm. However, this algorithm can be also considered to be the very sensitive to errors (Malacara, 2007).

2.3.2 Least-squares Algorithm (N-bucket Discrete Fourier Transform)

The least-squares algorithm for PSI came from the idea of performing least-squares fit method on the sinusoidal function which shaped the measured intensity levels of a certain location in order to determine the original phase value. This method was first introduced in 1974 by Bruning, Herriott, Gallagher, Rosenfeld, White and Brangaccio and then further discussed in 1984 by Greivenkamp (Malacara, 2007).

In this case, the Equation 2.1 can first be rewritten as the following,

$$I_i = I_{DC} + I_{AC}(\cos(\theta) \cos(b_i) - \sin(\theta) \sin(b_i)) \quad (2.9)$$

which can be substituted as the following,

$$I_i = a_0 + a_1 \cos(b_i) + a_2 \sin(b_i) \quad (2.10)$$

where

$$a_0 = I_{DC} \quad (2.11)$$

$$a_1 = I_{AC} \cos(\theta) \quad (2.12)$$

$$a_2 = I_{AC} \sin(\theta) \quad (2.13)$$

Note that using equation 2.12 and 2.13, the original phase value θ , can be determined through equation 2.14,

$$\theta = \tan^{-1} \left(\frac{a_2}{a_1} \right) \quad (2.14)$$

By applying the least squares method on the i number of intensity measurements, equation 2.15 can be obtained,

$$\sum e^2 = \sum (I_i - a_0 - a_1 \cos(b_i) - a_2 \sin(b_i))^2 \quad (2.15)$$

Error (e) can be minimized through differentiating equation 2.15 and setting the value of the derivative of the error to 0. Solving for the derivative equation would lead to the solving of a_0, a_1, a_2 in terms of I_i and the phase shift b .

Substituting the values of a_1, a_2 obtained from the least-squared method to equation 2.14, would then show a new method for PSI. In fact, a special case where $\sum \sin(b)$, $\sum \cos(b)$, and $\sum \sin(b) \cos(b)$ equals to zero, would result in the following formula as shown below (eq. 2.16).

$$\theta = \tan^{-1} \left(\frac{-\sum I_i \sin(b_i)}{\sum I_i \cos(b_i)} \right), \quad b_i = \frac{i2\pi}{N} \quad (2.16)$$

where I_i refers to the intensity level of a point at the i th phase shift, and b_i refers to the total amount of phase shift at the i th iteration, i in this case, refers to a value between 1 to N , where $N > 3$ and N is the number of different phase shifts recorded, or the frame number.

The N -bucket DFT family of algorithms refers to the algorithm generated from eq. 2.16. Examples of the N -bucket DFT algorithm can be seen in Table 1.

Frame Number (N)	Formula	Phase Shift
3	$\theta = \tan^{-1} \left(\frac{\sqrt{3}(I_3 - I_2)}{2I_1 - I_2 - I_3} \right)$	$b_i = (i - 1) \frac{2\pi}{3}$
4	$\theta = \tan^{-1} \left(\frac{I_4 - I_2}{I_1 - I_3} \right)$	$b_i = (i - 1) \frac{\pi}{2}$
6	$\theta = \tan^{-1} \left(\frac{\sqrt{3}(I_5 + I_6 - I_2 - I_3)}{2I_1 + I_2 - I_3 - 2I_4 - I_5 + I_6} \right)$	$b_i = (i - 1) \frac{\pi}{3}$
8	$\theta = \tan^{-1} \left(\frac{-\sqrt{2}(I_2 + \sqrt{2}I_3 + I_4 - I_6 - \sqrt{2}I_7 - I_8)}{\sqrt{2}(\sqrt{2}I_1 + I_2 - I_4 - \sqrt{2}I_5 - I_6 + I_8)} \right)$	$b_i = (i - 1) \frac{\pi}{4}$

Table 1. Examples of N -bucket DFT PSI Algorithms

2.3.3 Family of Averaging Algorithms

Despite being a widely used method for PSI in around the 1970s, however, the N -bucket DFT family of algorithms in general is not free from phase shift errors. A study by Schwider, Burow, Elssner, Grzanna, Spolaczyk, and Merkel in 1983 attempted to analyze as well as provide a solution for the errors faced by this technique. The study proposed that averaging PSI calculations with a certain value of phase offset can help cancel out some of the errors. Equation 2.17 demonstrates the method to generate an averaging PSI technique.

$$\theta = \tan^{-1} \left(\frac{N_1 + N_2}{D_1 + D_2} \right) \quad (2.17)$$

N_i and D_i represents numerator and denominator of the i -th run of the algorithm. In this case, averaging is performed across two runs. Using a phase offset of $\pi/2$ and the base algorithm N -bucket DFT with 4 frames (see Table 2.1), an averaging algorithm with 5 frame numbers can be obtained as seen in Equation 2.18.

$$\theta = \tan^{-1} \left(\frac{2(I_4 - I_2)}{I_1 - 2I_3 + I_5} \right), \quad \Delta\theta_i = (i - 1) \frac{\pi}{2} \quad (2.18)$$

Over the years, more ways to incorporate averaging for PSI have been researched. Research by Schmit and Creath in 1995 and 1996 develop methods for extending

averaging techniques as well as averaging multiple algorithms. Table 2 shows several averaging algorithms which were derived on the 4-framed N-bucket DFT.

Frame Number (N)	Formula	Phase Shift
4 (base)	$\theta = \tan^{-1} \left(\frac{I_4 - I_2}{I_1 - I_3} \right)$	$b_i = (i - 1) \frac{\pi}{2}$
5	$\theta = \tan^{-1} \left(\frac{2(I_4 - I_2)}{I_1 - 2I_3 + I_5} \right)$	$b_i = (i - 1) \frac{\pi}{2}$
6	$\theta = \tan^{-1} \left(\frac{4I_4 - 3I_2 - I_6}{I_1 - 4I_3 + 3I_5} \right)$	$b_i = (i - 1) \frac{\pi}{2}$

Table 2. Examples of Averaging Algorithms with 4-frame N-bucket DFT as base.

2.3.4 Other PSI Algorithms

Other techniques to develop PSI algorithm have also emerged with different forms and different complexities. Below are some techniques that have been chosen for this project.

2.3.4.1 Windowed Discrete Fourier Transform

Windowed Discrete Fourier Transform or WDFT is a method introduced by Surret in 1996. This algorithm is developed with the purpose to be insensitive towards Harmonics up to certain order (based on Frame Number), as well insensitive towards phase shift errors. Equation 2.19 demonstrates a general formulation of the WDFT algorithms.

$$\theta = \tan^{-1} \left(\frac{-\sum_1^{N-1} k(I_k - I_{2N-k}) \sin\left(\frac{k2\pi}{N}\right)}{NI_N - \sum_1^{N-1} (I_k + I_{2N-k}) \cos\left(\frac{k2\pi}{N}\right)} \right), \quad \Delta\theta_k = \frac{k2\pi}{N} \quad (2.19)$$

In this case, N does not refer to frame number, but can be noticed as a counter for the generating the full set algorithm. Therefore, for N=3, a 5-framed algorithm will be generated. A 7-framed algorithm with N = 3 can be seen in equation 2.19. Notice that the value of the phase shift could exceed 2π .

$$\theta = \tan^{-1} \left(\frac{-I_1 + 3I_3 - 3I_5 + I_7}{4I_3 - 2I_5 - 2I_7} \right), \quad \Delta\theta_k = \frac{k2\pi}{3} \quad (2.20)$$

2.3.4.2 N+3 Bucket Algorithm

Another family of algorithm is introduced by Hibino in 1997. This algorithm is designed to address the issue for non-linear and spatially non-uniform phase shift errors. The general formation of the algorithm can be seen in equation 2.21.

$$\theta = \tan^{-1} \frac{\frac{1}{4}(I_1 + I_2 - I_{N+3} - I_{N+4}) \frac{\sin\left(\frac{3\pi}{N+2}\right)}{\sin^2\left(\frac{2\pi}{N+2}\right)} + \sum_2^{N+3} I_j \left[\sin\left(\left(\frac{2\pi}{N+2}\right)\left(j - \frac{N+5}{2}\right)\right) \right]}{\frac{1}{4}(I_1 - I_2 - I_{N+3} + I_{N+4}) \frac{\cos\left(\frac{3\pi}{N+2}\right)}{\sin^2\left(\frac{2\pi}{N+2}\right)} + \sum_2^{N+3} I_j \left[\cos\left(\left(\frac{2\pi}{N+2}\right)\left(j - \frac{N+5}{2}\right)\right) \right]} \quad (2.21)$$

with phase shift determined by $\Delta\theta_i = \frac{2\pi}{N+2} \left(j - \frac{N+5}{2} \right)$.

An example implementation for this algorithm with $N = 2$ can be seen in equation 2.21a.

$$\theta = \tan^{-1} \left(\frac{I_1 - 3I_2 - 4I_3 + 4I_4 + 3I_5 - I_6}{-I_1 - 3I_2 + 4I_3 + 4I_4 - 3I_5 - I_6} \right), \quad \Delta\theta_k = \frac{\pi}{2} \left(k - \frac{7}{2} \right) \quad (2.21a)$$

2.3.4.3 $N+1$ Bucket

The $N+1$ bucket is another family of algorithms developed by Larkin and Oreb in 1992. This family of algorithm consist of two forms coined Type A and Type B. These family of algorithm are tailored such that it produces a symmetrical phase shifting algorithms with Type B is further tailored for handling phase shift errors. The general form of Type A and Type B algorithms can be seen in Equation 2.22a and 2.22b, respectively.

$$\theta = \tan^{-1} \frac{\sum_2^N I_j \left[\sin\left(j \frac{2\pi}{N}\right) \right]}{-\frac{I_1 + I_{N+1}}{2} - \sum_2^N I_j \left[\cos\left(j \frac{2\pi}{N}\right) \right]} \quad (2.22a)$$

$$\theta = \tan^{-1} \frac{\frac{-I_1 + I_{N+1}}{2} \cot\left(\frac{2\pi}{N}\right) - \sum_2^N I_j \left[\sin\left(j \frac{2\pi}{N}\right) \right]}{-\frac{I_1 + I_{N+1}}{2} - \sum_2^N I_j \left[\cos\left(j \frac{2\pi}{N}\right) \right]} \quad (2.22b)$$

An example implementation for this family (Type A) when $N = 3$, can be seen in Equation 2.22c below,

$$\theta = \tan^{-1} \left(\frac{\sqrt{3}(I_2 - I_3)}{-I_1 + I_2 + I_3 - I_4} \right), \quad \Delta\theta_k = \frac{2\pi}{3} (k - 1) \quad (2.22c)$$

2.4 Error Analysis

There are various error sources which attribute to an error in the calculated phase in Phase Shifting Interferometry. Error sources exists in the form of phase shift errors (phase shift calibration errors), detector nonlinearities (harmonics), vibration errors air turbulence, quantization errors, interferometer optical errors, etc. (Malacara, 2007). Methods for simulating and analyzing PSI error sources normally relies on the use of mathematical representations of these errors. This project focuses on simulating PSI algorithms using the following error representations, Additive White Gaussian Noise, Harmonics, and Phase Shift Miscalibration.

In general, different PSI algorithms have different sensitivities towards different kinds of error. Various researches across the years have aimed to develop a better method to try to minimize sensitivity towards these errors. The averaging techniques first introduced by Schwider in 1983 attempted to reduce linear phase shift errors from the N-bucket DFT technique. The WDFT algorithms introduced by Surrel in 1996 are made to be insensitive towards harmonics towards a certain order as well as towards phase shift miscalibration. The algorithms developed by Hibino in 1997 were a further refined family of algorithms which were designed to withstand non-linear and spatially non-uniform phase shifts (Malacara, 2007).

2.5. Selected Error Representations

2.5.1 Additive White Gaussian Noise

Additive White Gaussian Noise or AWGN is a noise model normally used in information analysis. In modelling, AWGN is also widely used due to its ability to simply demonstrate random behaviors that occur in nature. In PSI, additive noise is used to analyze the performance of PSI algorithms against noisy interferometers, which may be plagued with various error sources.

A paper published by Servin in 2009 demonstrates how additive noise can be modeled into an interferogram intensity equation (Eq. 2.1). The equation can be seen in Eq 2.23.

$$I_i = I_{DC} + I_{AC} \cos(\varphi) + n_A \quad (2.23)$$

In the case of an additive white Gaussian noise, the Gaussian distribution or normal distribution function can be used as a substitute for the value of n_A . In computer aided simulation of PSI algorithms, the values of n_A is attributed to a pseudo-random Gaussian distribution function with mean of 0 and a standard deviation of 1. From that point, a multiplier can be added to test out the sensitivity of the PSI algorithms towards different levels of noise.

2.5.2 Harmonics

In general, not all errors faced by PSI algorithms can be modeled as a linear function. Some PSI algorithms might be insensitive towards linear error sources, however, some of these algorithms might be sensitive towards a nonlinear type of error. Errors from detector nonlinearities and several other errors sources can be represented in the form of a harmonic analysis.

In general, harmonic analysis mathematical analysis which uses a form of harmonic series to model the data it represents. Harmonic series is a mathematical series where its member consists of sinusoidal waves with differing frequencies in the form of a multiple of a certain chosen fundamental frequency. Simulation of harmonic errors in PSI, can be seen in Equation 2.24 below.

$$I_i = I_{DC} + I_{AC}(\cos(\varphi)) + I_{AC} \sum_2^N k_j \cos(j\varphi) \quad (2.24)$$

In eq. 2.24 above, the harmonics error is simulated in the form of the summation $I_{AC} \sum_2^N k_j \cos(j\varphi)$, where here j represents the order harmonics up to a value of N , and k_j represents a multiplier to indicate the level of exposure of the simulation towards the harmonic of order j .

2.5.3 Phase Shift Miscalibration

The reliance of PSI towards phase shifting has made phase shift error a major concern for PSI. Most of the time, phase shift errors are caused by phase shift calibration errors or miscalibration (PSMC). An equation to describe phase shift errors can be seen in Equation 2.25.

$$I_i = I_{DC} + I_{AC}(\cos(\theta + \Delta\theta')) \quad (2.25)$$

Eq. 2.25 shows $\Delta\theta'$ which can be described in the following equation.

$$\Delta\theta' = \Delta\theta + \varepsilon(\Delta\theta) \quad (2.26)$$

The value of the error or miscalibrated phase shift can be considered as a deviation from the original phase shift value. In the event of a phase shift error, therefore, phase shift increments in most frames would result in a different value, which in turn would allow the interferometer to output incorrect interferogram data. In the case of a N -bucket DFT algorithm which relies on Least-Squares fit, phase error in this case would lead to a least-squares fitting of a different set of points, which in turn results in its inaccuracy. Methods such as the family of averaging algorithms are developed in order to try mitigating the problems caused by phase errors. Further studies have also shown that while certain algorithms are insensitive to linear calibration errors, these algorithms might still be sensitive in terms of nonlinear calibration errors. Algorithms such as the $N+3$ bucket family algorithms are designed to be immune to phase shift error up to a certain degree of order (quadratic).

In this project, phase shift miscalibration error will be simulated using the polynomial fit mathematical model. An illustration on how the phase miscalibration error is simulated can be seen in Equation 2.27 below.

$$\varepsilon(\Delta\theta) = \sum_{k=0}^{inf} n_k \Delta\theta^k \quad (2.27)$$

In eq. 2.27, n_k refers to the coefficient of the polynomial equation at degree k which can be modified as the simulation sees fit. Using polynomial fit, it is possible to derive a model which simulates both linear and nonlinear phase shift calibration errors.

2.6 Development Platforms

2.6.1 REST API

REST, or Representational State Transfer, refers to a software architecture style for distributed systems first introduced by Roy Fielding in 2000. In general, the REST uses the following guideline principles where in general a REST system in a Client-Server model, stateless, cacheable, with uniform interface, a layered system, and implements an optional code on demand (usage of reusable libraries).

The main idea of REST relies on exchanging and delivering resources. Resources may come in many different forms, and through identifying resources that a client need or require, a client may be able to receive the resources. Client does not need to know the real form of the REST itself, if the client is capable of exchanging resources, clients in general can visualize their own states and position in their own point of view. In general, REST utilizes HTTP to regulate the flow of resources from and towards the client.

This project uses REST API as a method to exchange information from Backend to Frontend and vice versa. In the case of this project, this project uses Django REST framework for Backend and React JS for Frontend Application. Using the REST API, clients (in this case, the Frontend) can request and receive information as they say fit from the REST API. A simple application for this REST architecture is the following, REST allows the Backend server to serve as in an advanced remote function, where the client can send in their input in the form of web requests, and the server can receive and process these web request and return it back towards the client in the form of resources or data-only (no template). The use of REST API allows for ease of development for single page web-application, where not all request will be delivered in the form of a normal web browser request (HTTP request and receives back HTML template).

3. Project Development

3.1 Overview

This section describes the project development process in detail over the duration of this project. Development of this project can be best split onto 5 different phases or iterations. Table 3 below shows the 5 iterations of this project as well as the period it takes for each iteration.

#	Phases/Iteration	Period/Timestamp
1	Early Development (React Native)	August 2019 – December 2019
2	Backend Development	November 2019 – January 2020
3	Frontend Development	December 2020 – February 2020
4	Integration and Debugging	February 2020 – March 2020
5	Refinement	April 2020 – May 2020

Table 3. 5 Project Development Iterations/Phases

The following sections of this chapter contains detailed description of each of the development phases. Each section shall include an overview, an outline of project schedule, issues faced during development, as well as a conclusion.

3.2 Early Development Phase

3.2.1 Overview

The early development phase starts from where the project was first announced. The first few weeks of this development phase are used for research on topics of PSI as well as PSI algorithms along with research on platforms for mobile app software development and tools. Results on early research on PSI leads to a further understanding towards how PSI conducted as well as how these algorithms can be modeled and simulated to conduct error analysis. These further understanding on the topic led to a foundation on how the intended application should perform as well as obtaining an early requirement set for this app.

These foundations as well as the results from research on several platforms for mobile app development led to a list of shortlisted platforms. The first recommended platform for this development is using Android Studio. Android Studio in general is one of the best software platforms for Android app development. But this leaves a problem for iOS users, Android Studio, as the name suggests, is only for Android operating system, therefore, there will be a need for development under the iOS banner, which can be done using Swift. These development platforms have different properties as well as different set of available tools and libraries, this will presumably cause issues in later stages of development, especially in terms of design and debugging.

Another choice for development is through the use of a new mobile app development environment called Native. Using Native, it is possible to develop a vanilla app (normally web-based) which is intended to run on current mobile devices of different OS. Due to the flexibility of the Native development platform as well as the considerations mentioned in the previous paragraph, one of the development environments for Native web apps, React Native, which uses JavaScript as its programming language, is used for the development of this project.

Afterwards, an early draft of project prototype as well as software development plan can be formulated. Initial project plan includes defining the PSI simulation model. This process includes developing a working PSI simulator, choosing and introducing the appropriate error representations, defining the required data structures, deriving a select number of PSI algorithms for prototyping (into the selected data structure), until finally establishing an input and output mechanism. This process results in a general structure of the app, inputs are in the form of textboxes (1 for AWGN, and 9 for each order of PSMC and Harmonics). Outputs are in the form of a graphical representation of the three phase errors calculated from simulation.

Following the definition process, software development can finally start. The software development environment is as follows, software (development server) is deployed in PC environment where emulators or mobile devices can pick up the data from the server and run a version of the Native mobile app on their screens. Testing and debugging are performed through deploying the app on mobile devices and running functionality tests. Error messages can be displayed on the PC console.

In this first phase of project development, project is developed under a full stack React Native environment. In this case, UI mechanics as well as PSI simulation or data processing are all developed using JavaScript. Input and output mechanism change and evolve overtime. Especially in terms of output mechanism, the graphical representation has been decided in the form of a scatter plot and followed by an interactive table. Various other enhancements are added as requirements keep changing over the course of this phase. An outline on the project development on this phase can be seen in Table 4.

3.2.2 Development Phase Outline

#	Development	Estimated Period	Details
1	Literature Review on PSI	August – Sept 2019	-
2	Review on Development Platforms	August – Sept 2019	React Native chosen
3	PSI Model Definition and Development	September 2019	Early Lo-Fi prototype formulated

4	Environment Setup	September 2019	-
5	Dummy Software Initialization (Platform familiarization)	September 2019	Develop a dummy software as foundation for further developments
6	PSI Simulation Model on React Native	September 2019	Develop the mathematical calculation model for PSI under JavaScript functions
7	Develop Parent/Container Component	September 2019	Develop Main Component which houses all other components
8	Develop Simple Input Output Component	September 2019	Create simple I/O UI and prepare simple integration methods (for PSI simulation functions)
9	Develop Table, Graph Component	Sept – Oct 2019	Early development for table component as well as integrating output with table component
10	Improve Input Component	October 2019	Input component re-design as well as minor improvements
11	Improve Table Functionalities	October 2019	Improve functionalities of table component with the goal of making a more interactive table
12	Code Refactoring	October 2019	-
13	Incorporate more PSI algorithms	October 2019	-
14	App improvements to provide better user closure	Oct – Nov 2019	Development to address issues with lag, state update errors, etc. (Lag issue not solved)
15	Improve Graph Component	Oct – Dec 2019	Further development for interactive graph as well as finding ways to incorporate 3D graph (Failed)
16	Latex Rendering	Nov – Dec 2019	Latex rendering for mathematical formulas (Failed, no method found)
17	Extensive Bug Fixes	Oct – Dec 2019	Bugs as well as hidden bugs can be found in various parts of the code.

Table 4. Early Development Phase Outline

3.2.3 Phase Issues

Development during the first phase in general have faced various issues. Major issues found during this development phase includes the following,

- No support found for 3D chart components
- No solutions found for React Native Latex rendering

- Software instability due to heavy mathematical processing
- Software lag issues (due to the PSI simulation, as well as processing power of a mobile device)
- UI gesture problems (cannot incorporate pan, zoom, and rotate combined with scroll onto the UI)
- State update issues (React Native uses state to maintain current software processes state, this is plagued with problems in state refresh)
- Very slow debugging process (due to the need to fetch the program onto mobile, and re-download program every software modifications, emulator is even slower)
- Constantly changing requirements (Software requirements and expectation during this period is not fixed and constant changes to requirements led to inconsistent software development)
- Small screen size and small processing power on mobile devices led to inability to improve in various sections
- Various errors in PSI simulation functions due to absence of Mathematics tools
- Software developer inexperience with the development platform as well as PSI mechanisms
- Too many bugs (due to too many modifications and improvements need to be added every time, projects have been overrun by bugs)

By mid-October up until December 2019, project development had become slow and plagued with too many issues. Results during those periods have been very minimal. Some required improvements such as the 3D graphs and Latex rendering cannot be developed, and a lot of time are wasted in trying to implement these features (with no success). Implementing fixes (both in terms of bug fixes as well as improvements fixes) have also been plagued by the long debugging process which rendered the development no longer feasible and no longer effective.

3.2.4 Conclusion

The first development phase is considered a total failure. Several required requirements cannot be developed under React Native. Screen size for mobile devices are deemed too small to show an acceptable UI interface. Processing power for mobile devices are inefficient in trying to render both the UI as well as the PSI simulation processes. Workflow had also become very inefficient due to the lag issues as well as the slow debugging processes. At this point, project need to re-hauled.

3.3 Backend Development

3.3.1 Overview

The second phase of the development is aimed in trying to avoid most of the mistakes done in the first phase. One of the major issues faced in the first phase is the lag issues which stemmed out from the PSI calculation phase done in a full stack frontend environment. This second phase is aimed to develop a web-based app (not Native) with a web-based server designed to ease up PSI simulation processes.

This development phase relies on using Python instead of JavaScript due to the extensive library available for Python for mathematics related processes (Numpy, Matplotlib) as well as for web app development. Django web framework is chosen as for the foundation of this Backend part of the application. Reasons for the selection of Django is mostly due to its simplicity, flexibility, and scalability.

The initial plan for this development phase is to develop a full stack web app under Django. However, limitations towards input mechanism as well as reliance of multi-page web applications upon using pure Django (and HTML) led this idea to be reconsidered. The idea of developing a pure Django web app is finally axed and replaced with developing a Django Representational State Transfer API (REST API). REST API generally allows Django to act as a core backend module, which receives request in terms of PSI error inputs and send back a reply towards the sending interface (the Frontend server) on the data it needs. Table 5 shows a brief outline for this phase.

3.3.2 Phase Development Outline

#	Development	Estimated Period	Details
1	Review on web server frameworks	Nov – Dec 2019	-
2	Small Scale Development	Dec 2019	Development of small-scale web app for early testing purposes
3	PSI Simulation Module	Dec 2019	Develop a Python module for PSI calculations
4	Full stack Django web app development	Dec 2019	Development of Django full stack as per original plan (as well as to ease backend testing)
5	Code refactoring	Dec 2019	Code refactoring in preparation to port towards REST API
6	REST API Development	Dec 2019 - Jan 2020	Development of the REST API for PSI simulating purposes
7	REST API Debugging	Jan 2020	Debug REST API through various request/reply testing

8	Latex Formula Generator	Jan 2020	Since algorithm data is available only in Backend, therefore, a system to fetch formula data is required
9	PSI Algorithm Generation Modules	Jan 2020	Kickstart development of automated PSI Algorithm generation
10	Bug fixing	Jan 2020	-

Table 5. Backend Development Phase Outline

3.3.3 Phase Issues

Most issues faced in the first stint of the development could be mitigated through the development of this Backend web server using Python. However, this does not mean the development phase is free of issues, issues are found especially at the later stages of development. Major issues in development can be seen in the list below,

- Wrong algorithms
There is an uncertainty in debugging PSI results, most of the time errors are due to wrong algorithms (different papers have different interpretations), but there is also a possibility where phase unwrapping or other errors occur during this code. The uncertainty on the sources of a bug, lead to prolonged debugging time.
- Phase unwrapping issues
The PSI simulation is plagued with phase unwrapping errors. These errors can normally go undetected as several algorithms are tested. This results in a never-ending debugging process for phase unwrapping (sometimes fixing the issue at one point in time lead to another phase unwrapping bug)
- Other PSI errors
Other errors such as, wrong output, fraction simplifications, error modelling, and other minor bugs, exists throughout software development, the nature of the error or bug sometimes is very trivial which led to problems during debugging that consume lots of time to debug.

In general, there are virtually no major issues in terms of technical software development. All major issues are centered around fixing the PSI simulation function

3.3.4 Conclusion

This development phase can be considered as a relative success. The REST API successfully ensure smoothness of the PSI simulation process. Using the REST API also localizes the problems faced in the PSI and UI processes, ensuring an easier to maintain software development.

3.4 Frontend Development

3.4.1 Overview

Due to the limitations found in both the React Native development as well as the Django full stack development, considerations for developing a separate Frontend software for this project arises. One of the major limitations for the Django Full Stack Development is its inability to provide a dynamic form (for input). This combined with the fact that pure Django web apps would result in a traditional multi-page web application (as it uses regular HTML for web layout) instead of the much interactive single-page application led to finally develop a separate Frontend application.

The popular JavaScript library React is chosen as a basis for the Frontend development. React in general has a similar architecture to React Native. In fact, the React Native framework is developed upon the basis of React and modified for Native mobile application. However, the degree of flexibility in terms of available libraries, use of components, as well as the support for React development among the community led to React being the easily preferred choice compared to React Native. One defining example for this is the following, React allows developers to render their own components using HTML, while React Native requires developers to use a pre-defined basic components (i.e. text, label, etc.) which in turn is not flexible and not easy to modify.

With the decisions being made, project development can proceed. The Frontend application is generally made of a component, the Parent or Container Component. This parent component houses 4 other main components, with each component having their own respective duties and functions within the UI. These 4 components include component such as FormInput (for input mechanism), Graph (for displaying 3D graph), Table (for displaying PSI error values), and Details (for PSI algorithm information-related display). These 4 main components may house other smaller components as needed.

These 4 main components along with the Parent communicate with each other through exchanging states and props. For example, the FormInput component needs to register input data, then send them web server to request a reply, and communicate with the other components such as Graph and Table to display the output. Some of the communication logic have been implemented in React Native. But in general, many of the communication logic needs to be overhauled or redesigned to cater for the different set of libraries as well as the use of REST API (from Backend) to obtain data from the PSI simulations.

A general outline of this process can be seen in Table 6.

3.4.2 Phase Development Outline

#	Development	Estimated Period	Details
1	Review on React.js	Jan 2020	-

2	Small Scale Development	Jan 2020	Development of small-scale frontend web app for early testing purposes
3	FormInput Component Development	Jan 2020	Develop dynamic form mechanism
4	Table Component Development	Jan 2020	-
5	Code refactoring	Jan 2020	Code refactoring, develop Parent/Container Component
6	Graph Component Development	Jan 2020	-
7	Asynchronous API Design and Development	Feb 2020	Design and develop Frontend API calls mechanism
8	Details Component development	Feb 2020	Develop component for PSI algorithm information block
9	Component Improvements	Feb 2020	Improve major components, introduce more sub-components (Latex, etc.), CSS implementation
10	Bug fixing	Feb 2020	-

Table 6. Frontend Development Phase Outline

3.4.3 Phase Issues

Major issues faced during the Frontend Development Phase is concentrated on improving design. The expectations for an interactive design are high, and this creates a problem where development is bottlenecked by the readily available design ideas and design implementations. The following list shows several issues faced during this development phase,

- **Graph Design Issue**
Standard graphing library as well as standard scatterplot designs generally were unable to complement the standards for improvement within the industry. Therefore, there is a need to develop a new method to display the PSI algorithms which fit the required standards/expectations. Lack of experience in interactive data visualization as well as data modelling techniques led to a bottleneck in terms of design improvements
- **CSS Issues**
CSS or cascading style sheet is a language that depicts the styling for HTML components. While this is generally not an issue, however, frequent design changes as well as incorporating newly added design ideas has led to a need to redesign the CSS for many different HTML components. This has led to the process of coding CSS very time consuming, especially when designing edge cases details. (Need a better protocol for updating CSS)
- **Synchronization of State Update Rendering**

API calls normally uses asynchronous methods; this has led to synchronization issues in trying to perform REST API calls in conjunction with other processes. Another problem faced is in the form of Latex rendering. Several Latex failed to render or rendered a wrong data upon API fetches. While this issue has been solved, it took a considerable amount of time to figure out the perfect state update mechanisms as well as to perform debugging for all test cases.

- **Frequent redesigns**
In complement with the CSS issue, frequent UI redesign (in order to cater for newly added design ideas) led to several overhauls on major parts of components. While the problem has been well diversified as every component has its own tasks, however, some redesigns are quite time consuming due to the need of changing the rendered HTML codes (also very time-consuming in terms of trying new design ideas, which might or might not be used)
- **Other design issues**
Design in general is a quite an abstract knowledge. Therefore, sometimes it is very hard to develop a good design for different functionalities. While all major functionalities of this web app are technically done, the problem now relies on how to provide a satisfactory design for these functionalities.

All these issues combined with the bugs they brought with them, has led to a generally slow improvement especially at the latter stage of development. Despite the success in developing technically working components for the needed requirements, however, the problem still relies on how to provide a better design for this app.

3.4.4 Conclusion

This development phase can be considered as relative success when compared with the first development phase. However, the question still lingers on how to further enhance the available design with newer ideas and a better perspective in order to provide better experience for the users.

3.5 Integration and Debugging

3.5.1 Overview

This development phase is generally focused on integrating both the frontend and backend software as well as fixing bugs from the previous development section as well as new bugs that came along during this development phase. Integration perhaps can be performed seamlessly due to the availability of interfaces which has been developed and researched on the previous development phases. On the other hand, challenges still exist in the form of bugs and improvements needed to be fixed and added to the software.

Integration works were generally performed through the deployment of the 2 development servers (backend and frontend). Afterwards, checks and tests need to be done in order to make sure the data being called upon by Frontend server towards the Backend server is

correct and synchronized. For this purpose, several modifications in the Frontend software (i.e. managing state and props changes, etc.) needs to be implemented, while in the Backend site, several setups (i.e. installation of libraries, change of settings, etc.) need to be done to ensure the data request is accepted and can be replied back towards the requester.

After performing integration, the entire time is dedicated to fixing unsolved bugs from the previous phases, extensive debugging (to find new bugs), as well as providing improvements for unsolved issues (design improvements, algorithm checks, etc.). This is probably the most time-consuming process throughout all development phases, because as time goes by, problems became more complex and ideas began to run out.

A general outline of this phase can be seen in Table 7 below.

3.5.2 Phase Development Outline

#	Development	Estimated Period	Details
1	Review on Frontend and Backend system	Feb 2020	-
2	Integrate Frontend and Backend	Feb 2020	-
3	Fix API calls synchronization issues	Feb 2020	Solve problem where API calls and state changes are not synchronized
4	Debug simulation models	Feb 2020	Find issues related to the wrong phase error outputs
5	Fix phase unwrapping problem	Feb 2020	Fix phase unwrapping in PSI simulation module
6	Fix state update errors	Feb 2020	Fix errors in Frontend caused by wrong state update mechanism
7	Fix Algorithm Generator Module	March 2020	-
8	Fix Details component (Latex not rendering)	March 2020	-
9	Debug PSI simulation models (part 2)	March 2020	Solve new issues related to PSI simulation
10	CSS improvements and Design improvements	March 2020	Fix bad design and layout problem

Table 7. Integration and Debugging Phase Outline

3.5.3 Phase Issues

Most issues are carryovers from the previous development phases. Some of those issues were solved during this period while some of them remains a bottleneck for satisfactory completion of this project. The following list below shows major issues for this development phase,

- **PSI simulation errors**
There are many problems with PSI simulation as closer lookup are performed. Some issues are unsolved issues from previous development phases, while others are newly found issues. However, the biggest issues found on this topic were due to issues that are previously thought to be solved but remains persistent after further random tests are performed.
- **Design Issues**
The issue with developing a better and more robust design still lingers up towards this development phase. Issues such as better graph design for depicting the PSI error remains one of the biggest issues to be solved. Several other minor design issues include management of edge case UI improvements.

3.5.4 Conclusion

These issues remain a bottleneck towards a satisfactory completion of this project. More time allocation is required to be mainly focused on refinement of this project as well as incorporating newer design ideas while taking advantage of reusable components. Better standardization needs to be implemented to help omit unreasonable bugs as well as to effectively use the remaining time.

3.6 Refinement

3.6.1 Overview

The refinement phase is an additional phase implemented in order to solve all remaining issues within the project. As far as the project development is concerned, at the point where project development was supposed to be on its finishing stages (debugging phase), there are still unsolved issues that might require major changes in certain number of components.

Most of the time during this phase will be used to further improve visualizations and design. Other than that, bug fixes in both Backend and Frontend Software needed to be implemented. Considerations is also added in the case of more algorithm types can be added into the fold. This however is subject to the availability of a generalized method for generating the algorithms, as well as visualization limitations (whether or not adding more algorithms can help to make the design better).

An outline of this development stage can be seen in Table 8 below.

3.6.2 Phase Development Outline

#	Development	Estimated Period	Details
1	Final Fix on PSI Simulation Module	Mar – Apr 2020	-
2	Final Fix on Algorithm Generation Module	Mar – Apr 2020	Development of small-scale frontend web app for early testing purposes

3	Improve Design for Graph	Mar – Apr 2020	Develop a Python module for PSI calculations
4	Improve Overall Design	Mar - Apr 2019	Development of Django full stack as per original plan (as well as to ease backend testing)
5	Bug fixing	Apr - May 2020	-

Table 8. Refinement Phase Outline

3.6.3 Phase Issues

Most of the issues faced during this stage would be focused around fixing previously major issues, as well as, fixing newly found issues (or new issues formed due to the added improvements. In general, as this phase is the final phase for this project, the best approach towards this phase is to ensure all issues are solved and prevent rise of new issues (preventive programming).

Extensive Preventive programming or programming with the idea of avoiding development of new bugs might be very time consuming as all edge cases need to be free of issues before proceeding with the next action.

3.6.4 Conclusion

As this report is written, the final phase will be the last available phase for this project considering the remaining time available. In that case, this phase was aimed to solve all the major issues concerning design as well as edge case bugs that might still plagued the application. Any extra improvements that failed to mature during this stage (i.e. still have to many bugs or have a app-breaking issue) will be discarded.

4. Design and Implementation

4.1 Overview

Current Design Implementation is as follows, the project is split into Backend and Frontend.

Backend in general consists of 3 components, the PSI simulation module, the algorithm generator modules, as well the web app modules. The web app modules in general uses the Django Model-View-Template design. The Model module is not included for this project because this project did not require a database system, while the Template module is only used during the full stack web app development and would not be used as the project moves towards REST API. The View module similar to a controller logic which in turn serves the requests sent from the Frontend software, integrate with the PSI modules, and returns a reply to the Frontend software. Django REST framework is used as an additional module for this Backend site in order to allow the original Django framework to work in a manner of REST API. Other additional modifications for the Django framework are centered around settings as well as URL routing.

On the other hand, the Frontend software is generally made of a single container component which houses 4 other main components. These 4 other main components consist of the FormInput, Graph, Table, and Detail Component. These main components may house several other sub-components as seen fit. The FormInput component handles the dynamic form, Graph component handles the 3D graph, Table handles the interactive, and Detail handles detail display of a selected PSI Algorithms. These 4 components communicate through each other through exchanging props and states which most of the time are regulated within the Parent component. These components also communicate with the Backend server to obtain necessary data to be displayed. Some of these API fetches are done in the Parent, while some are performed individually in each component. The main idea of this Frontend web app development centers around managing states to ensure the correct data and views are displayed. Subsequent components might also be added upon refinement.

The following sections below describe each of the components within both the Frontend and Backend software in detail.

4.2 Backend

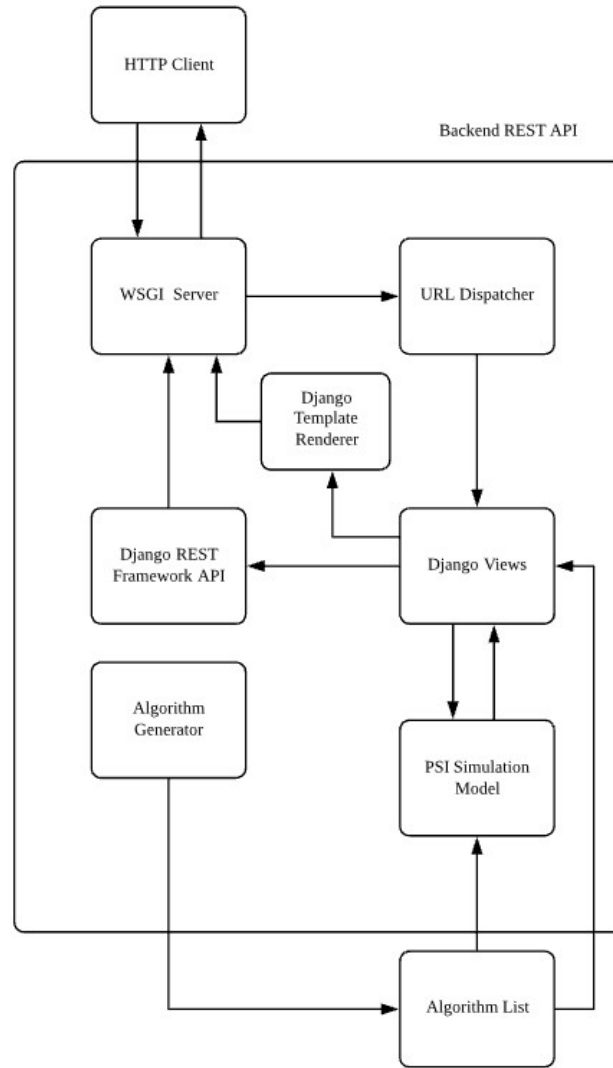


Figure 4. Backend REST API Software Architecture

Figure 4.1 serves as an illustration of the Backend Software (server) Architecture. Backend server generally works in the following, first, it receives a web request from a web client (from API calls, or web browsers). These requests are then received by the WSGI where it would then be forwarded towards the URL dispatcher, which examines the URL dispatched and calls in the requested function from the View module. The resulting function calls led to the View module returning values in 2 options. Depending on the input from web request, the View module can return values through the Django

REST Framework API (for API calls) or through a HTML Template (for web browsers). These responses are fed back into WSGI and then distributed to the requesting clients.

As mentioned previously, the main controller logic of this software is centered on the Views module. The Views module generally contains Python function with the following functionalities. The first functionality of the Views module is to read in PSI error input values (from client) and run the PSI simulation model based on the input values, and finally returning PSI phase errors based on the simulated model. The input format is in the form of a dictionary or JSON format with three key, value pairs. The keys are AWGN, Harmonics, and PSMC, while the values are in the format of only percentage error for AWGN, and an array or list of order and percentage error pairs for Harmonics and PSMC.

The second functionality for the Views module is to create a Latex representation for the PSI algorithms (to be rendered by the Frontend modules). This function is placed on the Backend since algorithm list is only available in Backend. In order to perform this functionality, therefore, the Django views module must fetch data from the Algorithm List, which are is placed in the software in the form of a static JSON file.

The static JSON file is generated by another Python module called Algorithm Generator. The Algorithm Generator generate a JSON file containing PSI Algorithms derived from the generic formula for PSI Algorithms. Equation 2.16, 2.17, 2.21, 2.22a, etc. serves as examples of generic formula for the PSI Algorithms. As different types of PSI algorithms have different generic formula, separate functions for different types of PSI Algorithms are created to produce a PSI Algorithm based on a frame number input. The output of these functions is shown in the form of a dictionary, containing the data required to construct PSI Algorithms. Table 9 seeks to further elaborate the contents of the dictionary output.

Key	Value
s	Coefficients of each intensity value on the numerator of a PSI Algorithm
c	Coefficients of each intensity value on the denominator of a PSI Algorithm
p	Phase shift (cumulative) amount
f	Frame number or number of intensity recordings

Table 9. Algorithm Data Structure

Another function on this module is also designed to generate PSI algorithms, label them, and collect them in the form of a static JSON file. Note that a static JSON file instead of a database is used in this project, this is because in general these PSI algorithms data are considered immutable, and therefore a database structure normally specialized for mutable data is not needed.

The final portion of the Backend software lies on the PSI Simulation Module. The PSI Simulation Module simulates different PSI algorithms when exposed to three different error representations. The method for simulation is through simulating the intensity values for each phase shift. Equation 2.24, 2.25, and 2.26 can be used as reference for the equations used to simulate PSI intensity values when exposed to the error representation AWGN, Harmonics, and PSMC. Equation 4.1, 4.2, and 4.3 demonstrates the equation used to perform PSI intensity value simulation. Note that for simplicity purposes, the variables I_{AC} , I_{DC} are set to be 0.5 (this does not matter as the values are cancelled out in the end).

$$I_i = 0.5 + 0.5 \cos(\varphi + \Delta\theta_i) + (n_{AWGN}/100)N(0,1) \quad (4.1)$$

$$I_i = 0.5 + 0.5 \cos(\varphi + \Delta\theta_i) + 0.5 \sum n_{harmonic_i} \cos(o_{harmonic_i} \varphi) \quad (4.2)$$

$$I_i = 0.5 + 0.5 \cos(\varphi + \Delta\theta_i + i \cdot n_{psmc_i} (PS)^{o_{psmc_i}}) \quad (4.3)$$

where i refers to frame number, φ refers to phase angle (simulated), $\Delta\theta_i$ refers to phase shift value (cumulative) on frame i , n refers to the error input (from the user), o refers to order or degree of the error input, $N(0, 1)$ refers to a randomized Gaussian distribution function, and PS refers to phase shift value $|\Delta\theta_i - \Delta\theta_{i-1}|$

Intensity values are simulated across phase value of range -3.14 to 3.13 with interval of 0.01. Afterwards intensity values calculated across all frames. Using the formula or data provided from Algorithm List, PSI algorithm can be simulated by multiplying each intensity values with the coefficient provided in the Algorithm List. Afterwards, numerators and denominators can be summed, and the summation values can be reverted to phase angle using the arctangent function.

Comparing the values of the calculated phase angles with the original phase angle (-3.14 to 3.13, with interval 0.01) and then calculating the standard deviation of the difference would then give us an RMS error between the original reference phase and the error-exposed phase angles. This RMS error are the output of the PSI simulation functions. In total 3 RMS phase angle errors from three different error representations will be obtained for each of the included PSI algorithms. A vector sum can be performed on these 3 outputs to obtain the resultant or “total” error.

In the end, a dictionary containing an entry for each PSI algorithms, with each entry a list consisting of the three error plus the resultant error will be returned to the web client. These data are the results of running the PSI simulation functions and should be the results that is demonstrated in the Graph and Table in the Frontend section.

Other than the main functions listed above, the backend server is also responsible other functionalities such as relaying information regarding the contents of algorithms (for

algorithms details section) or providing thresholding functionalities as requested by the frontend/client-side application.

4.3 Frontend

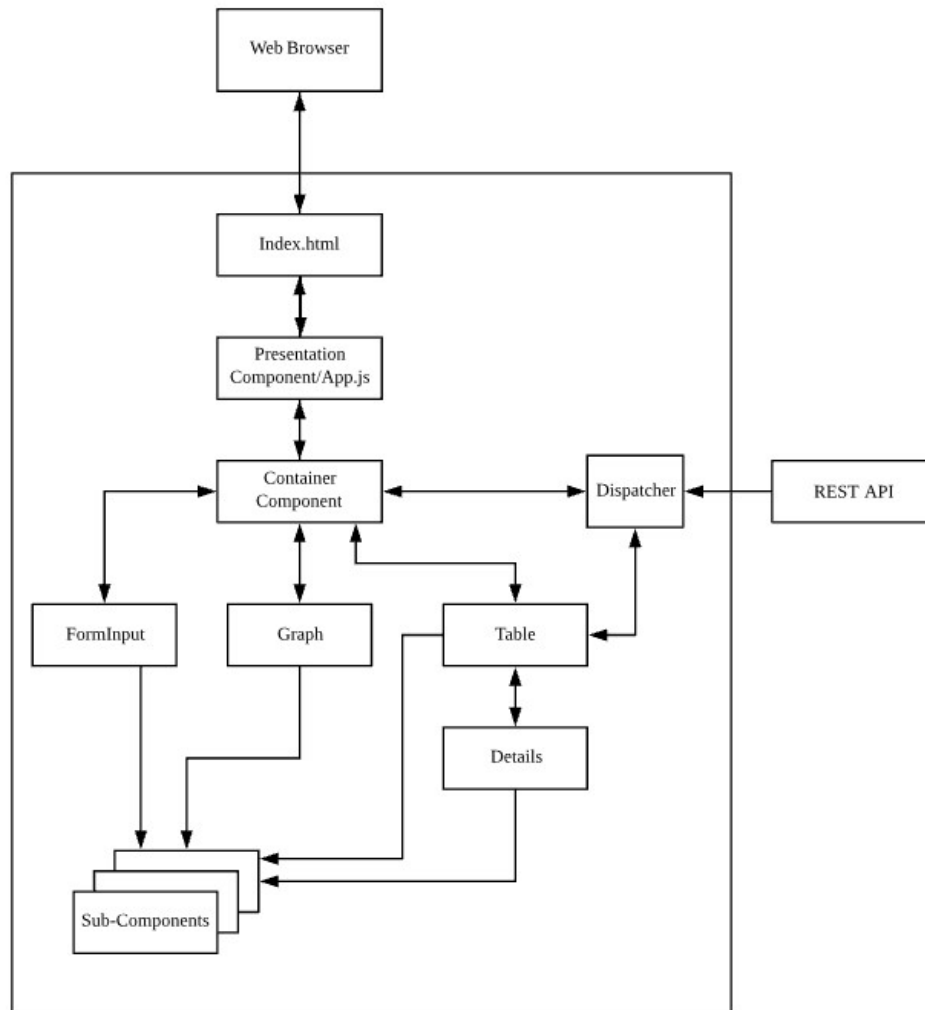


Figure 5. Frontend Software Architecture

Figure 4.2 serves as an illustration of the Frontend Software Architecture. The Frontend Application/Software is developed under the create-react-app webpack, which is one of the basic webpack for React Development. The components under this React app are displayed through the index.html web template. The presentational component App.js is used to shape index.html into the desired web app. In this project, the purpose of the presentational component App.js is to render the Container component, in other words, no state changes are performed within the Presentational component as the main functionalities are all facilitated under the Container Component. In React, individual components are re-rendered as changes on state or props are applied to them. Special

mechanisms for re-rendering also needs to be implemented especially in the wake of API calls which is normally performed asynchronously.

The container component houses the following components FormInput, Graph, and Table component. The Detail component, considered as a major component, is facilitated under Table component. The reason for such allocation is to ensure the Container component is not overwhelmed with all kinds of component interaction. The container component generally maintains state for the PSI error input and output, as well as passing props in the form of data or variable or function towards its child components (the major components).

The FormInput component, as the name suggests, functions as a dynamic form, which allow users to key in PSI error inputs. The main form generally consists of a single textbox for the percentage AWGN error, and a modifiable form for the Harmonics and PSMC inputs. Harmonics and PSMC input field in general consist of two textboxes for entering order and percentage error for the chosen order. The option to add or remove fields for Harmonics and PSMC is available such that users can input error designated in different orders. These fields are joined with an optional input field in the form of threshold which functions to filter out algorithms with error higher than the threshold input. The value contained on these fields are recorded in the form of component state variable. The state is updated every time values on one of the input fields are changed. An illustration of this component can be seen in Figure 6 below.

Input Error

AWGN (%)

Harmonic (%)

ADD ORDER

PSMC (%)

ADD ORDER

Order

Error



Order

Error



Order

Error



☐ Threshold (rad)

SUBMIT

Figure 6. FormInput Component

A submit button is included in the FormInput component to complete the input process. Upon pressing the submit button, the state variable value is transferred into the Parent or Container Component where using these values as an input, an API call for PSI simulation

can be performed. The output from this API call came in the form of “average” phase errors for many different PSI algorithms. These phase errors are then recorded or updated in a state variable in the Container Component. These state variables containing the phase error data can be passed down to other components that needed it.

The next major component is the Graph component. The Graph component for this project uses the Plotly charting library with a special emphasis taken onto 3D charts. Plotly is developed under d3 charting library and is currently one of the most widely used charting library for JavaScript. The Graph component obtains data from props variable passed down by the Container Component upon the completion of PSI simulation API call. From these raw data, several data grouping and processing are performed to fit in the phase error data on to the 3D scatter plot.

The 3D scatter plot in general have axis x, y, and z, with each of these axes represents phase error due to AWGN, Harmonics, and PSMC respectively. Each point or node in the scatter plot represents the phase errors for different PSI algorithms. Point labelling is performed through means of color coding as well as text labelling. Each family of algorithms are given a specific color scale to use on its members. A member with generally higher frame number will receive a darker color, while small frame numbered members will receive a lighter color in correspondence with the family’s color scale. Besides this, the other interactions in this 3D scatter plot is also available, allowing users to zoom, pan, and rotate graph. Further modifications are also added to the graph which were focused around adapting ranges, controlling axis, coloring, etc., all with the focus of improving design and interactivity of the graph. An illustration of the graph component can be seen in Figure 7 below.

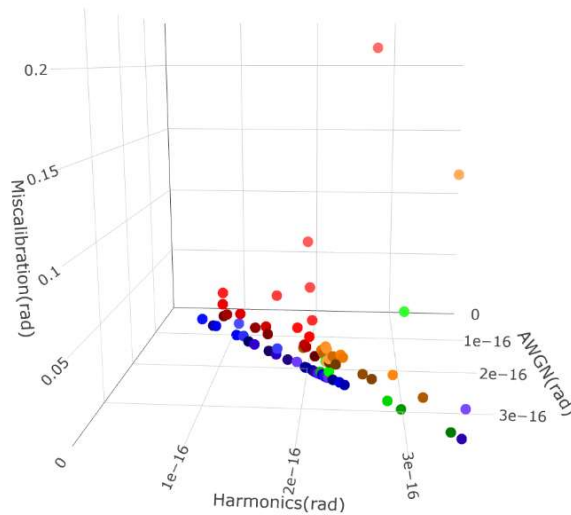
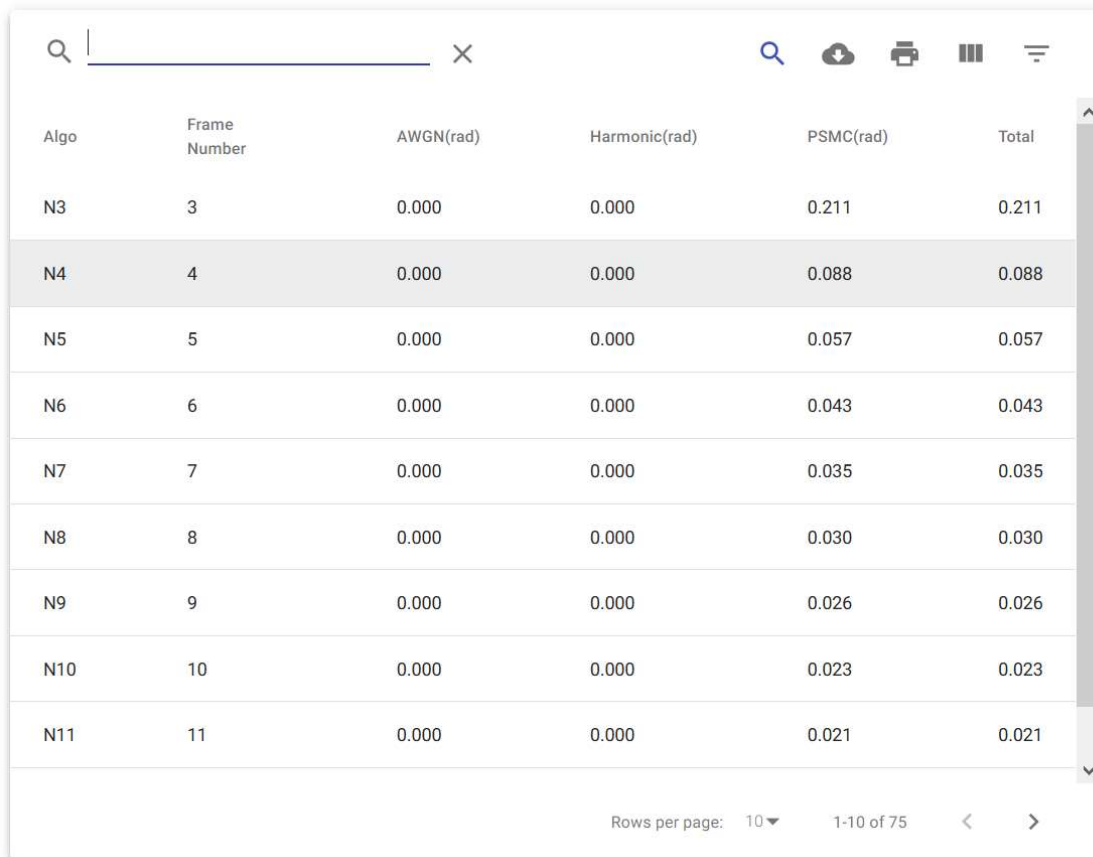


Figure 7. Graph Component

The Table Component for this project demonstrates a DataTable which receives PSI phase error data from the Container Component in the form of props variables. Data are displayed in the form of an interactive table with columns for PSI algorithm name/type, frame number, AWGN phase error, Harmonics phase error, PSMC phase error, and Total (or resultant error). Users can interact with this table through means such as searching, sorting (for all columns, ascending and descending order), clicking through rows for further information, etc. (other DataTable functions). An emphasis on clicking table rows led users into the Details component which were housed by the Table Component. The table component is illustrated in Figure 8 below.



Algo	Frame Number	AWGN(rad)	Harmonic(rad)	PSMC(rad)	Total
N3	3	0.000	0.000	0.211	0.211
N4	4	0.000	0.000	0.088	0.088
N5	5	0.000	0.000	0.057	0.057
N6	6	0.000	0.000	0.043	0.043
N7	7	0.000	0.000	0.035	0.035
N8	8	0.000	0.000	0.030	0.030
N9	9	0.000	0.000	0.026	0.026
N10	10	0.000	0.000	0.023	0.023
N11	11	0.000	0.000	0.021	0.021

Figure 8. Table Component

The Details component is a major component where information regarding chosen PSI algorithms can be seen. This generally includes the PSI algorithm equation, as well as the values of phase shift, plus other additional information related to a chosen algorithm. Information in the form of mathematical equation are further wrapped with a Latex formatting component, which allows for showing rendered mathematical equation. Selecting PSI algorithm can be performed through the interacting with the DataTable (clicking selected table row), as well as through selecting algorithm type and frame

number through a form. An illustration of the Details component can be seen in Figure 9 below.

Details

Algorithm Type: N-bucket

Frame Number: 3

Formula

$$\theta = \tan^{-1}\left(\frac{-0.87I_2 + 0.87I_3}{I_1 - 0.5I_2 - 0.5I_3}\right)$$

$$\Delta\theta = \frac{2\pi}{3}$$

Figure 9. Details Component

The Table component which houses the Details component maintains a state variable which corresponding to the current chosen or selected algorithm shown in the Details component. Upon selecting a different PSI algorithm (through table or form), this state variable will be updated, and the corresponding algorithm data will be loaded. Note that all information regarding algorithm data are located on the Backend server, therefore, an API call needs to be performed to fetch in rendered algorithm data every time a state is updated. The received data are passed down from the Table component into the Details component through the props variable.

Several other sub-components are also used in the development of this Frontend Software project. These components (such as the Latex renderer, 3D plot, menu bar, etc.) supports all the major components, enabling them to function as intended as well as providing further aesthetics towards those components. Aside from these React components, CSS modifications are also an integral part of this project, and therefore requires as much attention as well. These CSS modifications allows the components to be placed within the correct layout as well as providing design input towards the existing components. In conclusion, all these components work together in order to demonstrate the intended web view within the HTML template.

5. Conclusion

This project introduces a new and interactive method for analyzing and visualizing PSI algorithm errors. A web-app specialized for this mission is designed and developed with the aim to provide an interactive and intuitive 3-dimensional representation of PSI algorithm phase errors when exposed to 3 different error representation. Development of web-app consists of a free-to-use Backend REST API as well as a specially customized Frontend web application. Based on the visualization provided by this web-app, when exposed with a fixed amount of error across the three error representations, the most feasible yet optimal PSI algorithm can be filtered out and selected.

At the current state of the project, there is still a wide room for improvement. One of the main issues for improvement is with regards to the current design implementation. In the near future, perhaps a new and fresh design idea can be developed to help visualize data in a better and more interactive format. Other than that, more PSI algorithms can be added and integrated towards the design. With better design, perhaps more algorithms can be fit into the application without sacrificing user convenience.

Aside from the design point of view, in terms of simulation method, perhaps in the future, a much more efficient and robust PSI simulation method can be integrated as opposed to the traditional PSI simulation methods. This surely would help out not only in terms of development, but also in terms of scalability and maintainability of the project. Another benefit coming from a much more robust and efficient software came in the form of platform integrations. With a better system as well as a distinctive UI design, perhaps further integration with mobile devices apps or embedded systems could be possible. Integration with other platforms surely would greatly increase portability which led into a better implementation on the industrial basis.

References

- Bruning, J. H., Herriott, D. R., Gallagher, J. E., Rosenfeld, D. P., White, A. D., & Brangaccio, D. J. (1974). Digital Wavefront Measuring Interferometer for Testing Optical Surfaces and Lenses. *Applied Optics*, 13(11), 2693. doi: 10.1364/ao.13.002693
- Carré, P. (1966). Installation et utilisation du comparateur photoélectrique et interférentiel du Bureau International des Poids et Mesures. *Metrologia*, 2(1), 13–23. doi: 10.1088/0026-1394/2/1/005
- Creath, K. (1988). V Phase-Measurement Interferometry Techniques. *Progress in Optics*, 349–393. doi: 10.1016/s0079-6638(08)70178-1
- Fielding, R. T. (2000). Architectural Styles and the Design of Network-based ... Retrieved March 28, 2020, from https://www.researchgate.net/publication/216797523_Architectural_Styles_and_the_Design_of_Network-based_Software_Architectures
- Goodwin, E. P., & Wyant, J. C. (2006). *Field guide to interferometric optical testing*. Bellingham, WA: SPIE.
- Greivenkamp, J. E. (1984). Generalized Data Reduction For Heterodyne Interferometry. *Optical Engineering*, 23(4). doi: 10.1117/12.7973298
- Groot, P. D. (2011). Phase Shifting Interferometry. *Optical Measurement of Surface Topography*, 167–186. doi: 10.1007/978-3-642-12012-1_8
- Hariharan, P., Oreb, B. F., & Eiju, T. (1987). Digital phase-shifting interferometry: a simple error-compensating phase calculation algorithm. *Applied Optics*, 26(13), 2504. doi: 10.1364/ao.26.002504
- Hariharan, P. (2007). *Basics of interferometry*. Amsterdam: Elsevier.
- Hibino, K., Oreb, B. F., Farrant, D. I., & Larkin, K. G. (1997). Phase-shifting algorithms for nonlinear and spatially nonuniform phase shifts. *Journal of the Optical Society of America A*, 14(4), 918. doi: 10.1364/josaa.14.000918
- Hibino, K., Larkin, K. G., Oreb, B. F., & Farrant, D. I. (1998). Phase-shifting algorithms for nonlinear and spatially nonuniform phase shifts: reply to comment. *Journal of the Optical Society of America A*, 15(5), 1234. doi: 10.1364/josaa.15.001234
- Kemao, Q., Fangjun, S., & Xiaoping, W. (2000). Determination of the best phase step of the Carré algorithm in phase shifting interferometry. *Measurement Science and Technology*, 11(8), 1220–1223. doi: 10.1088/0957-0233/11/8/316
- Larkin, K. G., & Oreb, B. F. (1992). Design and assessment of symmetrical phase-shifting algorithms. *Journal of the Optical Society of America A*, 9(10), 1740. doi: 10.1364/josaa.9.001740
- Malacara, D. (2007). *Optical shop testing*. Hoboken, NJ: Wiley-Interscience.
- Schmit, J., & Creath, K. (1995). Extended averaging technique for derivation of error-compensating algorithms in phase-shifting interferometry. *Applied Optics*, 34(19), 3610. doi: 10.1364/ao.34.003610
- Schmit, J., & Creath, K. (1996). Window function influence on phase error in phase-shifting algorithms. *Applied Optics*, 35(28), 5642. doi: 10.1364/ao.35.005642
- Schreiber, H., & Bruning, J. H. (2007). Phase Shifting Interferometry. *Optical Shop Testing*, 547–666. doi: 10.1002/9780470135976.ch14

- Servin, M., Estrada, J. C., Quiroga, J. A., Mosiño, J. F., & Cywiak, M. (2009). Noise in phase shifting interferometry. *Optics express*, 17(11), 8789-8794.
- Surrel, Y. (1996). Design of algorithms for phase measurements by the use of phase stepping. *Applied Optics*, 35(1), 51. doi: 10.1364/ao.35.000051
- Surrel, Y. (1998). Phase-shifting algorithms for nonlinear and spatially nonuniform phase shifts: comment. *Journal of the Optical Society of America A*, 15(5), 1227. doi: 10.1364/josaa.15.001227
- What is an Interferometer? (n.d.). Retrieved from <https://www.ligo.caltech.edu/page/what-is-interferometer>