TK
8315
Y36
2008

Development and Implementation of the Linear Phase Algorithm in a Two-Dimensional Sun-Sensor

by

Albert Kar-Kei Yam,

Bachelor of Engineering, Aerospace Engineering,

Ryerson University, 2006

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Applied Science

in the Program of

Aerospace Engineering

Toronto, Ontario, Canada, 2008

©Albert Kar-Kei Yam 2008

PROPERTY OF
RYERSON UNIVERSITY LIBRARY

Author's Declaration

I hereby declare that I am the sole author of this thesis or dissertation.
I authorize Ryerson University to lend this thesis or dissertation to other
institutions or individuals for the purpose of scholarly research.

Albert Yam

I further authorize Ryerson University to reproduce this thesis or dissertation
by photocopying or by other means, in total or in part, at the request of other
institutions or individuals for the purpose of scholarly research.

Albert Yam

Development and Implementation of the Linear Phase Algorithm in a Two-Dimensional Sun-sensor

Albert Kar-Kei Yam

Master of Applied Science Aerospace Engineering

Ryerson University 2008

Abstract

The sub-pixel peak position estimation performance of the Linear Phase algorithm was enhanced and extended for use with a two-dimensional sensor. The two-dimensional system is composed of orthogonal pairs of slits diffracting light onto a single pixel array. The adaptation of the Linear Phase algorithm into the two-dimensional system was performed by manipulating the sensor image into two one-dimensional images. Performance testing used simulated sensor output and images attained from a real sun-sensor. The Linear Phase algorithm was then implemented into an embedded system to simulate on-board processing. A C8051 microcontroller was used as the embedded microcontroller. Results from the embedded processing were compared against the native Matlab implementation for consistency in sub-pixel peak position estimation performance. The Linear Phase algorithm was able to perform with excellent results, comparable to current two-dimensional sun-sensor algorithms.

Acknowledgments

I would like to thank my thesis supervisor Dr. John Enright for allowing me the opportunity to pursue a Masters Degree. His direction and guidance allowed me to focus on the goal without straying too far, and his advice and insight provided me with the resources to be able to pursue my work.

My family has been a significant source of support. Giving me the strength and inspiration to pursue my goals. My parents, Vince and Flora Yam, have brought me to where I am. Never pressuring me and allowing me to be myself. Thank you so much.

I would also like to thank my lab mates. The friendly environment in the lab made the long hours staring at a computer easier and a good stress reliever.

Last, but not least I would like to thank Zilly for being the one thought in my head that keeps me going. Thank you for your love and support, being at my side, and waiting for me to finish.

Contents

Declaration	iii
Abstract	v
Acknowledgments	vii
1 Introduction	1
1.1 Objective	2
1.2 Outline	3
2 Background	5
2.1 The Discrete Fourier Transform	5
2.2 Phase Correlation Method	8
2.3 The Linear Phase Algorithm	9
2.4 Summary	14
3 One-Dimensional Linear Phase Algorithm Tests	17
3.1 Simulated Images for One-Dimensional Sensors	17
3.2 Effect of Large Delays	21
3.3 Phase Unwrapping	22
3.3.1 Phase Reconstruction using Linear Assumptions	23
3.3.2 Phase Reconstruction using Geometric Slope Bounds .	24
3.3.3 Phase Reconstruction using Algebraic Simplifications .	26

3.3.4	Comparison of Unwrapping Methods	27
3.4	Effect of Shortening the Image	28
3.5	Summary	30
4	Two-Dimensional Algorithm Testing and Simulation	31
4.1	Simulation of Images from a Two-Dimensional Sun-Sensor . . .	32
4.2	Types of Images from a Two-Dimensional Sun-Sensor	33
4.3	Two-Dimensional Shift Estimation Algorithms	36
4.3.1	Case Determination	36
4.3.2	Cross Correlation	40
4.3.3	LP Algorithm	41
4.4	Decomposition of an Image from a Two-Dimensional Sun-Sensor	43
4.4.1	Peak Identification	43
4.4.2	Peak to Slit Matching	44
4.4.3	Separating the Image	44
4.5	Capturing Images from a Two-Dimensional Sun-Sensor	45
4.5.1	Results	47
4.6	Gap Correction and Non-Uniformity	48
4.6.1	Gap Correction	48
4.6.2	Non-Uniformity	49
4.6.3	Interfering Cases	51
4.6.4	Results	51
4.7	Conclusions	52
5	Embedded Implementation	55
5.1	Migration Approach	55
5.2	The C8051 Microcontroller	56
5.3	DFT	57
5.3.1	Butterfly FFT	57
5.3.2	Radix-2 FFT	58
5.3.3	Goertzel Method	58

5.4	Algorithmic Memory Optimizations	59
5.4.1	Pre-computing Constant Arrays	60
5.4.2	Polynomial Fitting	61
5.4.3	Slope Unwrap	62
5.4.4	Gap Correction and Non-Uniformity	62
5.5	Manual Optimizations	63
5.6	Final Results	65
5.7	Implementation Conclusions	66
6	Conclusions	69
6.1	Summary	69
6.1.1	One-Dimensional Linear Phase Algorithm Testing . . .	69
6.1.2	Two-Dimensional Algorithm Testing	70
6.1.3	Embedded Implementation	71
6.2	Future work	72
	Appendix	73
A	Additional Topics	73
A.1	LP width test	73
A.2	LP window length test	73
A.3	SS-411 Pixel Array	74
A.4	Non-Linear Least Squares	74
	Bibliography	77

List of Tables

4.1	SS-411 sun-sensor parameters.	46
4.2	Mean Error over the FOV of the same image set.	52
5.1	Memory Restrictions from the 8051 development board.	57
5.2	Final memory usage.	66

List of Figures

2.1	The signal flow diagram of a two point DFT, using the decimation in time FFT.	7
2.2	Orientation of the sensor array in the detector plane.	11
2.3	Example of a wrapped and unwrapped function.	13
2.4	Noise corrupting extremities of the Signal Phase.	14
3.1	One-dimensional sun-sensor system.	19
3.2	Shift size to wrapping correlation.	22
3.3	Reference Slope Error.	24
3.4	Example of Geometric Bounds used for unwrapping.	25
3.5	Error with shifts spanning the whole pixel array.	28
3.6	Linear section shortened due to a shorter image used.	29
4.1	Mask to pixel array arrangement of a two-dimensional sun-sensor.	32
4.2	Sensor Images of various cases.	35
4.3	FOV plot of problematic case locations.	36
4.4	Two-dimensional sensor images.	38
4.5	Cross-Correlation Signal Example.	41
4.6	Problematic Case, equi-distance.	42
4.7	Image separation flow chart.	43
4.8	Identified Peaks and Valleys for sample images.	44
4.9	Segmenting an image into constituent axes.	45

4.10	Visual schematic of SAIL facility.	46
4.11	Initial results of Matlab LP algorithm.	47
4.12	Schematic View of the Pixel Dies.	49
4.13	Non-uniform pixels near the die gap.	50
4.14	Final Matlab LP algorithm results.	52
5.1	Migration approach flow chart.	56
5.2	Final results of the C code LP algorithm.	66
A.1	LP width test.	74
A.2	LP window length error.	75
A.3	Close-up of the Pixel Array of the SS-411 sun-sensor.	75
A.4	NLSQ results.	76

Nomenclature

Symbol	Definition
$I[n]$	Sensor Image
$x[n]$	Index points of image values
α	Index displacement
τ	Linear displacement
ΔT	Pixel spacing
h	Height of the optics
ψ	Angle between x-axis and pixel array
$\psi[n]$	Phase angle points
L_x, L_y	Half-width between slits (normalized)

Chapter 1

Introduction

Attitude determination is one of the most essential components of spacecraft navigation. Since the sun is the brightest object around Earth, it can be used as a reference point. Sun-sensors are one of the most popular methods to determine a spacecraft's attitude. The basic goal of a sun-sensor is to determine the direction of the sun. Sun-sensors are widely used for several reasons, they are low cost, have low power requirements and do not add a lot of mass to a spacecraft. Many different classes of spacecraft employ sun-sensors due to these factors. Since not all spacecraft are sun-synchronous, multiple sun-sensors are sometimes used.

Sun sensors require two main components, a mask and a detector. The mask is used to shape the incoming light, and to keep the detector from saturating. There are two main types of sun-sensors, analog sun-sensors, and digital sun-sensors.

Analog sun-sensors use a small number of photo-sensors as the detector. Each photo-sensor produces a variable output. A function of the output from these photo-sensors are used to determine the sun angles. The main drawback of analog sun-sensors is the possibility of significant error due to other objects in the field of view (FOV). Objects such as the Earth and the moon can affect the precision up to 20° [Appel, 2003].

1. INTRODUCTION

Digital sun-sensors use a large number of pixel sensors as the detector. The pixels are small such that, through the mask, each pixel is related to an angular position of the sun. Each pixel produces an output relative the amount of light that strikes their respective surface areas. Due to the diffraction through the mask, many of the pixels in the array will produce an output. The collection of these outputs is defined as an image. The simplest way to find the sun vector is to find the brightest pixel in the image. The precision of this type of digital sun-sensor would be directly related to the size of the pixels.

Many digital sun-sensors also have a processor. The reason for this is to employ algorithms to process the image received from the detector. Smarter processing routines allow more information to be extracted from the same image. These processing routines allow the sun-sensor to determine the sun vector to sub pixel accuracy. Sub pixel accuracy is when an algorithm is able to determine the peak of the image to within a pixel, rather than at the center of a pixel. The algorithms used are generally peak position estimators, that use more information from the image than just the brightest pixel. The Linear Phase (LP) algorithm is a frequency domain algorithm for peak position estimation.

1.1 Objective

The LP algorithm has been shown to work in the one-dimensional system, with good results [Godard, 2004]. Expanding the LP algorithm into a two-dimensional system is the next step. The objective of this research is to adapt the LP algorithm into the two-dimensional system, and implement the algorithm onto an onboard microprocessor. To develop the LP algorithm into the two-dimensional system, simulated images are used. This adaptation of the algorithm is performed within the Matlab environment. The Matlab environment is ideal to analyze and test the different functions within the

LP algorithm. Once the LP algorithm is adapted to the two-dimensional system, the algorithm is validated using real image data. The real image data is captured from a sun-sensor in a laboratory setup. Having validated the LP algorithm using real image data, testing the LP algorithm outside of the Matlab simulation environment was next. To replicate the microprocessor in a sun sensor, the LP algorithm is then migrated into a form able to be embedded for implementation. This means that the LP algorithm is to be translated into C code. This C code is to be embedded on a microprocessor to simulate processing on a sun sensor. Verifying the results of the this on board processing is performed by comparing to the established Matlab version. Through these steps, the LP algorithm is adapted into a two-dimensional system, and implemented onto a microprocessor.

1.2 Outline

The background required for the development of the LP algorithm is discussed in Chapter 2. Chapter 3 presents the tests that were performed using a simulated one-dimensional sun-sensor. The adaptation of the LP algorithm to a two-dimensional system is detailed in Chapter 4. Additional algorithms that were used are mentioned, as well as the developmental process of the LP algorithm. The conversion of the LP algorithm to an embedded form is discussed in Chapter 5. The problems that were faced and methods used to overcome the problems are also discussed. Chapter 6 includes the conclusion of the development process. Suggestions on future research are also discussed.

Chapter 2

Background

This section describes the theoretical background, and mathematics behind the Linear Phase (LP) algorithm.

2.1 The Discrete Fourier Transform

The Discrete Fourier Transform (DFT) is a Digital Signal Processing (DSP) technique to analyze a signal in the frequency domain [Oppenheim and Schafer, 1975]. The DFT of the discrete signal $x[n]$ is defined as:

$$X[k] = \sum x[n]e^{-j\frac{2\pi kn}{N}} \quad \text{for } k = 0, 1, 2, \dots, N-1 \quad (2.1)$$

with the inverse function:

$$x[n] = \frac{1}{N} \sum X[k]e^{j\frac{2\pi kn}{N}} \quad \text{for } n = 0, 1, 2, \dots, N-1 \quad (2.2)$$

Where N is the number of elements in the sequences n and k .

The DFT relates a discrete signal in the time domain to a signal in the frequency domain. The image captured by a linear pixel array is a sampled version of the real projected image. In this case, the DFT relates spatial brightness variations to spatial frequencies. Since the pixel array is only able

2. BACKGROUND

to report intensity values, only DFTs of real signals are studied.

Symmetry Properties

DFTs of real valued signals have interesting symmetry properties. The DFT of a real valued sequence is a conjugate symmetric sequence:

$$\text{for } x[n] \in \mathbb{R}, \quad X[k] = X_{[N-k]}^* \quad (2.3)$$

Where $()^*$ denotes the complex conjugate. This means that the real values are symmetric about the center point and the imaginary values are anti-symmetric about the center point. For clarity, an example of a symmetric sequence is the *cosine* function, while the *sine* function is an example of an anti-symmetric sequence.

The Fast Fourier Transform

The Fast Fourier Transform (FFT) is a simplified form of a DFT. The FFT uses the symmetry properties to implement the DFT using less calculations. There are many types of FFTs. In general FFTs perform best with sequence lengths of $N = 2^n$, although with zero padding, any length sequence may be computed using FFTs to reduce the number of calculations required for a DFT [Mitra, 2006].

An example would be the Decimation-in-Time FFT. For a two point DFT, the decimation-in-time signal flow diagram looks like Figure 2.1, which refers to:

$$X[0] = x[0] + x[1] \quad (2.4)$$

$$X[1] = x[0] - x[1] \quad (2.5)$$

Where, $x[0]$ and $x[1]$ are the two input signals, and $X[0]$ and $X[1]$ are

Figure 2.1. *The signal flow diagram of a two point DFT, using the decimation in time FFT.*

the two corresponding DFTs. The odd indexed input signal, $x[1]$ would be multiplied by a twiddle factor. The general twiddle factor is:

$$W_n^r = e^{-j(\frac{2\pi}{N})r} \quad (2.6)$$

$$= \cos(\frac{2\pi}{N}r) - j \sin(\frac{2\pi}{N}r) \quad (2.7)$$

The twiddle factors can be simplified based on even or odd indices. For the two point signal, the twiddle factors simplify to 1 and -1 . Using these twiddle factors and the above signal flow diagram, any sequence length, 2^n , will have a DFT that can be expressed in a signal flow diagram in a similar fashion. The decimation-in-time FFT requires $\frac{N^2}{2}$ calculations. For a DFT using equation 2.1, the number of calculations required is N^2 , a saving of half the calculations. Other FFTs require even less calculations.

Time-Shifting Theorem

Another relevant property of the DFT is the Time-Shift Theorem (also referred to as the Frequency-Shift Theorem [Mitra, 2006]), that states that if,

$$x[n] \leftrightarrow X[k] \quad (2.8)$$

2. BACKGROUND

then:

$$x[n] \cdot e^{\frac{-j2\pi nm}{N}} \longleftrightarrow X[k - m] \quad (2.9)$$

and,

$$x[n - m] \longleftrightarrow X[k] \cdot e^{\frac{-j2\pi km}{N}} \quad (2.10)$$

This property relates a shifted signal, $x[n - m]$, to a unshifted reference signal, $x[n]$. The time-shifting theorem is interesting because it holds for a delay, m , that is not an integer.

2.2 Phase Correlation Method

The phase correlation method is based on the Fourier shift property, which states that shift in the coordinate frames of two functions is transformed in the Fourier domain as linear phase differences [Knapp and Carter, 1976]. For a two-dimensional case, this can be described as follows:

Let $f_1(x, y)$ and $f_2(x, y)$ be two functions such that,

$$f_2(x, y) = f_1(x - x_0, y - y_0) \quad (2.11)$$

The Fourier shift property states that,

$$\hat{f}_2(u, v) = \hat{f}_1(u, v) e^{-j(u x_0 + v y_0)} \quad (2.12)$$

Hence the normalized cross power spectrum is given by:

$$\frac{\hat{f}_2(u, v) \hat{f}_1^*(u, v)}{|\hat{f}_1(u, v) \hat{f}_1^*(u, v)|} = e^{-j(u x_0 + v y_0)} \quad (2.13)$$

Where $*$ indicates the complex conjugate, (u, v) are coordinates in the Fourier domain, and (x_0, y_0) are the horizontal and vertical shifts occurring

between f_1 and f_2 . The inverse Fourier transform of the normalized cross power spectrum (2.13) would yield the result of $\delta(x - x_0, y - y_0)$ which is a Dirac delta function centered at (x_0, y_0) . Replacing the Fourier transform by a DFT and the Dirac delta function by a unit impulse, the results still hold and work remarkable well [Foroosh and Zerubia, 2002].

The phase correlation method is able to estimate sub-pixel shifts, with high accuracy. This method served as a precursor to the development of the Linear Phase Algorithm.

2.3 The Linear Phase Algorithm

The LP algorithm was developed using the above DSP techniques and phase correlation methodology. The LP algorithm works by first finding DFT of the unshifted reference image. The reference image is when the sun is normal to the sun-sensor, defining the zero delay signal. Taking, $I_0[n]$ as the reference image, the DFT is $S_0[k]$:

$$I_0[n] \longleftrightarrow S_0[k] \quad (2.14)$$

Since the mask of the sun-sensor ensures that the image pattern is similar for the entire FOV, off-axis illuminations are assumed to be shifted versions of normal incidence. The time-shifting theorem is used to show the DFT of the shifted image, shifted by α , to an expected DFT form of:

$$I[n] = I_0[n - \alpha] \longleftrightarrow e^{-j\frac{2\pi k\alpha}{N}} S_0[k] \quad (2.15)$$

As mentioned, the shift does not have to be an integer value, this allows for the sub-pixel accuracy. The exponent of the Fourier series coefficients of the DFT of the shifted image contains the shift. To isolate the Fourier series coefficient, the DFT of the shifted image is divided by the DFT of the unshifted image:

2. BACKGROUND

$$Y[k] = \frac{e^{-\frac{j2\pi k\alpha}{N}} S_0[k]}{S_0[k]} = e^{-\frac{j2\pi k\alpha}{N}} \quad (2.16)$$

The phase angle of this Fourier coefficient is what is sought (Equation 2.17). This step is conceptually simple, but not computationally. The DFT of the reference image contains zero points, which naturally cause distortion when evaluating Equation 2.16. A threshold is used to avoid the indices where the zero points occur, without an effect on the precision of the LP algorithm:

$$\psi[k] = \angle Y[k] = -\frac{2\pi k\alpha}{N} \quad (2.17)$$

The estimate of delay on the array τ is directly related to the signal phase, α , by the pixel width ΔT . The slope is a linear phase system, thus α can be approximated using a least square fit. The DFTs are complex numbers in exponential form, so finding the angle must be carefully performed due to wrapping. Furthermore, noise can have a large effect on the signal phase. Once α is approximated, the delay, τ , is then calculated using the relation:

$$\tau = \alpha \cdot \Delta T \quad (2.18)$$

This is the delay of the peak position from the peak's reference, zero delay position.

Sun Vector

To calculate the sun vector, a second shift in the orthogonal axis is required. Once the shifts in the orthogonal directions are known, τ_x and τ_y , the sun vector is calculated using temporary quantities, A_m and B_m , defined as:

$$A_m = [\Delta T \cdot \cos \psi \cdot \tau_x + \rho_x] \quad (2.19)$$

Figure 2.2. Orientation of the sensor array in the detector plane. X and Y are parallel to the slits, Z is normal to the pixel array. ρ_x and ρ_y denote the center of pixel-0 on the array. The pixel array is at an angle, ψ , from the X axis.

$$B_m = [\Delta T \cdot \sin \psi \cdot \tau_y + \rho_y] \quad (2.20)$$

Where the pixel spacing, ΔT , the location of pixel zero, ρ_x and ρ_y , and the angle between the x-axis and the sensor array, ψ , are physical parameters of the sun sensor. Figure 2.2 details the parameters on the sensor array. With A_m and B_m , the sun-vector can be calculated with h , the height of the optics, by:

$$s_x = -\frac{A_m}{h} s_z \quad (2.21)$$

$$s_y = -\frac{B_m}{h} s_z \quad (2.22)$$

$$s_z = -\sqrt{\frac{h^2}{A_m^2 + B_m^2 + h^2}} \quad (2.23)$$

This derivation to find the sun vector from the estimated peak positions is based on the geometry model of the SS-330 sun-sensor [Enright et al., 2008]

2. BACKGROUND

Phase Unwrapping

Phase Unwrapping, referred to as wrapping is a mathematical complication that is inherent in the frequency domain. Mathematical calculation of angles leads to wrapping. The wrapping complicates the LP algorithm when the slope of the signal phase is measured. The formulation of the DFT provides the phase angle in complex exponential form:

$$z = Ae^{j\theta} \quad (2.24)$$

In polar form this translates to:

$$Ae^{j\theta} = A(\cos(\theta) + j \sin(\theta)) \quad (2.25)$$

Finding the phase angle:

$$\theta = \text{atan2}\left(\frac{\sin(\theta)}{A}, \frac{\cos(\theta)}{A}\right) \quad (2.26)$$

Equation 2.26 will only give results between $\pm\pi$. The bounded results cause the problem of wrapping. This is justified by Euler's Identity that says:

$$e^{j2\pi} = 1 \quad (2.27)$$

Figure 2.3 is a simple example of a wrapped sequence and its unwrapped counterpart.

Relating this to the LP algorithm, is the calculation of $\psi[k]$. As calculated, $\psi[k]$, is correct modulo 2π , but in order to fit a line to $\psi[k]$, the phase function must be unwrapped beyond $\pm\pi$ in order to restore a straight line.

Slope Measurement

The purpose of the unwrapping is to measure the slope of the signal phase. Once the points of the signal phase are unwrapped, the slope can be estimated. The simplest way is to fit a line to the points. This would be done by a

2.3. The Linear Phase Algorithm

Figure 2.3. Example of a wrapped and unwrapped function. The phase angle points (ψ) relating to the points (k) of the angle of the phase term, $-\frac{2\pi k\alpha}{N}$.

linear least squares approximation. As mentioned, due to the anti-symmetric nature of the signal phase, one is able to assume the y -axis intercept is zero, and with the known points in the sequence as $x[k]$ and the corresponding signal phase angles as $\psi[k]$, the slope can be averaging the slopes:

$$Slope = \frac{\sum_{k=1}^N \frac{\psi[k]}{x[k]}}{N} \quad (2.28)$$

This research performed several optimizations to the slope measurement, this can be seen in Section 5.4.2.

Noise Effects

Noise has an underlying effect that acts upon the LP algorithm. The LP algorithm expects that equation 2.15 holds true for the comparison in equation 2.16 to isolate a linear phase term. However, the signal phase is not linear when noise is present in the system. Only the center section of the signal phase is usable. This can be seen in Figure 2.4. The reason that only the

2. BACKGROUND

Figure 2.4. *Noise corrupting extremities of the Signal Phase. The phase term of a zero noise image, and the same image with random noise of 2% of max intensity.*

center section is usable is due to the comparison in Equation 2.16. The division by $S_o[k]$ is such that the noise is amplified, except for the center section, where the values of $S_o[k]$ are high. The length of the center section is not directly related to the noise, however, the noise affects the higher frequencies of the signal phase. Within the linear section spikes of noise do appear; however, the unwrapping method removes these points from corrupting the results.

2.4 Summary

This chapter examined position estimation using the Linear Phase algorithm and the related Phase Correlation Method. The most prominent part of the LP algorithm is the DFT of the signal images. Basics of the DFT were reviewed along with some helpful properties and theorems. A simple geometric model of the sun sensor optics allows calculation of the sun vectors from the peak position estimates. Several important implementation challenges

2.4. Summary

were also discussed, including phase unwrapping, slope measurement, and the effects of noise on the LP algorithm.

Chapter 3

One-Dimensional Linear Phase Algorithm Tests

The LP algorithm is proven to work well for a one-dimensional sun-sensor system [Godard, 2004]. To further the development and implementation of the algorithm, a look at the limitations and adaptability was the next step taken. Looking at the extreme cases of a sun-sensor on the LP algorithm served as a robustness test as well as a stepping stone to be used for later development of the algorithm. The tests were performed in a simulated environment using Matlab. The images were simulated based on the specifications of a *SS-330* sun-sensor [Enright and Godard, 2006].

3.1 Simulated Images for One-Dimensional Sensors

Sun-sensors use the sun as a reference point to study the geometry of the system starting from the sun is logical. Due to the distance from the sun, the light emitted can be approximated as plane waves. These plane waves hit the aperture mask of the sun-sensor. The mask has a known slit width,

3. ONE-DIMENSIONAL LINEAR PHASE ALGORITHM TESTS

the direction of the slit is also a reference direction. The diffraction of the plane waves through the aperture mask produces a specific intensity curve on the pixel array. This intensity curve is specific due to the designed width of the slit [Enright and Godard, 2006].

The pixel array is arranged in such a way that each pixel directly relates to a certain sun angle region in the FOV. The pixel response is proportional to the amount of photons that strike the surface area of the pixel. This amount of photons is directly related to intensity of the light. The diffracted pattern is sampled by the pixels, an estimation of the intensity curve. This means that the actual intensity curve produced is not directly known. The combination of the readings from the pixel array is the *image* produced by the sensor.

Figure 3.1 demonstrates this process. The thickness of the optics, z_s , is the distance between the aperture mask and the pixel array, a neutral density filter. The slit half-width is denoted by, w . The sun-angle sought in this process is denoted by, θ_{sun} . The estimate produced is based upon the linear displacement, τ , also referred to as the *shift*. The more precise an algorithm is able to determine the peak shift, τ , the more precise the sun-angle is able to be predicted.

One-dimensional images are simulated in the Matlab environment using this geometry. The parameters used are based upon the SS-330 sun-sensor from Sinclair Interplanetary. The analytical sensor model, uses the Fresnel solution to diffraction through a slit [Godard, 2004]. The continuous intensity at the pixel plane is given by:

$$I_c(X) = \frac{1}{2} \{ [C(\alpha_2) - C(\alpha_1)]^2 + [S(\alpha_2) - S(\alpha_1)]^2 \} \quad (3.1)$$

Where $C(\alpha)$ and $S(\alpha)$ are the Fresnel Cosine and Fresnel Sine integrals of:

$$\alpha_1 = -\sqrt{2}(\sqrt{N_F} + X) \quad (3.2)$$

3.1. Simulated Images for One-Dimensional Sensors

Figure 3.1. One-dimensional sun-sensor system. The aperture mask has a half slit width of w , and is z_s away from the pixel array. The shift, τ , in the sensor image relates to the sun angle, θ_{sun} .

3. ONE-DIMENSIONAL LINEAR PHASE ALGORITHM TESTS

$$\alpha_2 = \sqrt{2}(\sqrt{N_F} - X) \quad (3.3)$$

The Fresnel Number, N_F , is calculated using the slit width, w , index of refraction of the optics, n_{glass} , filter thickness, z_s , and free-space wavelength λ_0 :

$$N_F = \frac{w^2 n_{glass}}{\lambda_0 z_s} \quad (3.4)$$

The actual distance, x , is used to calculate the non-dimensional distance, X by:

$$X = \frac{x}{\sqrt{\lambda z_s}} \quad (3.5)$$

Using the measurements of the SS-330 sun-sensor, the limits of the pixel array can be approximated to $|X_{max}| \simeq 250$. This maximum is used to calculate a non-dimentionalized pixel spacing, ΔT , knowing the amount of pixels, N :

$$\Delta T = \frac{2X_{max}}{N} \quad (3.6)$$

The pixel spacing is used to discretize the continuous intensity image. This discretization would be similar to the sensor image produced by a one-dimensional sun-sensor:

$$I[n] = I_c(n\Delta T) \quad (3.7)$$

This is a one peak pattern. In order to extent the equation to a two peak pattern, equation 3.7 is extended to two diffraction slits. For the reference image, the peaks are shifted from the origin by the normalized half-slit width, L :

$$I[n] = I_c(n\Delta T + L) + I_c(n\Delta T - L) \quad (3.8)$$

Equation 3.8 is used for the reference image that is required for the LP algorithm. Shifting the illumination pattern by τ and adding noise, n_σ , to the system is important for simulation purposes. This results in the a simulated image in the form of:

$$I[n] = I_c(n\Delta T + L - \tau) + I_c(n\Delta T - L - \tau) + n_\sigma[n] \quad (3.9)$$

This image is a two peak pattern that is able to be shifted along the pixel array to simulate the field of view of a sun-sensor. Using these images the effect of large delays and windowing was studied.

3.2 Effect of Large Delays

Early tests considered only small magnitude delays [Enright and Godard, 2006]. This test studied the effects shifts that span the whole sensor array on the LP algorithm. The effects on the peak estimation performance of the LP algorithm are the main focus of this test.

This study used the whole pixel array to observe what happens with the slope of the signal phase as the shifted peaks are farther from the center, near the edges of the FOV. This study found that the signal phase becomes very discontinuous and wraps more frequently for higher delays. This is a problem for the LP algorithm because the value sought, α (equation 2.17), is the slope of the phase angle points. Figure 3.2 (left Figure) is an example of a small delay close to the center of the pixel array, where the original LP unwrapping method works well. The problem with this unwrapping method occurs at large delays when the discontinuities occur much more frequently. Looking at Figure 3.2 (right Figure), the points on the signal phase are seen to orient in a way where unwrapping is difficult. The reason that unwrapping is difficult is because of the short linear sections. At the extreme, the linear sections are only two points long, this causes the unwrapping to default to the sign of the reference point.

3. ONE-DIMENSIONAL LINEAR PHASE ALGORITHM TESTS

Figure 3.2. *Small shift causes a slightly wrapped Phase Angle term (left). Large shift causes a highly wrapped Phase Angle term (right).*

To solve the unwrapping problem at large delays several different unwrapping methods were proposed and developed.

3.3 Phase Unwrapping

Slope unwrapping deals with unwrapping the angle of the phase term, $e^{-j\omega n_0}$, resulting from the comparison of the shifted image to the unshifted image. The signal phase is bounded by $\pm\pi$ depending on the delay of the signal. The model, being simulated in Matlab, used the predefined `unwrap` function as an initial attempt. However, this function was unable to unwrap all of the points on the phase term when shifts became large. The native `unwrap` function requires that every point in the phase term be defined, which is not the case in this implementation, resulting in the inability to correctly unwrap the phase angle. Not being able to unwrap completely causes the line fit to give an incorrect slope.

3.3.1 Phase Reconstruction using Linear Assumptions

The linear assumption method finds the jumps in linearity, with an adaptive threshold for finding where the jumps occur. The linear assumption uses an estimated slope to be used as a reference slope for expected 2π breaks in linearity.

This threshold is based on the two points around the center point, an initial estimate of the slope. Since the DFT is of a real valued signal, the frequency signal is conjugate symmetric, as seen in Section 2.1. This means that the points around the zero frequency should be free of wrapping. The local unwrap code used the points around zero as an initial estimate of the slope, m_{ref} . The initial estimate was used with the index of the points to find approximations of the unwrapped positions at those indexes. The linear assumption method unwraps forward and back of the zero point until all indexes are covered. The jumps are found by looking at the phase values, ψ to an expected location for the point, $x[k] \cdot m_{ref}$:

$$error[k] = \psi[k] - x[k] \cdot m_{ref} \quad (3.10)$$

If,

$$error > \pi \quad (3.11)$$

Then there is a break at $x[k]$, where x is an array of the relevant indices. Progressing through all of the points of interest, $\psi[k]$, will have the π jumps removed, linearizing the signal phase.

The assumption of the points around the zero point being close to the intended unwrapped slope was later found to not always be true. This can be seen in Figure 3.3, where the points near the zero have a different slope than the rest of the indexes. This slope reversal would cause the reference slope to incorrectly unwrap the signal phase points.

This problem forced a change to the way the jumps are found. The

3. ONE-DIMENSIONAL LINEAR PHASE ALGORITHM TESTS

Figure 3.3. *Reference Slope Error. The points around the zero are shown to have a different slope.*

reference slope's absolute value is used as the check for jumps, rather than the signed value. The values are also checked against the neighboring points. The jump location changed to:

$$|(\psi[k] - \psi[k-1]) - (x[k] - x[k-1]) \cdot m_{ref}| \quad (3.12)$$

3.3.2 Phase Reconstruction using Geometric Slope Bounds

Bounds can be placed for the slope unwrapping based upon the closest pixel estimate. The geometric unwrapping uses these bounds to help guide the unwrapping process. This approach does not change the way the exponent of the Fourier series coefficient is obtained, but changes the unwrapping method. The LP algorithm uses the pixel accurate estimate as a bound for the signal phase points to wrap into. Since the peak shift is known to be accurate to within one pixel, the slope of the signal phase must also be bounded (equation 3.13). From the peak bounds, bounds on the signal phase can be created.

The slope of the signal phase is bounded as seen in equation 3.13:

3.3. Phase Unwrapping

Figure 3.4. *Geometric Bounds used for unwrapping a small delay (left). Geometric Bounds used for unwrapping a large delay(right).*

$$\frac{-(\alpha + \delta_p)2\pi}{N} < m < \frac{-(\alpha - \delta_p)2\pi}{N} \quad (3.13)$$

The δ_p corresponds to the bounds above and below the actual peak location. The bounds are chosen based upon the nearest pixel found. The bounds of the expected slope will cause any discontinuities higher or lower to be unwrapped correctly. Figure 3.4 (left) shows the geometric unwrapping being performed on a lower delay phase signal. The two lines provide maximum and minimum limits for the unwrapping to occur. An example showing a larger delay is seen in Figure 3.4(right). Delays higher than the one presented in Figure 3.4 unwrap similarly, though, due to scale are difficult to see, thus are not shown.

The bound is created for every point that is to be unwrapped. The bounds allow the unwrapping to occur without having to search for addition jumps in the signal phase. This reduces the amount of calculations due to the actual unwrapping, though the number of calculations required to create the bounds are comparable to the number for the unwrapping.

3. ONE-DIMENSIONAL LINEAR PHASE ALGORITHM TESTS

3.3.3 Phase Reconstruction using Algebraic Simplifications

Algebraic simplification is another approach to determining phase. The method separates the phase term into a sum of an integer component and a fractional component. This effectively generates a new reference image. The new reference image peak positions lie on the nearest integer pixel position to the shifted image. This is done using the Time-shifting Theorem (equation 2.8 and 2.10), separating the integer component before the DFT is performed. As mentioned the Fourier series coefficient can be expressed as a sum of a integer part and a fractional part. From the shifted image:

$$I[n - \alpha] \longleftrightarrow e^{\frac{-j2\pi k\alpha}{N}} S_0[k] \quad (3.14)$$

Where the shift is separated, and substituted into:

$$\alpha = \alpha_i + \alpha_f \quad (3.15)$$

$$m = \alpha - \alpha_i \quad (3.16)$$

The shifted sequence and corresponding DFT, can now be redefined as:

$$I[m - \alpha_f] \longleftrightarrow e^{\frac{-j2\pi k\alpha_f}{N}} S'_0[k] \quad (3.17)$$

Where the new reference image would use the peak detect estimate to get:

$$I_0[m] \longleftrightarrow S'_0[k] \quad (3.18)$$

This allows the fractional component of the phase term to be isolated in the form seen in equation 2.16:

$$Y'[k] = \frac{e^{\frac{-j2\pi k\alpha_f}{N}} S'_0[k]}{S'_0[k]} = e^{\frac{-j2\pi k\alpha_f}{N}} \quad (3.19)$$

Finding the phase angle is the same as in equations 2.17 and 2.18. The peak detect estimate used would be reapplied to the delay τ . The removal of the integer component prior to the phase correlation simplifies the slope of the angular fractional component. The simplification significantly decreases the amount of unwrapping required.

3.3.4 Comparison of Unwrapping Methods

For this comparison, a data set spanning the whole pixel array is used. The LP algorithm is applied with the different unwrapping methods, where the error in the peak position estimation is observed. Looking at the whole pixel array, the error to peak position estimation was observed for each unwrapping method. Figure 3.5 shows the error of the LP algorithm based upon the unwrapping method with images ranged over the whole pixel array. At the 66% mark of the pixel array, the local unwrapping method begins to fail. The peaks in the two peak pattern approach the edge of the sun-sensor at 89% of the pixel array, where the LP algorithm is not able to determine the peak pattern.

A closer look at the peak position estimation error is also displayed in Figure 3.5 (right). Peak detection is not usable at this scale, and is thus omitted. The local unwrap method and the algebraic simplification obtain the smallest error. These two unwrapping methods produce error of the same order of magnitude when shifts are near the center of the pixel array. The error is from the LP algorithm and noise, not the unwrapping methods themselves. The geometric unwrapping is seen to produce slight errors in these regions.

3. ONE-DIMENSIONAL LINEAR PHASE ALGORITHM TESTS

Figure 3.5. Error with shifts spanning the whole pixel array (left). Close up of the error around the center of the pixel array (right). The error in peak position estimation (δ_{eff}) related to the shift location in pixels (x -axis).

3.4 Effect of Shortening the Image

Only a small fraction of pixels in each image are illuminated, the rest of the image is composed of relatively low level noise. Only using the relevant portions of the image would reduce the amount of computations required to apply the LP algorithm. The shortening would also be used to reduce the effects of noise. This test is to study the effects of the shortened image on the peak position estimation performance of the LP algorithm. Reducing the amount of calculations and the effects of noise would be beneficial to the development of the LP algorithm.

Shortening of the image caused a problem. The LP width parameter, S , that defines the length of the linear section, becomes long such that the points used to calculate the slope of the signal phase is no longer linear. Figure 3.6 shows how the length of the width parameter, S , corrupts the slope calculation. This prompted a test to determine a LP width parameter that remains in the linear section. An LP width parameter related to the window size was found (Section A.1). A LP width parameter of $S = \frac{\text{window length}}{4}$, was found to no longer use the non-linear points of the signal phase.

Figure 3.6. *Linear section shortened due to a shorter image used. A constant width parameter, S , will use noisy values that will corrupt the slope estimation.*

Ensuring an efficient DFT was one problem that arose during this shortening. The issue appears when the window length is not of a length that can simplify DFT calculations by using an FFT. This issue was easily resolved by using a fixed length of 2^n . Testing the optimal length yielded interesting results (Section A.2). The shortening of the image did not adversely affect the precision provided by the LP algorithm. Section A.2 shows that the LP algorithm performs better using a shorter image length. This allows the LP algorithm to use fewer calculations, using a shorter DFT, without losing peak position estimation precision.

An attempt to improve performance while shortening the image was to zero-pad the shortened image. Zero padding is similar to zeroing the low level noise around the peaks. The results show the same trend as increasing the window length. The results of the zero-padding were similar to the windowing results, in that the reduction of noise (replaced by the zero-padding) had more effect on the LP algorithm than the change in the length of the image.

3.5 Summary

The phase effects on the LP algorithm when shifts are near the edge of the FOV were studied. It was found that the signal phase becomes difficult to unwrap for the native unwrap function. Several methods were found to resolve the problem. The local unwrapping and the algebraic simplification methods were able to achieve estimate the peak positions with the least amount of errors. The geometric bounding method was found to not be able to perform as well.

The effects of windowing on the LP algorithm was also studied. Shorter windows did not impair the peak estimation performance of the LP algorithm.

From this information, the transition to two-dimensional analysis is made simpler due to the images produced by the two-dimensional system. These images are presented in Section 4.2. Windowing will appear in Section 4.4.3 and Section 5.5. The phase effects due to large shifts will appear in partly in Section 4.4.3.

Chapter 4

Two-Dimensional Algorithm Testing and Simulation

One-dimensional testing is a good starting point for analysis on the precision performance of the LP algorithm. However, the one-dimensional system is not realistic. A more realistic system to study is the two-dimensional system. Adapting the LP algorithm into the two-dimensional system is a precursor to implementing the algorithm onto an embedded system.

This chapter will discuss the two-dimensional system studied. The first topic discussed will go over the two-dimensional system and how the images are simulated. A look into the sensor images will follow. From there, the different algorithms that are migrated into the two-dimensional system are shown. A baseline algorithm is developed and used as an upper bound for error and a lower bound for precision. The approach to implementing the LP algorithm using the new sensor images, is discussed and the implementation method is shown. The method in which real sensor data is used to verify the simulated data is explained following the algorithms. Several sources of error are explained as well as the corrections associated with them. The results of the final implementation of the LP algorithm in the Matlab environment are then shown.

Figure 4.1. Mask to pixel array arrangement of a two-dimensional sun-sensor. The half slit widths of the slits parallel to the X-axis, D_x , and the half slit widths of the slits parallel to the Y-axis slits, D_y . The pixel array is an angle, ψ from the X-axis.

4.1 Simulation of Images from a Two-Dimensional Sun-Sensor

Digital sun-sensors mask incoming light and diffract light onto a pixel array. A one-dimensional sun-sensor is described in Section 3.1. In the one-dimensional sun-sensor described, the pixel array is orthogonal to the mask slits, this is not true for the two-dimensional system studied. The mask in the two-dimensional sun-sensor has two pairs of orthogonal slits. The different spacing for each pair of slits ensures that sufficient information is accessible even in non-trivial image signals. Figure 4.1 displays the relationship between the mask and the detector array.

Since the pixel array is at an angle, to both pairs of slits, the two pairs of slits will diffract their respective patterns onto the pixel array. Separately, the two slit patterns essentially work as one-dimensional systems, due to their superposition the possibility of the patterns interfering with each other becomes a problem. The sensor image can be simulated using an expanded

4.2. Types of Images from a Two-Dimensional Sun-Sensor

form of equation 3.9. The expansion encompasses the two different normalized slit-widths, and the two shifts. The sensor image is approximated by the equation:

$$I[n] = I_c(n\Delta T + L_x - \tau_x) + I_c(n\Delta T - L_x - \tau_x) \\ + I_c(n\Delta T + L_y - \tau_y) + I_c(n\Delta T - L_y - \tau_y) + n_\sigma[n] \quad (4.1)$$

Equation 4.1 is the approximated sensor image that is to be processed. Following from equation 3.8, two reference images are created based on the different normalized half slit-widths:

$$I_x[n] = I_c(n\Delta T + L_x) + I_c(n\Delta T - L_x) \quad (4.2)$$

$$I_y[n] = I_c(n\Delta T + L_y) + I_c(n\Delta T - L_y) \quad (4.3)$$

Adaptation to the two-dimensional system used equation 4.1 to simulate test images.

4.2 Types of Images from a Two-Dimensional Sun-Sensor

In the two-dimensional system, the two patterns are superimposed, causing difficulties during processing. Images can be classified into several categories. For a majority of the field of view (FOV), the patterns do not interfere with each other, allowing for simple logical pattern-to-slit identification. These cases occur when the peaks for one axis are sufficiently far enough away from the other axis, on the sensor array. This is the simplest case, where the information provided by the sensor is easily processed to determine an estimate sun vector. This simple case can be seen in the top left of Figure 4.2. More problematic cases include; *overlap cases*, when only three peaks appear

4. TWO-DIMENSIONAL ALGORITHM TESTING AND SIMULATION

in the image; *interwoven cases*, when at least one set of the pairs peaks are not sequential; and *interfering cases*, where peak widths are overlapping or extremely close to one another.

Overlapped cases occur when a peak from one slit pattern superimposes upon a peak from the other slit pattern. In this case, the sensor image only has three peaks. In most instances of an overlap case, the overlapping peak also has a higher intensity than the other peaks. The pattern to slit pairing is thus required to be done using the information from the non-overlapping peaks. The top right of Figure 4.2 is an example of an overlap case, where three peaks appear due to the superposition of peaks.

Interwoven cases occur when the peaks are mixed such that the four peaks are visible but not clearly separated. Peaks are either alternating, or the narrow slit peaks lie within the wide slit peaks. Separating the peaks, causes the index of one set to be non-linear, causing a problem when separating the peaks. The bottom right of Figure 4.2 shows an interwoven case, the peaks for one axis are alternating with the other axis.

Interfering cases are a sub class of interwoven and overlapped cases. Since the light passing through the masks are diffracted, the peaks have a certain width. The width of a peak is able to superimpose upon another peak's width causing interference. The tip of the peaks may be clearly defined, though due to the overlapped widths, separation of the peaks becomes a problem. The bottom left of Figure 4.2 is an example of an interfering case, the peak's amplitudes are not significantly superimposed, however, the peak widths overlapping may cause errors to certain algorithms.

The algorithms used for a two-dimensional problem must be robust enough to process these types of problem images. These three types of images can be placed onto a FOV plot of the of the sensor, as seen in Figure 4.3. An FOV plot displays the error at each sun vector location. The error is between the estimated sun vector location and the actual sun vector location. For Figure 4.3 the error is modified to display the locations of the problematic cases

4.2. Types of Images from a Two-Dimensional Sun-Sensor

Figure 4.2. *Sensor Images of various cases. Simple Case (top left), Overlap Case (top right), Interfering Case (bottom left), Interwoven Case (bottom right).*

4. TWO-DIMENSIONAL ALGORITHM TESTING AND SIMULATION

Figure 4.3. *FOV plot of problematic case locations. This FOV displays the locations where the problematic cases occur at the sun vector points.*

at the sun vector location. This is useful later on to show which types of images an algorithm has problems with, as well as to determine where an algorithm's strengths are.

4.3 Two-Dimensional Shift Estimation Algorithms

Similar to one-dimensional testing where the peak detection served as a baseline for precision [Godard, 2004], a two-dimensional baseline that can be tested against was needed.

4.3.1 Case Determination

The peak detection algorithm was expanded to function for an image from a two-dimensional sun-sensor. The peak detection algorithm was able to find the brightest pixels in the sensor image, for a one-dimensional case, this would provide a pixel accurate estimate of the shift. Expanding on this, the case determination method finds the orientation of the peaks based on the peaks found in the peak detection. An example of the pseudo code to find

the maximum intensity of the sensor image is shown in algorithm 1.

Algorithm 1 Finding maximum intensity.

```

imax = 0
for k = 0 to 255 do
    if I[k] > imax then
        imax = I[k]
        peak = k
    end if
end for

```

From the maximum intensity found, a threshold is created to zero out intensity values that fall below the threshold. The case determination method then searches for points that are higher than its neighbors. The index locations for these points are taken as the pixel accurate peak positions. After finding the peaks in the image, determining the orientation of the peaks follows. The slit spacing is different for each axis and is a known quantity. From this spacing an expected peak pattern for each axis can be found [Enright et al., 2008]. The known spacing allows a simple if statement to ascertain the orientation of the peaks, the peak to slit matching. In the instance of four peaks, the peak to slit matching is simplified in that each set of slits will have a set of peaks. Figure 4.4, shows the five different 4 peak cases on the right. For 3 peaks, there are four cases, though one set of the peaks are overlapped. The 3 peak cases are seen on the left of in Figure 4.4.

Using these known cases, the case determination method evaluates the error found for each case. The minimum error found will be the most likely peak orientation. A sample of the pseudo code is seen in algorithm 2. Where peak locations $[x_1 x_2 x_3 x_4]$ are in pixels found above, and q is the orientation of the pixels, detailed by the labels $[1a 1b 2a 2b]$ referring to the narrow left, narrow right, wide left and wide right peaks respectively.

Having logically matched the found peak spacing to an expected peak spacing an estimate of the signal shift is obtained by averaging the pixel indexes of the respective pairs. Since only peak detection is used, this algo-

4. TWO-DIMENSIONAL ALGORITHM TESTING AND SIMULATION

Figure 4.4. *Two-dimensional sensor images. 3 peak orientation cases (left) 4 peak orientation cases (right).*

Algorithm 2 Peak to Slit Matching

```

given peaks = [x1 x2 x3 x4]
minerr = ((x2-x1)-Dy)-((x4-x3)-Dx)
q = [1a 1b 2a 2b]
test = ((x3-x1)-Dy)-((x4-x2)-Dx)
if test < minerr then
    minerr = test
    q = [1a 2a 1b 2b]
end if
test = ((x3-x2)-Dy)-((x4-x1)-Dx)
if test < minerr then
    minerr = test
    q = [2a 1a 1b 2b]
end if
test = ((x4-x2)-Dy)-((x3-x1)-Dx)
if test < minerr then
    minerr = test
    q = [2a 1a 2b 1b]
end if
test = ((x4-x3)-Dy)-((x2-x1)-Dx)
if test < minerr then
    minerr = test
    q = [2a 2b 1a 1b]
end if

```

4.3. Two-Dimensional Shift Estimation Algorithms

rithm is only pixel accurate. One source of error for this type of algorithm was the use of possibly corrupted information. Looking at the problem cases mentioned in Section 4.2, peaks may shift slightly due to the interference of the peaks with each other.

As an amendment to the case determination, centroiding around the peaks was introduced for sub-pixel accuracy. Centroiding the peaks by using the first moment of area equation, finds the tip of the peak within a pixel. A three point centroid equation would be in the form:

$$x'_n = \frac{I_{n-1} \cdot x_{n-1} + I_n \cdot x_n + I_{n+1} \cdot x_{n+1}}{I_{n-1} + I_n + I_{n+1}} \quad (4.4)$$

Higher order centroids were used based on the width of the peak. The problem that comes with centroiding was the requirement of a minimum width for the centroiding to take place. Interfering cases (Section 4.2), would sometimes cause the centroiding to presume there was only one peak at that location. If the centroid width was too large, the centroiding would become corrupt if the peaks are close together causing the peaks to be indistinguishable from each other.

The case determination method was used as a baseline for which other algorithms could be compared against. Under no conditions should more precise algorithms perform worse than the case determination method. This comparison was useful for offline comparison of algorithms.

The pixel accurate estimates produced by the case determination method are also useful for online use. Some algorithms may require a pixel accurate estimate for processing, phase reconstruction using geometric bounds (Section 3.3.2) and phase reconstruction using algebraic simplifications (Section 3.3.3) are some examples. The estimate from the case determination method can also be used as an error check during online processing. Estimates that are significantly different from the pixel accurate estimate can be seen as incorrect.

4. TWO-DIMENSIONAL ALGORITHM TESTING AND SIMULATION

4.3.2 Cross Correlation

Cross correlation is a common technique for detecting a known signal component within a larger signal. Cross correlation, however, is discrete, and thus is only able to find the pattern to pixel accuracy. This method was also very calculation heavy, requiring two cross correlations to be performed. This algorithm was also susceptible to amplitude modulation from overlap cases.

Cross correlation finds how closely one signal is to another signal. Since a reference expected pattern was known for each slit pair, cross correlation was a method used to obtain the pattern to slit pairing. Cross correlation convolves two signals together:

$$I_{cc} = \sum I[k - n] \cdot I_{ref}[k] \quad (4.5)$$

Where, the sensor image, $I[k]$ is convolved with the reference image I_{ref} , to produce the signal I_{cc} . This signal (I_{cc}) have a peak where I_{ref} appears in $I[k]$. This location was directly related to the estimate of the sun vector. Figure 4.5 is an example of how the cross-correlation performs. One reference image was correlated with the sensor image to produce a signal whose intensity was an indication of the likelihood of the signal appearing in a specific position. The example uses the wide peak pair as the signal sought.

This algorithm still required the use of a logic block similar to the case determination method. Cross correlation was susceptible to peaks being equi-distance apart, as well as overlap cases that have an amplitude modulated. Cross correlation can be calculation heavy, this can be rectified with windowing. Figure 4.6 is an example of equi-distance image causing problems to the cross-correlation signal. The cross correlation method would produce an incorrect estimate for the wide peaks. The incorrect estimate appears because the spacing between the pairs of peaks (between the right narrow peak, and left wide peak) is similar to the spacing between the wide peaks. The cross-

4.3. Two-Dimensional Shift Estimation Algorithms

Figure 4.5. *Cross-Correlation Signal Example. The correlation signal, I_{cc} is the sensor image, I , convoluted with the reference signal, I_{ref} . A peak is created where I_{ref} appears in I .*

correlation signal would produce a second peak because the reference image appears twice. This was remedied with additional logical analysis, but this required additional calculations in addition to the already calculation heavy algorithm.

4.3.3 LP Algorithm

The LP algorithm showed promise for one-dimensional testing [Godard, 2004]. Moving into two-dimensions, the process into which the phase correlation is performed was studied to see if this signal delay method was viable. However, the LP method is not mathematically simple in a direct two-dimensional case. This was due to the superposition of the slit patterns. Simplifying equation 4.1 with equations 4.2 and 4.3, and substituting $\alpha = \tau_1$ and $\beta = \tau_2$:

$$I[n] = I_x[n - \alpha] + I_y[n - \beta] + n_\sigma[n] \quad (4.6)$$

Following from Section 2.3 taking the DFT of the image would yield:

4. TWO-DIMENSIONAL ALGORITHM TESTING AND SIMULATION

Figure 4.6. *Problematic Case, equi-distance. The correlation signal, I_{cc} is the sensor image, I , convoluted with the reference signal, I_{ref} . A peak is created where I_{ref} appears in I . The reference image, I_{ref} , appears twice due to the equal spacing between the peak pairs.*

$$S[k] = e^{\frac{-j2\pi k\alpha}{N}} S_{x_0}[k] + e^{\frac{-j2\pi k\beta}{N}} S_{y_0}[k] + N_\sigma[k] \quad (4.7)$$

The next step would be to isolate the time-shifting terms, $e^{\frac{-j2\pi k\alpha}{N}}$ and $e^{\frac{-j2\pi k\beta}{N}}$. So:

$$\frac{S[k]}{S_{x_0}[k]} = e^{\frac{-j2\pi k\alpha}{N}} + e^{\frac{-j2\pi k\beta}{N}} \frac{S_{y_0}[k]}{S_{x_0}[k]} + \frac{N_\sigma[k]}{S_{x_0}[k]} \quad (4.8)$$

The time-shifting terms, $e^{\frac{-j2\pi k\alpha}{N}}$ and $e^{\frac{-j2\pi k\beta}{N}}$, cannot be isolated in a usable manner to find the shifts, α and β . For this reason, a direct two-dimensional phase correlation method to estimate the shifts was not found.

4.4. Decomposition of an Image from a Two-Dimensional Sun-Sensor

Figure 4.7. *Image separation flow chart.*

4.4 Decomposition of an Image from a Two-Dimensional Sun-Sensor

In order for higher precision one-dimensional algorithms to work for a two-dimensional system, the image must be separated into the contributing parts from the two pairs of slits. This means that the sensor image is transformed into two separate one-dimensional images to be processed by the algorithm, twice, once for each new image corresponding to an axis. Segmenting the image is done systematically, knowing that the spacing between the peak pairs is directly correlated to the mask spacing (equation 2.18). Figure 4.7 displays the approach taken to separate the image into its two constituent parts.

4.4.1 Peak Identification

The first step of processing the sensor image was to find a coarse estimation to the peaks in the sensor image. The coarse estimation is the pixel index of the tip of the peaks in the image. This step was performed finding the local maxima as well as local minima of intensity in the image, alternating to find the peak points as well as the valley points. The peak points will be used to consider which orientation the peaks are in, while the valley points are used as an index to separate the image. Figure 4.8, shows the peaks and valleys found for a 3 peak image (left) and a 4 peak image (right).

4. TWO-DIMENSIONAL ALGORITHM TESTING AND SIMULATION

Figure 4.8. *Identified Peaks and Valleys for sample 3peak (left) and 4 peak (right) images. The locations of the peaks and valleys are found.*

4.4.2 Peak to Slit Matching

From the information gathered from the course estimation, the type of image can be determined from the peak positions. This step is largely based on the amount of peaks found in the previous step. Similar to the case determination method described in Section 4.3.1, the peak to slit matching is performed evaluating the error in each case. The solution that minimizes spacing error provides the most likely estimate of the layout of the peaks. The peaks are labeled according to their corresponding slits. Peaks that are overlapped will have two labels to indicate which peaks are overlapping.

4.4.3 Separating the Image

Once the layout of the peaks is known, the image can be separated by axis. The start of the array, the valley points, and the end of the array are the locations where the separation occurs. The regions between the points will contain a peak. Based on the orientation a new image is created for each axis.

Three and four peak cases are segmented differently. Four peak cases separate the 4 peaks into two 2 peak images. For 3 peak cases, the overlapped

Figure 4.9. *Segmenting an image into constituent parts, 3peak (left) 4peak (right).*

peak is not segmented. The shift is calculated using only the non overlapped peak, the non overlapped peak is matched to a specific slit. The reference image is changed to use only the peak that corresponds to that specified slit. Figure 4.9 shows the 3 peak and 4 peak images from Figure 4.8 segmented into their respective axes.

The two images created from this separation are approached in the same way a one-dimensional image would be approached. The superimposed image is deconstructed into two constituent parts, one image for each axis. Each component image is not a direct decomposition of the superimposed sensor image, just an approximation of the individual images. At this point, the image is able to be processed by high resolution one-dimensional algorithms, such as the LP algorithm.

4.5 Capturing Images from a Two-Dimensional Sun-Sensor

After adapting the LP algorithm into the two-dimensional system using simulated images, testing the LP algorithm with real images captured from a sun-sensor was the next step. The sun-sensor used to capture images was a SS-411 from *Sinclair Interplanetary*. This sun-sensor utilizes a mask with

4. TWO-DIMENSIONAL ALGORITHM TESTING AND SIMULATION

Parameter	Measurement
X-axis Slit Spacing	2.5×10^{-4} mm
Y-axis Slit Spacing	7.5×10^{-4} m
Half-Slit Width	4.25×10^{-6} mm
Detector Array Length	16 mm
Pixel Width	6.35×10^{-5} mm ¹

Table 4.1: SS-411 sun-sensor parameters.

Figure 4.10. Visual schematic of SAIL facility.

two orthogonal two-slit patterns, as described in Figure 4.1. Table 4.1 details the relevant parameters of the SS-411 sun-sensor.

The SS-411 sun-sensor was mounted on a three-axis rotary platform. The three-axis rotary platform was computer controlled to orient the sun-sensor to capture the images. The sun-sensor was mounted near the center of rotation and was small such that the sensor was always illuminated. A xenon arc-lamp was used to simulate the sun for the captured images. Figure 4.10 is a visual schematic of the experimental setup for the sun-sensor.

For this study, the images captured are processed offline. This allows the

Figure 4.11. *Initial results of Matlab LP algorithm. This FOV plot displays the peak position estimation error at the input (true) sun vector point.*

same image set to be processed without having to re-capture the images with the sun-sensor.

4.5.1 Results

These are the results from the LP algorithm over the image set captured by the sun sensor. Looking at the FOV plot, Figure 4.11, several interesting features stand out. The first interesting feature is the significant drop in error seen in the top right quadrant stands out. The other three quadrants seem to have a mean error within their respective quadrants. The second feature is the increase in error with the bore sight angle. This can be seen as a slight increase in error going outward from the origin. The last feature that is of interest is the concentration of error on the main diagonal (going from the top right to the bottom left). From these features, several corrections are applied to the LP algorithm to improve the precision of the peak position estimates.

4.6 Gap Correction and Non-Uniformity

The LP algorithm assumes that the detector array is composed of uniform pixels, for the SS-411, this is not the case. The pixels are broken into two segments. Figure A.3 is a close-up of the gap due to the two pixel dies. Since there are two pixel dies, there is a possibility of miss-alignment. Pixels near the gap are oblong in an attempt to account for the gap [Solutions, 2007]. These manufacturing tolerances cause a small albeit problematic errors to the way the sensor image is interpreted.

4.6.1 Gap Correction

The gap correction resolves the small error due to placement of the pixel dies of the sensor array. The gap effect occurs over the second half of the array, pixel 129 to pixel 256. The gap distorts the sampling locations, the pixel centers, for each axis. Figure 4.12(a) shows how the pixel dies are positioned together. The correct implemented only takes into account a linear translation of the pixel dies. This can be seen in Figure 4.12(b), a linear translation of the second pixel die. Assuming that the sensor array is linear through the gap, the gap correction is simplified into a direct shift of the pixels for each axis. Figure 4.12(c) shows how the direct pixel correction is implemented.

The correction takes the intensities measured and modifies the associated index to match that of the estimated index. For pixels 0 to 127, the gap effect is not a factor, so the index stays the same:

$$I[n] \rightarrow I[n] \text{ for } n = 0, \dots, 127 \quad (4.9)$$

The gap effects the pixels on the second die, so indexing would change for pixels 128 to 256:

$$I[n] \rightarrow I[n'] \text{ for } n = 128, \dots, 255 \quad (4.10)$$

Figure 4.12. *Schematic View of the Pixel Dies. (a) The relative shapes of the actual and theoretical pixels around the gap. (b) A translational displacement in the second pixel die. (c) Relating a gap correction to the illumination line shows that the gap correction is a direct index shift.*

Where,

$$n' = n + \gamma \quad (4.11)$$

The gap parameter, γ , is different for each axis, based on the parametrization. Correcting the indexing for the LP algorithm resolves the problem with the gap between pixel dies by using a numerical approach.

4.6.2 Non-Uniformity

The shapes of the center four pixels of the detector differ from their neighbours. It appears that these elongated pixels were used to improve the uniformity of pixels appearing over the whole detector. Figure 4.13 shows how the center four pixels are elongated to have similar surface areas. The reason for this is to decrease the width of these four pixels to allow a space for the two pixel dies to be attached [Solutions, 2007]. The areas of the four pixels are detailed in the technical specifications, allowing the center of these pixels

4. TWO-DIMENSIONAL ALGORITHM TESTING AND SIMULATION

Figure 4.13. *Non-uniform pixels near the die gap.*

to be found which is the estimated pixel index. Taking these modified pixel indexes and the obtained pixel indexes into account, a fix relating to the non-uniformity of the center pixels was approximated.

Knowing the pixel centers of the non-uniform pixels and the ideal uniform pixels, a polynomial interpolation was used to determine the corrected index arrays. A Lagrange Polynomial Interpolation is used to calculate the corrected index points.

Given a set of $k+1$ data points:

$$(x_0, y_0), \dots, (x_k, y_k) \quad (4.12)$$

For this correction, the two sets of points would be the uniform and the non-uniform pixel center locations:

$$L(x) = \sum_{j=0}^k y_j l_j \quad (4.13)$$

Where the Lagrange basis polynomials are defined:

$$l_j(x) = \prod_{i=0, i \neq j}^k \frac{x - x_i}{x_j - x_i} \quad (4.14)$$

From here the corrected index locations would be applied to the center four pixels:

$$I[n] \rightarrow I[n'] \text{ for } n = 126, \dots, 129 \quad (4.15)$$

This corrects for the non-uniformity in the pixels near the center of the pixel array.

4.6.3 Interfering Cases

Interfering cases are problematic due to the width of the peaks. In the case of overlap peaks, the overlapping peaks do not take into account the width of the overlapped peaks. As a proposed fix to the band of error, on the main diagonal in Figure 4.11, was to change the way the LP algorithm processes interfering cases into the same way overlap cases are processed. This means that for interfering cases, only the non-interfering peaks are used in determining the estimate of the shift.

4.6.4 Results

Having applied the corrections, the results, using the same image set, can be seen in Figure 4.14. The gap correction significantly removed much of the errors seen as quadrant specific. The significant drop in error in the top right quadrant is now relatively uniform across all of the quadrants. The error seen with an increase in bore sight angle seen in the bottom left quadrant in Figure 4.11, has seemed to be corrected and the concentration of error on the main diagonal has been corrected to some degree, though errors still seem concentrated in that area compared to the rest of the FOV.

4. TWO-DIMENSIONAL ALGORITHM TESTING AND SIMULATION

Figure 4.14. *Final Matlab LP algorithm results. This FOV plot displays the peak position estimation error at the input (true) sun vector point.*

Algorithm	Mean Error
LP	8.93×10^{-4}
NLSQ	11.89×10^{-4}

Table 4.2: Mean Error over the FOV of the same image set.

The results obtained here for the LP algorithm can be compared to those of the non-linear least squares (NLSQ) algorithm in Figure A.4. The NLSQ is compared against because the NLSQ algorithm is the current algorithm being used on the SS-411 sun-sensor. The results for the NLSQ use the same image set (Section A.4). Comparing the two, the LP algorithm is able to achieve a lower mean error. The error in the LP algorithm seems to be more localized to several regions, where the NLSQ results have error that is not as localized. The mean errors for each algorithm are detailed in table 4.2.

4.7 Conclusions

Images were simulated for a two-dimensional sun-sensor. From these simulated images, algorithms were adapted to the two-dimensional system. The case determination method was adapted from the peak detection algorithm

and is used as a baseline for other algorithms to compare against. The LP algorithm was adapted to the two-dimensional case by segmenting the two-dimensional image into two constituent parts to be processed.

After adapting the LP algorithm to the two-dimensional case, the simulated images were verified using real captured sun-sensor data. From this real data, several corrections were required. The errors caused by the manufacture of the pixel array used on the SS-411 were corrected. The method in which interfering cases were processed was also changed to account for the associated errors. The LP algorithm is able to achieve a high degree of precision, comparable to the current algorithm used on the tested sun-sensor.

Having developed the LP algorithm to perform in a two-dimensional system, moving onto testing the LP algorithm outside the Matlab environment was the next step.

Chapter 5

Embedded Implementation

To test the algorithm in a more realistic environment the LP algorithm was migrated to an embedded microcontroller. The microcontroller was used to simulate an onboard sun-sensor processor. This required the Matlab LP algorithm to be migrated into C code. This implementation of the LP algorithm was used to verify the Matlab results in a more realistic processing environment.

This chapter discusses the migration of the LP algorithm into the C code for embedded implementation. The microcontroller used for the implementation is the C8051F410 board. Several of the problems and restrictions imposed are discussed. Major challenges regarding the implementation of the DFT are explained, as well as the method used that was able to overcome the restriction is presented. Other memory optimizations are described in this chapter.

5.1 Migration Approach

The approach to migrating the LP algorithm into C code was performed methodically. Figure 5.1 is a flow-chart detailing the migration approach. The first step was to change the Matlab functions into C-like functions. This

5. EMBEDDED IMPLEMENTATION

Figure 5.1. *Migration approach flow chart.*

meant that high-level Matlab code was removed and replaced with c-style syntax and hand-coded mathematic functions. The original Matlab code was constantly used as a benchmark to test the rewritten functions. Once this was done for all of the functions, the C-style Matlab code was converted into C. To test the C code, a Matlab executable (mex) wrapper was written so that the C code was able to be tested in the Matlab simulated environment. The C code was tested against the original Matlab code to ensure that the algorithm functioned to the same degree of performance.

5.2 The C8051 Microcontroller

The testbed for the C code was a C8051F410 development kit board. This testbed imposed several restrictions on the C code. In table 5.1 the memory size restrictions are shown. Although the code space was sufficient, the main difficulty was executing the code with the limited variable and working space. This was a problem for the conversion to C code of the LP algorithm, requiring the LP algorithm to require several rounds of optimizations.

To test the LP algorithm on the 8051, the algorithm is loaded into the microprocessor. Each image is sent to the board over a serial link. The image is sent one pixel of information at a time. The onboard algorithm processes the image and sends the result back on the serial link.

Type	Memory	Description
cdata	48 kilobytes	Executable and Constants
xdata	2048 bytes	Slow-speed RAM
data	128 bytes	High-speed RAM

Table 5.1: Memory Restrictions from the 8051 development board [Laboratories, 2006].

5.3 DFT

The LP algorithm requires the use of a DFT. In the Matlab version of the code there is a pre-defined FFT function. This was used in the Matlab code; however, C code does not have built-in functions. Standard libraries and implementations are available but memory limitations imposed made a direct application of the libraries difficult. Finding a working DFT function was problematic due to the amount of memory required for the the amount of output variables. This memory requirement served as the first obstacle to implementing the C code. For most computational implementations of DFTs, FFTs are used. This is because FFTs are able to compute DFTs of signals using a fraction of the computations due to symmetries based on signal lengths of 2^n . The computational requirement is also reduced by the symmetries produced by taking the DFT (or FFT) of a real valued sequence. Since the FFT is a common programming function, outside sources of a C code version of the FFT were found. As with many of the alterations to the core LP algorithm, the functions and changes are tested within the Matlab environment before testing on the C8051 development board.

5.3.1 Butterfly FFT

One version of a DFT implementation in C code was the butterfly FFT [Karn, 1997]. Mathematically and computationally this DFT method is based on the principle of bit swapping and the four-point butterfly. In the Matlab

5. EMBEDDED IMPLEMENTATION

environment, the mex wrapped C code was able to perform the LP algorithm using this DFT method without any problems. However, once testing on the board began, there were problems. The butterfly DFT was memory heavy, which was partially solved reusing variables. However, there was also a requirement on trigonometric functions, which are computationally heavy. The butterfly FFT also required the use of four 256 element arrays. This was the biggest problem with this DFT solution, memory size. The resultant code size with the required variable space did not fit onto the 8051 board memory. Due to these reasons, this solution for the C code DFT was abandoned.

5.3.2 Radix-2 FFT

Another DFT implementation in C code was the radix-2 DFT [Jones, 1992]. Looking at the formulation of the radix-2 DFT, the mathematics involved showed that the variables did not exceed the memory requirements. The code also utilized an efficient method to do trigonometrics, look-up tables. The radix-2 code was modified and applied to the existing LP C code. Again, in the Matlab environment, the LP C code was able to function, but on the 8051 board there was a problem. The code was able to fit onto the board, however, the problem was that the results produced are incorrect.

5.3.3 Goertzel Method

The Goertzel Method (selective DFT) is based on the first principles of DFT. Looking at the very core of the LP algorithm, the DFT of the reference image, $S_0[k]$, is constant based on the type of image. Only a subset of these values are used to correlate phase against the DFT of the shifted image, to avoid the zero points in $S_0[k]$. This was an important distinction, since this allowed the desired values to be indexed. The index of the values used in the phase correlation are constant for $S_0[k]$. This indexing significantly decreased the amount of memory space required by only performing the DFT on the

required index points. In a technical sense since the selective DFT is not an FFT method, the number of calculations may be higher, though, this is justified since the amount of points calculated is significantly less than in an FFT over the whole 256 array. By only calculating the DFT at specific indices, these constants were tabulated with respect to image type. The tabulated results are used as look-up tables. Since these are pre-generated these tables are put into code memory, significantly decreasing the variable memory requirement of the LP algorithm.

An example of the implementation of the selective DFT is shown in algorithm 3. The selective DFT takes advantage of the symmetries in the DFT (Section 2.1), as well as the algebraic simplification for the phase angle slope (Section 3.3.3).

Algorithm 3 Pseudo code of selective DFT.

```

for (i = 0 to length(k)) do
   $W_t = \text{twiddlelookup}[i]$ 
   $W = 1 + 0j$ 
  for n = 0 to 255 do
     $n_{alg} = n + peak_{est}$  //algebraic simplification
    if  $((1a - hpw) < n_{alg} < (1a + hpw) \text{ or } (1b - hpw) < n_{alg} < (1b + hpw))$ 
    then
       $S_{tmp} += I[n_{alg}] * W$  //sum I with twiddle factor
    end if
     $W = W * W_t$  //increment twiddle factor
  end for
   $S[\text{length}(k) + i] = S_{tmp}$ 
   $S[\text{length}(k) - i] = \text{conj}(S_{tmp})$  //due to symmetry
end for

```

5.4 Algorithmic Memory Optimizations

Parallel to the memory reduction from the DFT, memory optimizations to the functions of the LP algorithm were sought. The memory usage still

5. EMBEDDED IMPLEMENTATION

exceeded the available storage on the C8051 board. For this reason optimizations that reduce the memory usage and computational efficiency were found and implemented. A profile of the C code provided insight on the functions of the LP algorithm. The amount of time spent in each function and the amount of times a function was called was found from this profile. This allowed more specific optimizations to take place. Optimizations such as changing frequently called functions into in-line calculations, and memory savings by reusing variables came about from these optimizations.

5.4.1 Pre-computing Constant Arrays

Before the indexing of the constant arrays, four 256 length float arrays were required for the signal phase analysis. Four arrays were required, two for the unshifted image and two for the shifted image, due to the DFTs producing real and complex results as seen in Section 2.1 for both the shifted and unshifted images, these working variables were a significant problem. Just these four arrays would use all of the variable space.

$$4 \text{ arrays} \cdot 256 \text{ elements} \cdot 4 \text{ bytes per element} = 4096 \text{ bytes}$$

As mentioned in Section 5.3.3, the values for the unshifted image arrays were tabulated as a header file to be used as a look-up table. This by itself would be a saving of half of the variable space. Indexing the relevant parts of the image decreases the variable memory usage even further. Optimizations for some of trigonometric functions were also applied using this look up table method. Below is a list of the values recorded in the header file:

- (half) index of DFT values used
- DFT of reference image
- length of index values

- sum of index values
- sum of index values squared
- twiddle factors used for the DFT of shifted image

A set of these values is required for the six different cases. The cases being:

- wide peaks
- narrow peaks
- wide left peak
- wide right peak
- narrow left peak
- narrow right peak

5.4.2 Polynomial Fitting

The original instance of the linear fit to determine the slope of the unwrapped signal phase, was calculation heavy due to summing (Equation 2.28). The indices where the signal phase is calculated are found constant allowing the indices and some of the summing to be pre-calculated and tabulated. The polynomial fit was also redefined using a closed form equation. The resultant function was transformed into an in-line function:

$$slope = \frac{(N \cdot \sum xy) - (\sum x \cdot \sum y)}{(N \cdot \sum x^2) - (\sum x \cdot \sum x)} \quad (5.1)$$

In equation 5.1, there are many constants, only the sums, $\sum xy$ and $\sum y$, need to be calculated, the rest are tabulated based on the image type. This modification reduced the number of calculations, and reduced the amount of working memory required by pre-calculating and storing the values of the indices and their respective sums in the header file.

5.4.3 Slope Unwrap

The original method of unwrapping relied on finding breakpoints within the signal phase. The slope unwrap was rewritten more efficiently. Instead of searching for breakpoints, correcting and repeating the process, the unwrap now calculates the correction required for a point before applying the correction. The correction is also recorded for the next point, since the signal phase is linear, a correction by an amount at least as much as the previous correction is required. This allows the slope unwrap to only have to apply corrections once, without repeating previously corrected points.

In addition to the modification to the slope unwrapping, the algebraic simplification, based on the work in Section 3.3.3, was instituted to reduce the amount of slope unwrapping required. The image segmentation process required finding the peak estimates, and making use of that information allows the slope unwrap to avoid highly wrapped signals. This reduces the wrapped signals found problematic in Section 3.2 from high peak shifts.

The amount of calculations required for unwrapping is reduced by the algebraic simplification, when the slope unwrap is required, the modifications performed simplifies the process and reduces the amount of memory required.

5.4.4 Gap Correction and Non-Uniformity

In the Matlab version of the LP algorithm, the gap correction was performed by changing the index array to non-integers. This was not an option in the C code, since an individual index array was not used. Instead, intensity values were modified. The intensity values are recalculated using the index and the corrected index values using the linear interpolation function:

$$I'[n] = I[n-1] + (I[n] - I[n-1]) \frac{x' - x[n]}{x[n] - x[n-1]} \quad (5.2)$$

Defining the gap parameter, γ , as:

$$x' = x[n] + \gamma \quad (5.3)$$

and noting that $x[n]$ and $x[n-1]$ are index values, equation 5.2 simplifies to:

$$I'[n] = I[n-1] + (I[n] - I[n-1])\gamma \quad (5.4)$$

Similar to equation 4.11, the gap parameter, γ , is axis specific.

The non-uniformity of the center pixels was approached in a similar manner as in the Matlab iteration. Again, the intensity values at each index were recalculated using a polynomial interpolation function. Equation 4.12 to 4.14 are used to calculate the coefficients. For those equations, x is the corrected indices, and y would be the Intensity values.

5.5 Manual Optimizations

Additional optimizations to the code were performed to further increase memory efficiency and decrease code size. These optimizations were not a functional change to the code, rather than methods to reduce memory usage.

Variable Types and Memory Maps

The Matlab implementation of the LP algorithm is implemented on a personal computer. This simplifies significant portions of the code used. In the Matlab environment, code size and memory usage are not important. Matlab code also does not require variables to be predefined prior to use unlike C code, where using the correct variable type reduces memory usage. Making sure to choose the correct variable type in C code was an important manual optimization. Signing and unsigned variables also helped to lower the memory usage due to a change in bounds for different variable types [Software, 2001].

5. EMBEDDED IMPLEMENTATION

For the C8051F410 microcontroller, the memory bank is structured in such a way that there are different storage classes. The distinct memory types defined in the 8051 means that each storage class has different access speeds. Choosing the correct memory location for each variable is important in optimizing the LP algorithm for the 8051 board. Since, the fastest memory (data space) only contains a small amount of memory space, only short calculations are performed there. For more complex calculations and larger variables, slower memory (xdata space) is used.

Dynamic Memory Allocation

In the Matlab environment creating an array of an unspecified length was possible. In C code, the *calloc* function performed this memory management. However, the *calloc* function, is very resource draining. In the very first instance of the C code, the *calloc* function was used. Subsequent versions replaced this with arrays of a maximum possible length. This requires more variable memory in certain instances but overall saves in execution time.

Complex Variables

The LP algorithm requires the DFT, this inherently means that complex numbers are created. For the Matlab implementation there is no problem, but C code does not have native support for complex variables. The image that is transformed is a real value array, this means that the DFT will be a conjugate symmetric (equation 2.3), the real values will be symmetric, and the imaginary values will be negative symmetric [Mitra, 2006]. Taking this property into account, only half of the values are needed to be calculated. This was used within the selective DFT (Section 5.3.3). Only half of the DFT values are calculated while the other half was generated by mirroring those values, with a negative for complex values, without for real values. The manual implementation of the DFT allowed the real and imaginary parts to

be implemented separately.

In-Place Computations

The Matlab implementation of the LP algorithm did not have the memory requirements from the 8051 microprocessor. To segment the image into its two constituent parts, two images were created from the original shifted image. This created two additional 256 element arrays. In subsequent versions, this operation was changed such that the shifted image was not split into two parts. Two shorter length indexing arrays are used along side the shifted image, rather than the two created images. Windows are created around each peak point based on an assumed peak width. The valley points are also used as a cut off when peaks are near each other. The windowing takes the index for an axis's relevant peaks and relates the indices to the sensor image. This allows the full sensor image to be separated into two images using 2 index arrays. The windowing of the shifted image was a saving in memory due to the variable type used for the arrays. The image arrays being in *floats* and the index arrays being *unsigned chars*. This follows in the gap corrections and non-uniformity corrections where the corrections are only made with respect to the index arrays. From Section 3.4, it was found that windowing was not a detriment to the LP algorithm's precision.

5.6 Final Results

Figure 5.2 shows the final results of the C code. This simulation test was performed within the Matlab environment utilizing the same image data set as Section 4.6.4. The results were able to emulate the Matlab iteration of the algorithm. The mean error for the C code implementation of the LP algorithm was 8.7313×10^{-4} . Comparing to table 4.2 the error in C code was slightly lower, this improvement in precision was not expected.

5. EMBEDDED IMPLEMENTATION

Figure 5.2. *Final results of the C code LP algorithm. This FOV plot displays the peak position estimation error at the input (true) sun vector point.*

Memory Type	Amount used
cdata	34670 bytes
xdata	867 bytes
data	96.2 bytes

Table 5.2: Final memory usage.

The final memory usage is detailed in table 5.2. The resultant LP algorithm was able to successfully be embedded into the C8051F410 microcontroller to compute the shifts from the sensor images.

5.7 Implementation Conclusions

Having tested the LP algorithm to an acceptable precision within the Matlab environment, the LP algorithm was modified and rewritten in to C code to be tested upon the 8051 processor board. The search for a method to perform a DFT within the confines presented by the 8051 board was a significant problem. The solution of writing a DFT code that saves computations by windowing and memory by exploiting symmetries was found. Several other

5.7. Implementation Conclusions

memory management and programming problems were resolved. The C code was able to achieved performance comparable with the LP algorithm in the Matlab environment.

Chapter 6

Conclusions

The goal of this study, was to enhance the performance of sub-pixel peak position estimation and to extend the LP algorithm into two-dimensions. In addition, expanding on the adaptation into two-dimensions was migrating the LP algorithm into an embedded system.

6.1 Summary

Key features and other results observed from the previous chapters are detailed in this section.

6.1.1 One-Dimensional Linear Phase Algorithm Testing

The LP algorithm works well for one-dimensional sun-sensor system. To develop and aid in implementation further testing on the algorithm was performed.

The LP algorithm was tested with delays spanning the whole array. The signal phase was found to be come discontinuous and wrap more frequently for higher delays. This resulted in problems with unwrapping, that is required for the LP algorithm. Several unwrapping methods were studied to resolve

6. CONCLUSIONS

the unwrapping issue.

The local unwrapping method and algebraic simplification method were shown to provide the best results for the majority of the array. However, when shifts were larger than 66% of the array, the local unwrap would be unable to unwrap the phase angles. The local unwrap and the algebraic method were implemented into the the embedded implementation of the LP algorithm based on the performance shown in these results.

The effect of windowing was also looked into. Windowing is an attempt to reduce the amount of calculations required for the DFT. The reduction in the linear section of the signal phase was the main effect upon the LP algorithm. The windowing itself did not affect the peak position estimation performance of the LP algorithm. The results found were used to help with the segmentation of the two-dimensional system, where windowing was required.

6.1.2 Two-Dimensional Algorithm Testing

Two-dimensional testing began with a study on the images produced by a two-dimensional sun-sensor. The images from read from a linear pixel array from a two-axis mask were simulated for the testing. The problem cases were found to be overlap cases, interwoven cases and interfering cases.

Initial implementation of the LP algorithm into two-dimensional used a case determination method as a basis to compare against.

The methodology of the LP algorithm was applied to the two-dimensional case, however, the ambiguity due to the superposition of the images made a direct two-dimensional implementation of the LP methodology not viable. The LP algorithm was then applied to the two-dimensional case by segmenting the sensor image into two one-axis images.

The segmentation of the sensor image was performed in three steps; a coarse estimation of the peaks, peak to slit matching, and the segmentation of peaks. The LP algorithm was then able to estimate peak positions on two

one-dimensional systems.

Using real data for sensor images, several corrections were required to compensate for differences between the ideal sun sensor and the manufactured sun sensor. These corrections included the gap between pixel dies, and the non-uniform pixel surface areas of specific pixels. A modification to the way that interfering cases were approached was also applied to improve the peak position performance.

After the corrections were applied, the peak position estimation performance of the LP algorithm was comparable to the NLSQ algorithm that is currently being used on the SS-411 sun sensor.

6.1.3 Embedded Implementation

To test the LP algorithm on an embedded system, the LP algorithm was required to be transformed into C code. The C code was to be tested on the 8051 development board, which imposed strict memory restrictions on the algorithm.

Several major changes were implemented to allow the LP algorithm to fit within the memory restrictions imposed. The first change was the method in which DFTs were calculated, only specific points of the image are calculated. Tabulating constants was the second significant change. The constants defined the locations where the DFT is to be calculated as well as other constant values.

The polynomial line fit, and the slope unwrapping functions were also modified to optimize the code for memory.

The LP algorithm was able to be implemented on the 8051 board, as well as provide results that were comparable to results obtained within the Matlab test environment.

6.2 Future work

The LP algorithm was optimized for memory to be implemented on the 8051 microcontroller. The focus of optimizations now should be to optimize the code for speed. Using the remaining memory available on the 8051 to increase the speed of the algorithm be an approach to this optimization.

Having the code able to run on the board is useful for processing tests, but does not fully simulate a full sun-sensor cycle. Running the board on-line, while the sun-sensor is capturing images would be a useful test.

The SAIL setup uses an aperture stop, as shown in Section 4.5, to simulate the sun disk. Although the sun-sensor used for capturing images is completely exposed to the incoming light, a test involving a wider or narrower sun disk would be interesting. The effects of a wider or narrower peak upon the LP algorithm would assist in the code's robustness.

Looking at the final results of both the Matlab and C code LP algorithm, the error is not seen to be completely random. The algorithm seems to perform less precisely in concentrated areas. A certain amount of error due to randomness will always be apparent, however, the FOV plot shown in Section 4.6.3 and Section 5.6, do not seem to be random. Examining this unmodeled error would be beneficial to the development of the LP algorithm

Appendix A

Additional Topics

A.1 LP width test

This test was to determine the linear section of the signal phase. A direct relation was not found since the width of the linear section is based upon the window length (Section 3.4) and the noise (Section 2.3). Since it was found that the noise effects are only on the outer extremities of the signal phase, regardless of the window length, the approximate linear section was based upon the window length.

Figure A.1 displays the result. The x-axis of the graph is the divisor of the window length. The associated error resides in the y-axis. The LP width with the least error was found to be a quarter of the window length.

A.2 LP window length test

The effect of the window length upon the resultant error of the LP algorithm was tested. This test used window lengths varying from 32 pixels, 64 pixels, 128 pixels, and a full 256 pixels. The LP width parameter, S was modified according to the results in Section A.1. A smaller window could not be used due to the spacing of the peaks. The test incorporated a 0.03% noise upon

A. ADDITIONAL TOPICS

Figure A.1. *LP width test. The error over a data set using different window lengths (y-axis) relating to the window length divisor (x-axis), where the LP width used was $\frac{\text{window length}}{\text{divisor}}$.*

the simulated images, a typical value. The results are detailed in Figure A.2. The noise was found to affect the array more than the windowing on the LP algorithm. The windowing was found to not directly affect the performance of the LP algorithm.

A.3 SS-411 Pixel Array

The pixel array used on the SS-411 sun-sensor, from Sinclair Interplanetary, is composed of two pixel dies. A magnified view of the separation of the pixel dies is seen in Figure A.3. The center four pixels are seen to be elongated compared to the rest of the pixels on the array.

A.4 Non-Linear Least Squares

The NLSQ algorithm is currently being employed on the SS-411 sun-sensor. The results shown in Figure A.4, are obtained by using the NLSQ algorithm

Figure A.2. *LP window length error. The error over a data set (δ_{err}) against the window length used (x-axis).*

Figure A.3. *Close-up of the Pixel Array of the SS-411 sun-sensor.*

A. ADDITIONAL TOPICS

Figure A.4. *NLSQ results. This FOV plot displays the peak position estimation error at the input (true) sun vector point.*

with the same date set tested upon the LP algorithm in Chapter 4. These results were used to as a benchmark for the LP algorithm to be tested against.

Bibliography

- P. Appel. Attitude estimation from magnetometer and earth-albedo corrected coarse sun sensor measurements. In *4th IAA Symposium on Small Satellites for Earth Observation*, 2003.
- John Enright and Godard. Advanced sun-sensor processing and design for super-resolution performance. In *IEEE Aerospace Conference*, 2006.
- John Enright, Albert Yam, and Chris Li. Parametric processing for two-dimensional digital sun sensors: Algorithms, modeling and testing. *Spacecraft and Rockets*, 45:359–369, 2008.
- H. Foroosh and J.B. Zerubia. Extension of phase-correlation to subpixel registration. *IEEE Transactions on Image Processing*, 11:188–200, 2002.
- Godard. Advanced sun-sensor processing and design for super-resolution performance. Master’s thesis, Ryerson University, 2004.
- Douglas L. Jones. Decimation-in-time (dit) radix-2 fft. Technical report, <http://cnx.org/content/m12016/latest/> , Accessed September 10, 2007, 1992.
- Phil Karn. Floating point complex fft package. Technical report, <http://www.setileague.org/software/karnfft.htm> , Accessed March 29, 2007, 1997.

C. Knapp and G. Carter. The generalized correlation method for estimation of time delay. *IEEE Transactions on Acoustic, Speech, and Signal Processing*, 24, 1976.

Silicon Laboratories. 8051f410/1/2/3. Technical report, Accessed May 4, 2007, 2006.

Sanjit Kumar Mitra. *Digital Signal Processing: A Computer-Based Approach, Third Edition*. McGraw Hill, 2006.

A Oppenheim and R Schafer. *Digital Signal Processing*. Prentice, 1975.

Keil Software. Cx51 compiler. Technical report, Accessed April 25, 2007, 2001.

Texas Advanced Optoelectronic Solutions. Tsl1402r 256 x 1 linear sensor array with hold. Technical report, Accessed September 6, 2007, 2007.