ROTATION ANGLE MEASUREMENT ALGORITHMS FOR REAL-TIME MACHINE VISION APPLICATIONS

by

Sumeyra Ummuhan Demir

A thesis submitted to

the Graduate Institute of Sciences and Engineering

of

Fatih University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Electronics Engineering

2007

Istanbul TURKEY

APPROVAL PAGE

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

. . . .

Prof. Dr. Muhammet Koksal
Head of Department

This is to certify that I have read this thesis and that in my opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

. . . .

Assoc. Prof. Dr. Onur Toker
Supervisor

Examining Committee Members	
Prof. Dr. Kemal Fidanboylu	. . . .
Assoc. Prof. Dr. Halil Rıdvan Öz	. . . .
Assoc. Prof. Dr. Onur Toker	. . . .

It is approved that this thesis has been written in compliance with the formatting rules laid down by the Graduate Institute of Sciences and Engineering.

. . . .

Assist. Prof. Dr. Nurullah ARSLAN
Director

01.08.2007

ROTATION ANGLE MEASUREMENT ALGORITHMS FOR REAL-TIME MACHINE VISION APPLICATIONS

Sumeyra Ummuhan Demir

M S, Thesis-Electronics Engineering
August 6, 2007

Supervisor: Assoc. Prof. Dr. Onur Toker

ABSTRACT

The purpose of this study can be summarized as rotation estimation algorithms. The algorithms are implemented to estimate the rotation angle of continuously rotating and translating fabrics which is useful for textile industry, especially for automatic weft-straightening systems. The methods studied are mainly based on FFT and Autocorrelation computations. Number Theoretic Transform is also discussed which allows the FFT calculations to be done in modular arithmetic.

Keywords: rotation, machine vision, image, weft-straightening

ROTATION ANGLE MEASUREMENT ALGORITHMS FOR REAL-TIME MACHINE VISION APPLICATIONS

Sumeyra Ummuhan Demir

Yuksek Lisans Tezi-Elektronik Muhendisligi
August 6, 2007

Tez Yoneticisi: Assoc. Prof. Dr. Onur Toker

ÖZ

Bu çalışmanın temel amacı dönüş açısı hesaplama algoritmaları olarak özetlenebilir. Üzerinde çalışılan algoritmalar sürekli dönen ve kayan kumaş parçalarının dönüş açısını hesaplamakta kullanılmak üzere tasarlandı. Tekstil endustrisinde ve bilhassa otomatik kumaş düzeltme sistemlerinde gerçekleştirilebilir. FFT ve ACF hesaplamaları içeren metotlar kullanıldı. FFT hesaplamalarını modüler aritmatikle hesaplamaya izin veren Teorik Sayı Dönüşümü yaklaşımı da denenen metotlar arasındadır.

Anahtar Kelimeler: dönüş açısı, makine görüş uygulamaları, resim, atkı düzeltme uygulamaları

DEDICATION

To my mom and dad.

ACKNOWLEDGEMENTS

I would like to thank to my supervisor Assoc. Prof. Dr. Onur TOKER for his patient attention and guiding me throughout my first academical study and writing the thesis.

My special thanks go to my friend, Ayşe Nur Taşlıpınar for her great hospitality in my awful time.

I express appreciation to Zafer Çatmakaş, my co-worker for his understanding and gentle attitude during my study.

At last I want to thank to EE 202 & ELM 202 students for their sensibility.

TABLE OF CONTENTS

ABSTRACT . . . iii

ÖZET . . . iv

DEDICATION . . . v

ACKNOWLEDGEMENTS . . . vi

LIST OF FIGURES . . . ix

LIST OF TABLES . . . xi

LIST OF SYMBOLS/ABBREVIATIONS . . . xii

CHAPTER 1.INTRODUCTION . . . 1

- 1.1. Introduction to Image Rotation Detection . . . 1
- 1.2. Weft-Straightening Problem . . . 1
- 1.3. Proposed Approaches . . . 3

CHAPTER 2.PREVIOUS WORK . . . 5

CHAPTER 3.POLAR TRANSFORM METHOD . . . 13

- 3.1. Autocorrelation of Images . . . 13
- 3.2. Polar Transform Approach . . . 16
- 3.3. Determining the Rotation Angle . . . 18

CHAPTER 4.FGT-CONSTELLATION METHOD . . . 22

- 4.1. Number Theoretic Transform . . . 26
  - 4.1.1. Prime Numbers Algorithm . . . 27
- 4.2. Divide and Conquer Algorithm of FFT . . . 35

CHAPTER 5.EXPERIMENTAL RESULTS AND PERFORMANCE ANALYSIS . 40

CHAPTER 6.CONCLUSIONS . . . 45

APPENDIX A: MATLAB CODE OF POLAR TRANSFORM APPROACH . . . 47

APPENDIX B: C CODE of COMPUTATION ACF, (FFT computed via NTT) . 50

APPENDIX C: C CODE FOR COMPUTATION of FFT via DIF TECHNIQUE . 57
REFERENCES 61

LIST OF FIGURES

Figure 1.1 Seiren Electronics Weft Straightening Machine 2

Figure 2.1 Sample Text Images 7

Figure 2.2 Thresholded Fourier Transforms of images in Figure **2.1** 7

Figure 3.1 A texture image and its translated and 25° rotated version 15

Figure 3.2 Autocorrelation Function of the Images 15

Figure 3.3 Gamma Functions of the original and rotated ACF 16

Figure 3.4 Polar Transform of ACF in Figure 2.2 17

Figure 3.5 Phase Angle of the Polar Transforms' Ratio 18

Figure 3.6 1.2°, Primarily Defined Range 19

Figure 3.7 18°, Primarily Defined Range 20

Figure 3.8 18°, Extended Range 21

Figure 3.9 1.2°, Extended Range 21

Figure 4.1 Thresholded ACF of Figure **3.2** 23

Figure 4.2 First stage in DIF of N=8 38

Figure 4.3 Flow Graph of Complete DIF with N=8 38

Figure 5.1 Error Graphic between $-25°$ - $25°$ for different Q_e's 42

Figure 5.2 Estimation Error Graphic for rotation angle between $-30°$ - $30°$ . . . 43

LIST OF TABLES

Table 2.1 MATLAB algorithm of Rotation Estimation by Hilbert Transform . . 9

Table 3.1 MATLAB algorithm of ACF using FFT 15

Table 3.2 MATLAB algorithm for Polar Transform 17

Table 4.1 Logarithmic Thresholding Algorithm in C 24

Table 4.2 Finding The Brightest Spot Nearest to Origin Algorithm in C 25

Table 4.3 Determination of the Displacement of the Rotated Spot Algorithm in C 25

Table 4.4 Primeness Test Algorithm in C 30

Table 4.5 Primitive Root Algorithm in C 32

Table 4.6 Modular FFT Algorithm in C 34

Table 4.7 Pseudocode to implement DIF Fast Fourier Transform. (a) FFT (b) Bit Reversal Routine 39

Table 5.1 Error values for different angles and Q_e 41

Table 5.2 Computation Time for Different Size Images (Both use NTT) 44

LIST OF SYMBOLS/ABBREVIATIONS

F(l)	Fourier Transform of the given data
$\sum$	Summation of all inside values from lower limit of the symbol to the upper limit
$R(x)$	Autocorrelation of the given data
W_N	The exponential term $\exp^{2\pi/N}$
N	Size of the image
p	Prime number
r	Primitive root of prime number p
G	Generator used to replace $\exp^{2\pi/N}$

FT	Fourier Transform
FFT	Fast Fourier Transform
DFT	Discrete Fourier Transform
ACF	Auto Correlation Function
FGT	Fast Galois transform
DIF	Decimation In Frequency
DIT	Decimation In Time

CHAPTER 1

INTRODUCTION

1.1. Introduction to Image Rotation Detection

With the increase in the use of digital cameras, image processing and machine vision applications have become popular areas to study. Digital images are used in a wide area from daily life applications to scientific studies. The captured images are generally afflicted with scaling, translation and rotation problems. The detection of translation or rotation allows to recover such problems. To detect the distortion, some features of images that change with translation and/or rotation can be calculated like Zernike Moments[1] or some transformations can be applied such as autocorrelation, fourier and cosine transforms. For this work, the images are taken from a weft-straightening system and the estimated information is sent to the system for further steps. For a weft-straightening system, the existing events are translation and rotation. The rotation is need to be estimated independently from translation. The autocorrelation and fourier transform are the calculation methods to estimate the rotation angle of the weft.

1.2. Weft-Straightening Problem

Textile industry is a daily growing industry with high technological machines. Weft-Straightening is one of the common problems in the industry. Before finishing of fabrics for the manufacture, straightening the weft of textile is an important stage. Manual process is boring, repetitive, tiring and time consuming. It is very difficult for the human eye to detect weft distortions in fabric running continuously. Advanced technology reduced the price of computer systems and digital cameras which results low-cost machine vision systems for automation of weft straightening. The weft-straightening process deals with two main problems: distortion and rotation. This study is concerned with the estimation of rotation angle of the running fabric.

The history of automatic weft straightening goes back more than 40 years. Generally, there are two common automatic weft-straightening principles used today: automatic straightening by means of mechanics; and automatic straightening by means of electromechanics. Electromechanics is the most common method employed today. Weft geometry is automatically detected using a mechanical or electrical sensor. That information is then transformed into a signal that displaces rollers, making the correction as necessary. With the increased complexity of fabric types, and demand for faster and more accurate fabric processing, use of advanced machine vision systems with CCD cameras and embedded systems started to appear in commercial products. Erhardt+Leimer's ElStraight is a well-known example [2]. For this work, we have used a locally developed system : *Fatih University Smart Camera (FU-SmartCam).*

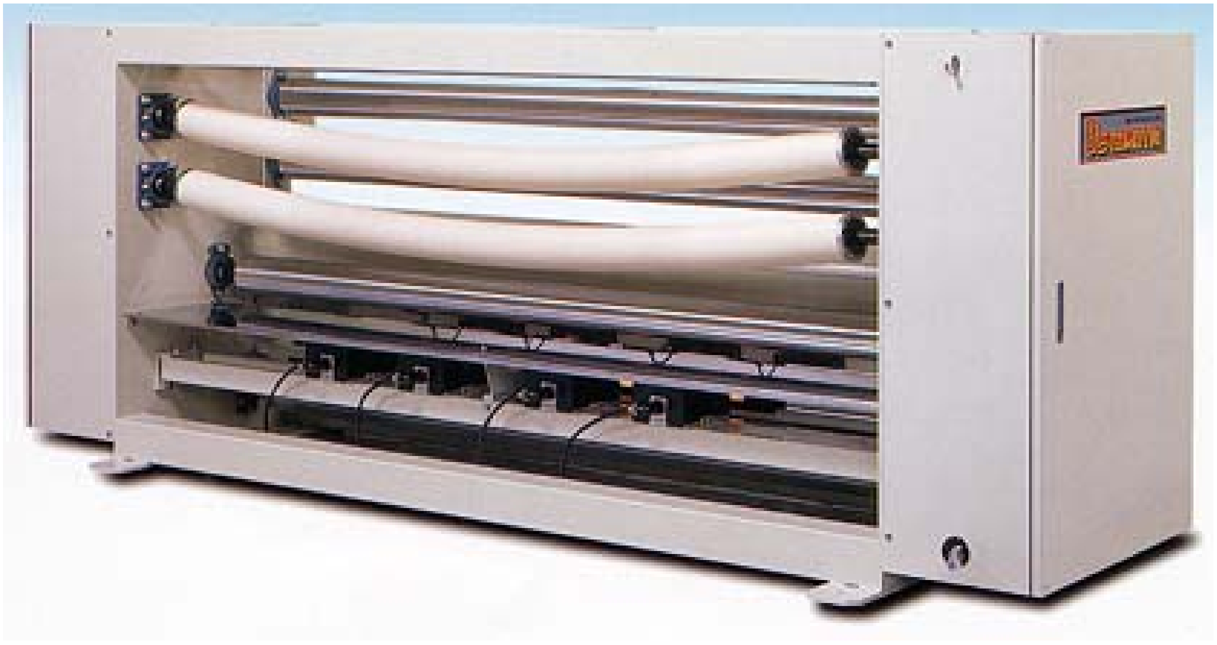


Figure 1.1 Seiren Electronics Weft Straightening Machine

1.3. Proposed Approaches

For this project, the aim is estimating the rotation independently from the transformation in a real-time manner. Third chapter describes a way using the Autocorrelation Function to leave out the translation effect. To define the calculated function in Polar Coordinate System is a reasonable way since the rotation reduces to a shift in one direction. The rest is to determine the relation between the angle and the distance of the shift.

However the calculation includes floating numbers and this increases the process time. A way that is computationally efficient than floating calculations can be replaced by doing all calculations with integer numbers. Third chapter introduces a calculation via modular arithmetic. The Fourier Transform of the images are calculated using Number Theoretic Transform. The phase information of the FT is used to determine the rotation angle.

For the first part of the project MATLAB is used so the algorithms given are all for MATLAB. MATLAB is a tool used for numerical computations with matrices and vectors. The name is shorted from "matrix laboratory". Cleve Moler invented MATLAB in the late 1970s when he was the chairman of the computer science department at the University of New Mexico. The first purpose of the tool was to give the students access to LINPACK and EISPACK without having to learn Fortran. It is available to implement most of the classical algorithms with a few comments using MATLAB. This results shorter and faster codes and calculations with respect to other high level programming languages. Since it is easy to implement the algorithms, MATLAB is used to compare the methods and to make a decision of an approach to use. During the implementation of MATLAB codes, the images are not taken from a camera. The input images and the rotation is manually determined.

The operating system for the project will be **Linux** and the programming language will be **C**. After deciding a method with MATLAB algorithms, the decided one is implemented to C and compiled by gcc-Linux C compiler. The Linux operating system allows the user to control the whole system much more than Windows does. The unnecessary hardware can be removed, the running processes can be minimized and moreover the whole system is under user's control. To be able to set an efficient system with desired specifications, Linux and C couple is chosen. The programming language C can be compiled with Linux without any extra software. C is a preferred language for academic studies because of its small size, extensive use of function calls, loose typing, structured language, low level (bit wise) programming and pointer implementation which allows extensive use of pointers for memory, array, structures and functions characteristics. So for the third chapter, all algorithms are implemented in C. And the images are captured from a camera in a real-time manner while the texture is rotating.

At last, the performance analysis is done by discussing the efficiency of both methods and comparing them in terms of process time especially.

CHAPTER 2

PREVIOUS WORK

Detecting the relationship between two images from the view of translation, scaling and rotation factors is a common problem in Machine Vision. The application areas of such a detection varies widely from military applications to non-military applications. Change detection, image fusion, depth perception, motion estimation, biological pattern recognition [3], object recognition [4], target localization and target recognition [5], [6], [7] are some of the examples.

The process of finding relative orientation and translation that aligns two images of the same objects is called image registration [8]. Registration process involves estimation of rotation, translation and scaling. However most of the registration techniques focus on the rotation invariance features to assign if two images captured from different viewpoints belong to the same object [9], [10]. They generally don't discuss the determination of important distortions such as angle of rotation.

The automatic registration algorithms are classified simply in four groups by Reddy and Chatterji [11]: 1.algorithms that directly use image pixel values; 2. algorithms that operate in the frequency domain (e.g., the Fast Fourier Transform (FFT) based approach used here); 3. algorithms that use low-level features such as edges and corners; and 4. algorithms that use high-level features such as identified objects, or relations between features.

The frequency domain algorithms generally use the phase correlation technique [12]. For phase correlation technique, the main approach is Fourier Transform. The discrete form of Fourier Transform(DFT) for a 1-D image with size N, can be calculated as follows :

$$F_l = \sum_{k=0}^{N-1} f_k * e^{\frac{-i2\pi kl}{N}} \tag{2.1}$$

where f_k is the digital image and F_l is value of the sampled Fourier Transform at frequency equals l and l = 0,...N - 1 [13].

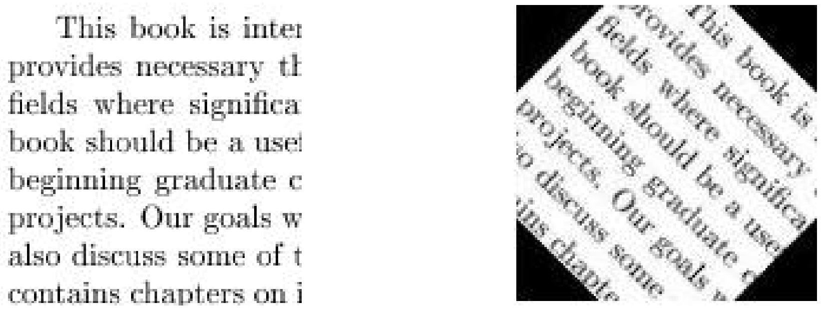
An important detail about FT is that f_k is the digital image and is real, but F_l is the FT and is, in general, complex, because of the exponential part of the formula. Generally, F_l is represented by its magnitude and phase, where:

$$F_{l,magnitude} = \sqrt{F_{l,real}^2 + F_{l,imaginary}^2} \tag{2.2}$$

$$F_{l,phase} = \arctan \frac{F_{l,imaginary}}{F_{l,real}} \tag{2.3}$$

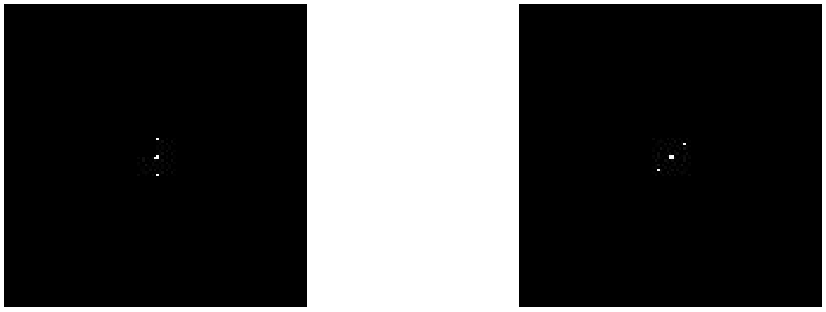
The amount information of a certain frequency component exists in the magnitude and and the position information of the component in the image exists in the phase. For any displacement problem like rotation, we can focus on the phase information of the FT. In general, rotation of the image results in equivalent rotation of its FT. By comparing the phase information of the original image's FT and the rotated one's, the rotation value can be estimated. The relevant pictures that show the rotation of FT with a rotation of an image can be seen in Figure **2.1** and **2.2** .

FFT-based approaches for image registration have been studied for many years. First use of Fourier transform is to determine rotation as well as shift in 1987 [14]. The phase difference method is introduced by Cideciyan in 1992 [15]. The mentioned study of


This book is inte
provides necessary tl
fields where significa
book should be a use
beginning graduate c
projects. Our goals w
also discuss some of t
contains chapters on i

(a) Original Text Image (b) Rotated Text Image

Figure 2.1 Sample Text Images



(a) Thresholded FT of Original (b) Thresholded FT of Rotated

Figure 2.2 Thresholded Fourier Transforms of images in Figure **2.1**

Reddy and Chatterji also involves an improvement on the algorithm of Morandi [14] by greatly reducing the number of transformations needed [11].

Beside the Fourier Transform, there are different transform techniques derived from Fourier Transform such as Fourier-Mellon Transform and Hilbert Transform. The Fourier-Mellon transform has been introduced to register images that are misaligned due to translation, rotation, and scale [11], [16], [17], [18]. This method applies a Fourier transform to images to recover translation. Then a log-polar transformation is applied to the magnitude spectrum and the rotation and scale is recovered by using phase correlation in the log-polar space . This is an efficient method but only works well for images with dominant features in the power spectrum. In addition, as the overlap amount of the images diminishes, the power spectra will alter in a remarkable way and no longer correlate. A spatial domain log-polar mapping has also been introduced in 2000 [19].

To be able to use the Hilbert Transform method, a complex function is generated with the real part being the rotated image and the imaginary part being its 90^o rotated version [20]. Fourier Transform of this complex function is calculated. For different rotation angles defined by the user, the original image is being rotated and a complex function is generated for each angle value. The FT of all complex functions are correlated with the rotated image's FT. The rotation angle that has maximum correlation coefficient gives the answer. The rotation angle range may differ for each application. For example for this study the weft is assumed to be rotated between $-30°$ and $30°$. The precision may be increased by decreasing the value of steps. The exact rotation angle value can be estimated by small enough angle steps. Besides, the algorithm works properly for an occlusion of $0\% - 20\%$ of the target image for the same image case. However this method is not fast enough for a real-time application. It has been tried as a MATLAB algorithm and for an image rotated $1.6°$, the program can estimate the exact angle in 9 seconds with a range $0° - 4°$. The MATLAB algorithm for the mentioned calculations is given in Table **2.1** .

Table 2.1 MATLAB algorithm of Rotation Estimation by Hilbert Transform

```
ran=0.2;
P=imread('kumas2.bmp'); % Read the image
Ip=P(:,:,1); % The color information is not taken
Ic=imrotate(Ip,1.6,'bilinear','crop'); % Rotation with 1.6 degree
I90=imrotate(Ic,90,'bilinear','crop');
myim=i*double(I90);
myre=double(Ic);
myco=myre+myim;
ref=fft(myco);
[M,N]=size(ref);
ref2=[];
mycoef=[];
sir=1;
for k=0:ran:4
    I2=imrotate(Ip,k,'bilinear','crop');
    I290=imrotate(Ip,90,'bilinear','crop');
    myim2=i*double(I90);
    myre2=double(I2);
    myco2=myre2+myim2;
    ref2(1:M,1:N,sir)=fft(myco2);
    mycoef(1:2,1:2,sir)=corrcoef(ref,ref2(:,:,sir));
    sir=sir+1;
end
myc=mycoef(1,2,:);
[r,c]=max(myc);
ang=(c-1)*ran
```

One of the approaches to calculate Fourier Transform is Number Theoretic Transform which utilizes Number Theory facts, specifically Elementary Number Theory facts. Number theory is a part of mathematics that is concerned with the numbers in general, and the integers in particular, and the properties and problems of them.

The aim of number theory is to uncover the many deep and subtle relationships between different sorts of numbers. There are square numbers and prime numbers and odd numbers and perfect numbers (but no square-prime numbers) [21].

The main areas of number theory are listed below:

Elementary NT: The studied subjects of elementary number theory are divisibility among integers -the division "algorithm", the Euclidean algorithm (and thus the existence of greatest common divisors), elementary properties of primes (the unique factorization theorem, the infinitude of primes), residue classes, congruences(and the structure of the sets Z/nZ as commutative rings), including Fermat's little theorem and Euler's theorem extending it, the quadratic reciprocity law, representation of numbers by forms, diophantine equations, continued fraction approximations and sieves. It is one of the best known areas of number theory because of its charm and general accessibility. *Cryptography* is highly related with factoring large integers and primeness tests.

Analytic NT: In analytic number theory an arithmetical phenomenon is represented by a related function, generally an analytic function of a complex variable. It involves the study of the Riemann zeta function and other similar functions such as Dirichlet series. The zeta function may be defined on half the complex plane as the sum $1+1/2^s+1/3^s+1/4^s+...$; its connection with number theory results from its factorization as a product $\prod(1-1/p^s)^{-1}$, the product taken over all primes p.

Algebraic NT: Algebraic number theory deals with fields of algebraic numbers, that is with numbers which are roots of a polynomial equation with rational coefficients. Questions in algebraic number theory often require tools of Galois theory; that material is mostly a part of Field Theory. The algebraic structure of the ring of integers is similar to that of other commutative rings such as rings of polynomials.

Probabilistic NT: In probabilistic number theory statistical limit theorems are established in problems involving "almost independent" random variables [22], [23].

Number Theoretic Transform is a Discrete Fourier Transform defined over a finite number field or ring. It requires basic *Elementary Number Theory* and Fast Fourier Transform knowledge. Since NTTs are defined over finite rings or fields, all arithmetic must be carried out modulo M, denoted by $(.)M$. The modulus M can be a prime $M = p$, a power of a prime $M = p^m$ or a composite number $M = \prod_{i=1}^{K} M_i$, resulting in NTT pairs defined over a Galois Field (GF), an extension field $GF(p^m)$ or a finite ring R_M [24].

There are many practical applications of NTT some of which are classified by Gudvangen:

Convolution and correlation: This is perhaps the most straightforward application of finite field/ring transforms. The prospect of reduced computational complexity is often of greater importance than error-free computation for this type of applications. In order to achieve a reduction in complexity, NTTs with highly composite N and with roots of unity G that leads to simple multiplications with powers of G must be employed. One-dimensional FIR filters [25], [26] and image filtering [27], [28] are areas where NTTs have been successfully employed.

This study is also an example of such a type of application. Toivonen has also used NTT to compute convolution to estimate block motion [29].

Multi-precision multiplication is another application area proposed by Schönhage and Strassen as an efficient method for multiplying very large numbers [30].

Matrix multiplication [31], [32], [33], Superfast Toplitz system solvers [35], [34], Cyclic Convolution Code (CRC) Coding and decoding which play a pivotal role in error detection and correction [36], [37] and computation of Prime Factor Algorithm (PFA) [38] are the other applications of NTTs mentioned by Gudvangen [24].

The two principal advantages to be gained from the use of NTTs are error-free computation and sometimes a significant reduction in computational complexity.

For the feature based methods, the first step is to find some features that suits with the purpose of the project. If the purpose is to determine whether two different pictures belong to the same object, like the object recognition applications, the features should be translation, rotation and even scaling invariant. To detect the translation or rotation amount, features varying with translation or rotation are being studied.

The features used for image registration are classified by Thomas Deselaers and Daniel Keysers in 8 groups [39] :

1. Image Features
2. Color Histograms
3. Invariant Features
4. Invariant Fourier Mellin Features
5. Gabor Features
6. Tamura Texture Features
7. Local Features
8. Region-based Features

These methods detect and match a set of features. Using the matched points the geometric transformation can be found through some pattern classification methods, i.e. support vector machine [40]. These feature based methods work for a large class of transformations but they are computationally costly.

No one technique is suitable for all situations and typically the best method is chosen depending on the problem.

CHAPTER 3

POLAR TRANSFORM METHOD

3.1. Autocorrelation of Images

Autocorrelation is one of the most common functions in engineering applications. The Autocorrelation Function measures the correlation of an image with its displaced version in all possible directions. The main feature of the function is not being affected by translation. The formulation can be given as :

$$\mathbb{R}(x) = \int_{-\infty}^{\infty} f(x\prime) * f(x + x\prime) dx\prime \tag{3.1}$$

where f(x) is the one-dimensional image, and x' is the integration variable. A two-dimensional formula can be described easily by double integration, for x and y directions.

For digital images, the discrete form of the Autocorrelation includes summation instead of integration :

$$\mathbb{R}(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) * f(x - u, y - v) \tag{3.2}$$

The dimensions of the original image and its Autocorrelation Function are the same. Using Fourier transforms to calculate the ACF is reasonable since a correlation (convolution) in spatial domain "reduces" to a multiplication in frequency domain and FFT methods are effective to compute FT. The formulation of the FFT for a 2-D image f(x,y) is defined by

$$F(k,l) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y) * W_M^{xk} * W_N^{yl} \tag{3.3}$$

where $W_M = \exp^{-j2\pi/M}$. Based on the Wiener- Khintchine's theorem, the autocorrelation image is calculated with FFT by [41]

$$R(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} |F(k,l)|^2 * W_M^{-uk} * W_N^{-vl} \tag{3.4}$$

which means in fact; the product of the FT of an image with its complex conjugate is the FT of the ACF.

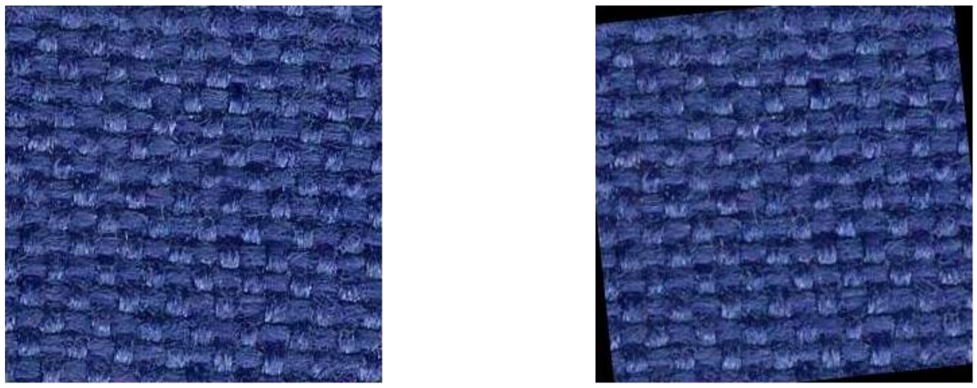
$$F\{R(u,v)\} = F\{x,y\} * F^{*}\{x,y\} \tag{3.5}$$

The images used in this project are continuously translating and rotating. If f(x,y)is taken as the original texture image function, the texture image rotated by α and translated by $(\Delta x, \Delta y)$ will be defined by

$$f_{rot,trans}(x,y) = f((x\cos\alpha - y\sin\alpha) - \Delta x, (x\sin\alpha + y\cos\alpha) - \Delta y) \tag{3.6}$$

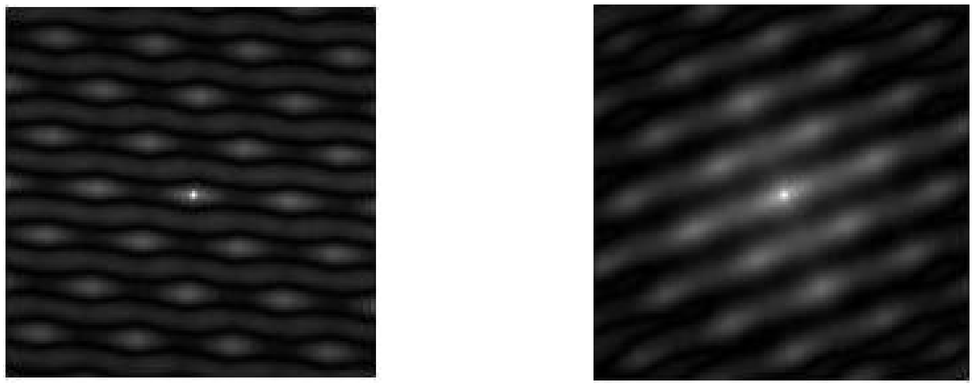
The original texture image and its translated and 25° rotated version used for this study can be seen in Figure 2.1. From the properties of the FT, the translation effect $(\Delta x, \Delta y)$ is eliminated at autocorrelation images. Using ACF to leave out the translation effect is a common process at image processing [41].

Further process is focused around the origin of the image in order to neglect the distorted edge sides. The displacement of the brightest points around the origin can be used to estimate the rotation. Such a focusing process will also reduce the calculation size. The required resizing process can be done manually or the resizing values can be



(a) Original Texture (b) Translated&Rotated

Figure 3.1 A texture image and its translated and 25° rotated version



(a) ACF of the Original Image (b) ACF of Rotated

Figure 3.2 Autocorrelation Function of the Images

defined. By drawing a rectangular that includes the brightest point around the origin is not efficient because it results a serious rise in the operation time. So the quarter part around the origin of the autocorrelated image is taken into account. The ACF of images in Figure 2.1 is cropped, the parts used for the further calculations are in Figure 2.2. Since the digital images are not defined with complex numbers, the multiplication of the

Table 3.1 MATLAB algorithm of ACF using FFT

```
function aI=myxcorr2(I)
aI=fftshift(ifft2(abs(fft2(I-mean(mean(I)))).^2));
```

function and its complex conjugate becomes the square of the image. 'fft2' is a FFT function for 2-D arrays. 'ifft2' is the inverse Fast Fourier Transform. 'fftshift' is used to center the resulted array.

After computing the ACF, next step is normalizing. To make the bright points prominent, Gamma Function is a good solution. Normalized autocorrelation images are multiplied with a γ constant to obtain the Gamma Function. For $\gamma = 0.7$, the resulting ACF are in Figure 2.3. The estimation of the pure rotation is going to be done in Polar Coordinates wherefore a rotation in Cartesian Coordinates results a translation in Polar Coordinates.

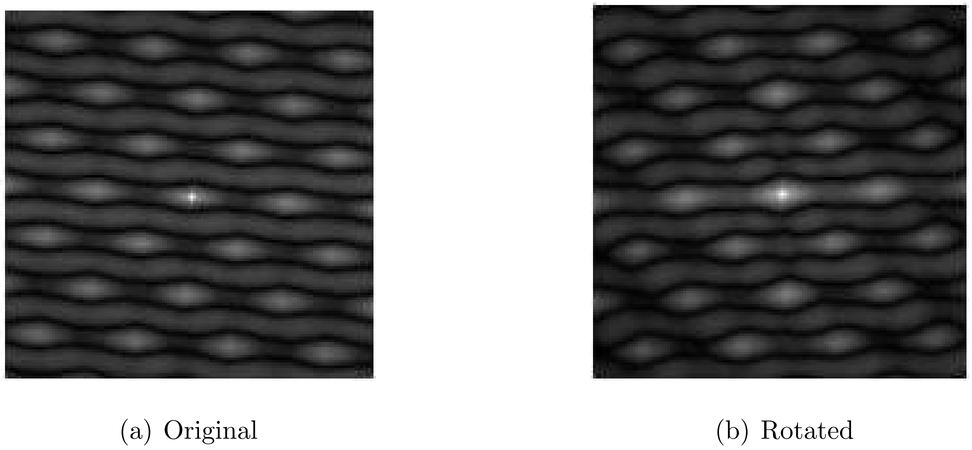
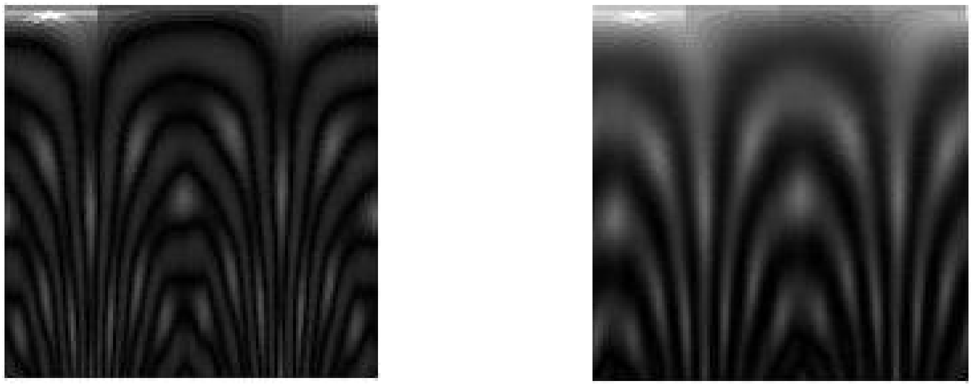

(a) Original (b) Rotated

Figure 3.3 Gamma Functions of the original and rotated ACF

3.2. Polar Transform Approach

Polar Coordinate System represents each point with radius and angle (ρ, Θ). The Polar Coordinate System is preferable when two different points are related with each other specifically by angle and distance values. The Cartesian Coordinate System defines such a relation in a triangular form only. It is easy to see that a pure rotation around the origin in the Cartesian space corresponds to translation in the Θ direction in the polar space. The translation in the polar transform of the ACF can be viewed in Figure 2.4. If the original image's ACF is $R_{t,p}(\rho, \Theta)$ in polar coordinates, the rotated image's ACF will be $R_{t-rotated,p}(\rho, \Theta + \Delta\theta)$. For N*N matrix $R_t(x, y)$, $R_{t,p}(k, l)$ is P*Q.

$$R_{t,p}(k,l) = R_t(\frac{N}{2} + k * s_r * \cos(l * s_\Theta), \frac{N}{2} + k * s_r * \sin(l * s_\Theta)), \tag{3.7}$$
$$\text{where } s_r = \frac{N/2}{P},\ s_\Theta = \frac{2 * \pi}{Q},\ k = 1, ..., P,\ l = 1, ..., Q$$



(a) Original (b) Translated&Rotated

Figure 3.4 Polar Transform of ACF in Figure 2.2

Now the problem is reduced to the estimation of the shift in the y direction of $\mathbb{R}_{t,p}(\rho, \Theta)$. A simple correlation analysis can be used to find the value of d for which the correlation between $\mathbb{R}_{t,p}(\rho, \Theta)$ and $\mathbb{R}_{t-Rotated,p}(\rho, \Theta + \Delta\theta)$ is maximum.

Table 3.2 MATLAB algorithm for Polar Transform

```
function pI=convPolar(I)
[P,Q]=size(I);
pI=zeros(P,Q);
sr=(min([P,Q])/2 - 1)/P;
st=2*pi/Q;
for k=1:P
    for l=1:Q
        r=k*sr;
        t=l*st;
        x=round(P/2+r*cos(t));
        y=round(Q/2+r*sin(t));
        π(k,l)=I(x,y);
    end
end
```

3.3. Determining the Rotation Angle

The difference between the original texture and the rotated one can be observed with the ratio of the polar transformed functions. The ratio is calculated by dividing the rotated function to the original one. The phase angle of the ratio can be viewed in Figure 2.5. Notice that the angle values are in radian.

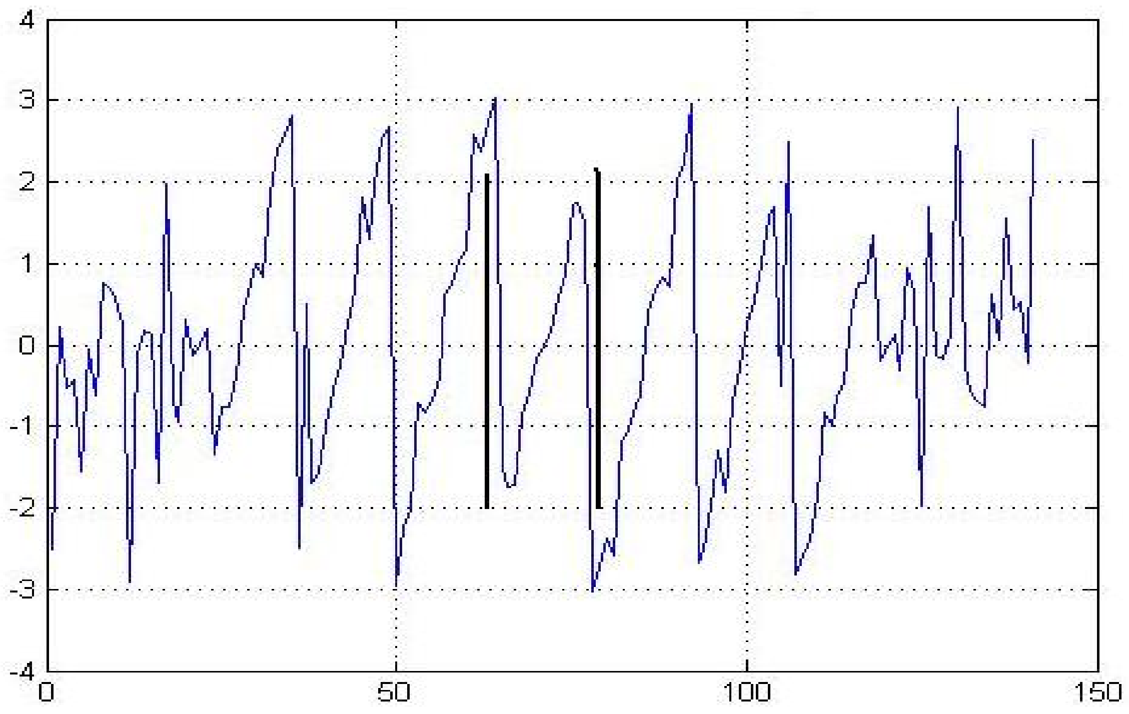


Figure 3.5 Phase Angle of the Polar Transforms' Ratio

To determine the exact rotation angle, a range on the phase angle line should be defined that is linear. The bold vertical lines show this linear range. After taking the average value of this part the resulting rotation angle is got in radian.The resulting angle value is converted to degree by multiplying with 180 and dividing to π. For the Figure 2.5 the image has been rotated 25° manually and the output of the program is 25.4687°. The estimation error is 0.4687. This error seems to be reasonable since it is less than one degree.

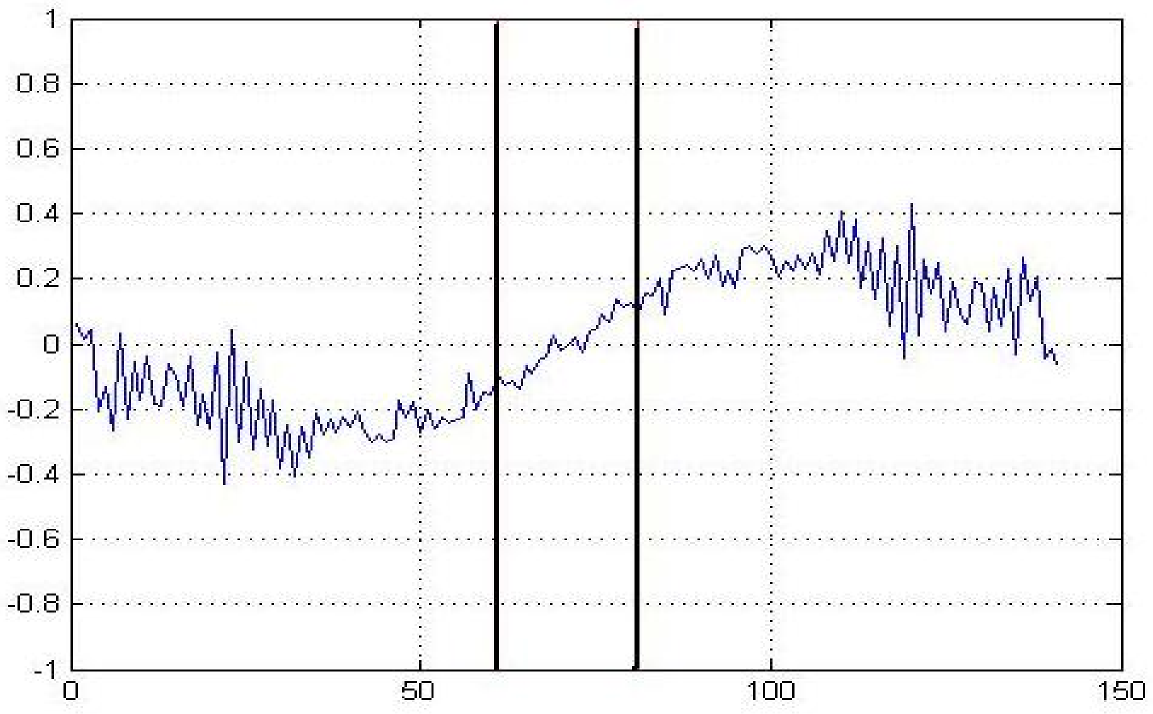


Figure 3.6 1.2°, Primarily Defined Range

However the error does not remain under one for different angle values. When the phase ratio graphic is checked, the reason of such an increase can be related with the definition of the linear range. For smaller rotation angle values, the defined range does not include a complete part from a maximum value to minimum. It loses the linearity, as a result higher estimation errors occur.

The range gets smoother for bigger angle values. For different range definitions, the calculation area can be viewed from Figure 2.6 and 2.7 for 1.2° and 18° degrees.

First, extending the range seems to be a good solution so that a whole part from peak to bottom can be involved for 1.2°, Figure **3.9** . However it causes an error increment for higher angle values, Figure **3.8** . The extended range includes more than one peaks and bottoms that results a confusion. Trying to find a reasonable value which suits with all angle values is meaningless. Instead, a dynamic range is used that if the angle is smaller the range will be extended. This is implemented by checking the error value. For a P*Q

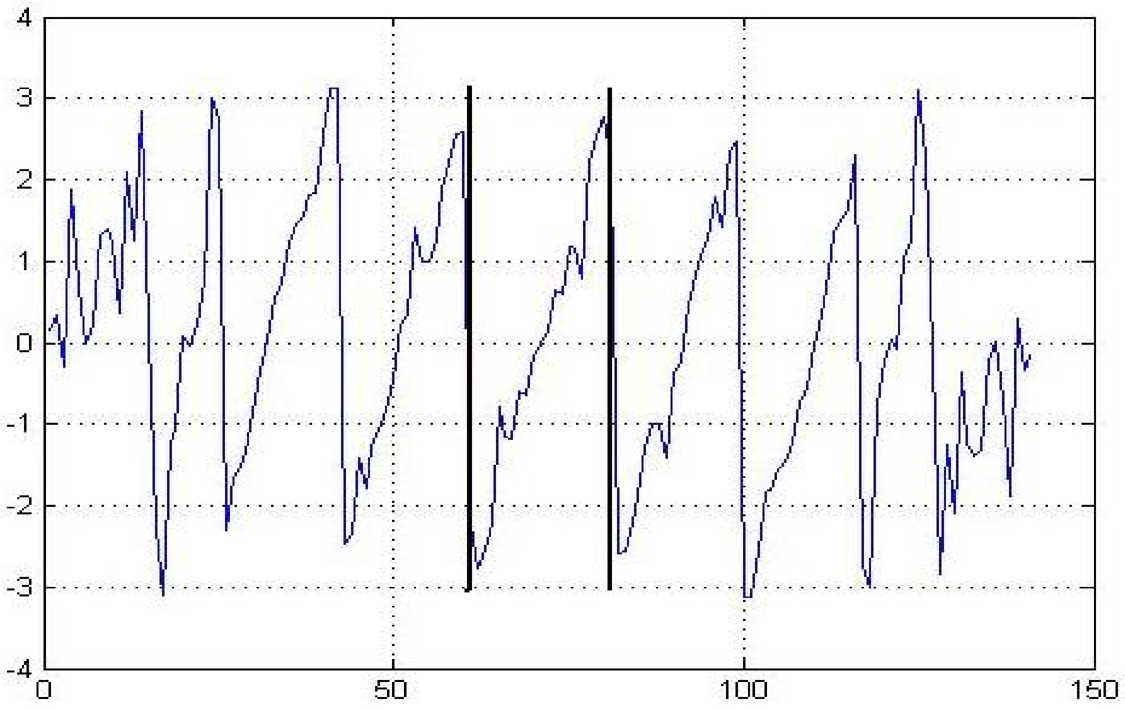


Figure 3.7 18°, Primarily Defined Range

size image, the linear range is defined from $Q/2 - Q_e$ to $Q/2 + Q_e$. After testing several values, two Q_e values have been defined, one is the default value and the second is for the range that the default one gives high errors .

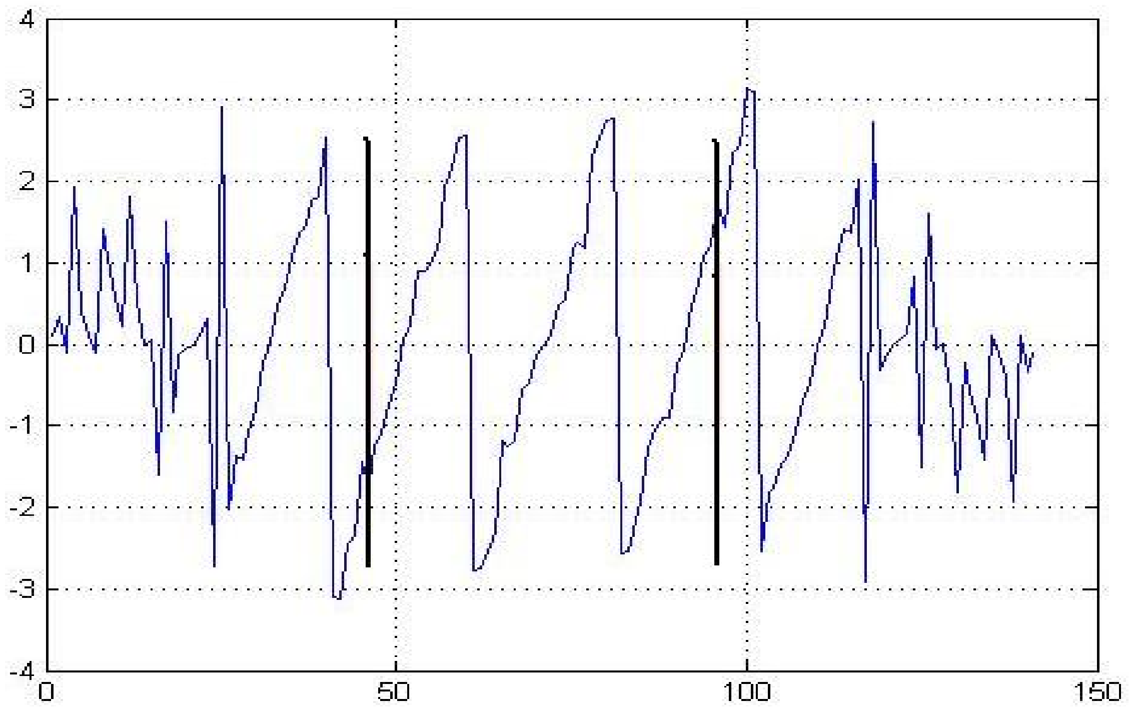


Figure 3.8 18°, Extended Range

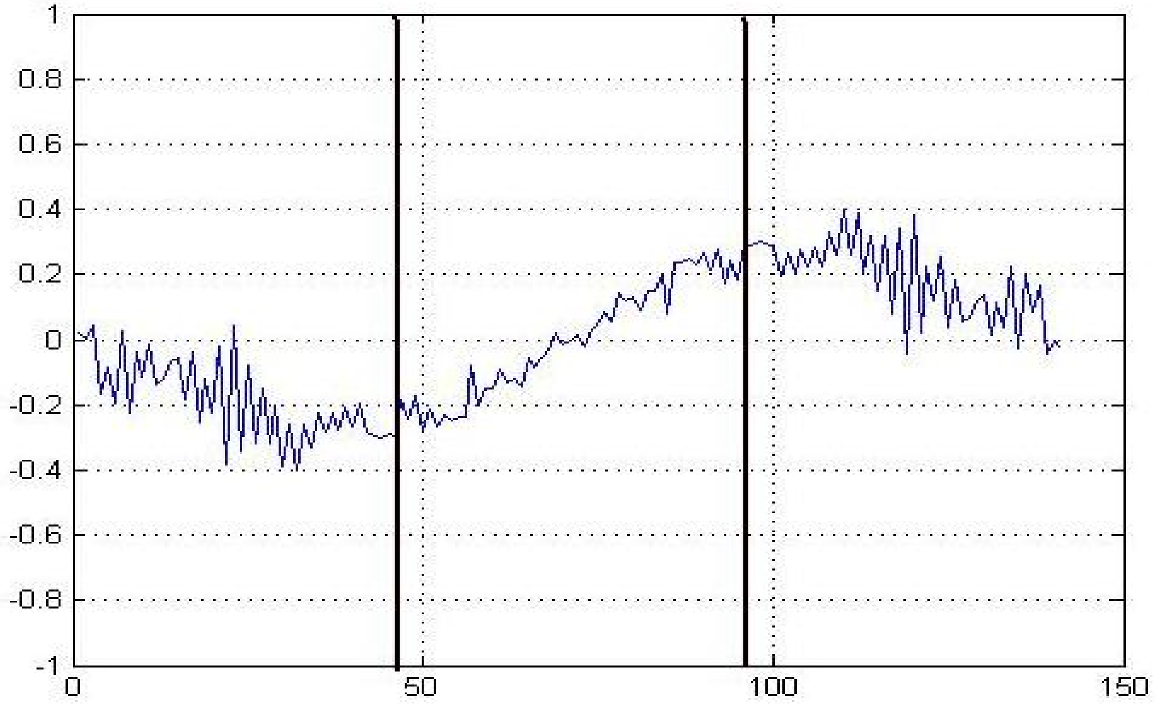


Figure 3.9 1.2°, Extended Range

CHAPTER 4

FGT-CONSTELLATION METHOD

Using certain number-theoretical transforms, integer convolution can be calculated easily. One of those transforms used for this project is Fast Galois Transform or FGT. The transform is implemented through some prime number theorems. Furthermore split-radix FFT algorithms can be applied to the FGT to improve the computation [42].

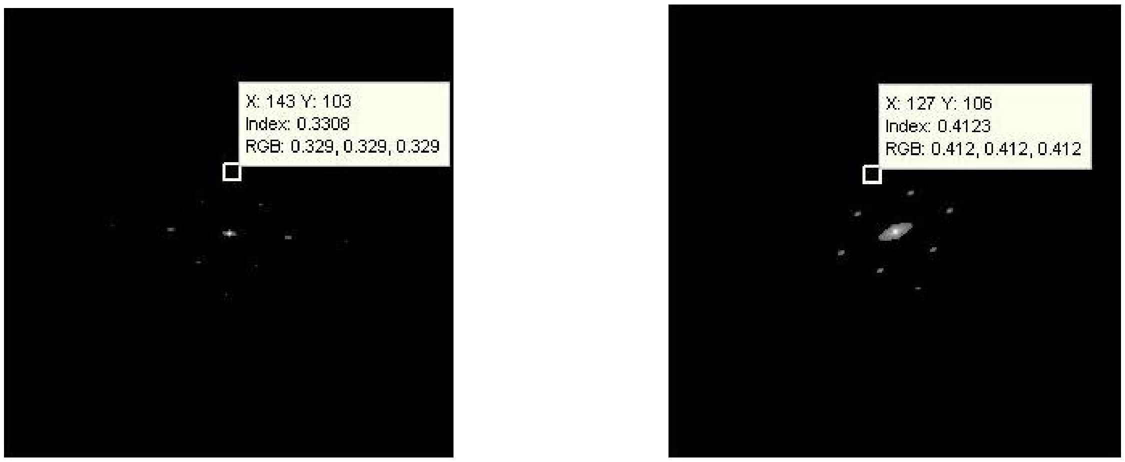
The FGT-Constellation approach is based on determination of the bright spot which has minimum absolute phase, and its closest to the origin (But not at the origin, as the origin is also a bright spot). This method includes the calculation of autocorrelation via FFT just like Polar Transform Approach. Different from the previous method, the implementation of FGT-Constellation involves a thresholding over the autocorrelation of the images. In the thresholded image, bright spots will appear that refer to the peaks of autocorrelation, $R(x, y)$. As it is mentioned before, the translation effect is eliminated with autocorrelation and the rotation of the texture rotates the bright spots of $R_t(x, y)$, "constellation" like images.

From the Figure 2.2(a) and 2.2(b), it is clear that the bright spots are also rotated with the same degree. After the ACF of the images have been computed through FFT algorithms, next step is normalizing and thresholding. The ACF is normalized with respect to its mean value. The thresholding and normalizing processes have been applied to get the bright spots more clearly and determine the nearest point to the origin. Instead of classical thresholding which rounds the lower levels to zero and higher levels to one, logarithmic thresholding has been applied. This is a common application to display the

Fourier Transform. Applying linear thresholding to an image with large dynamic range, let's say its maximum magnitude value is approximately 10 times bigger than the second largest value, results a display of only the largest value in the center. All other values will be rounded to zero and will not appear on the screen. The logarithmic transform will raise the low levels and reduce the high levels [13].

$$F(k,l) = 1 + \log(1 + F(k,l)) \tag{4.1}$$

After the logarithmic transform, a thresholding is applied. The maximum value of the FFT is computed and the thresholding level is determined according to the maximum value. In Table **4.1** FGThrVal is the proportion value of the maximum to define the thresholding level and it is pre-defined. For the texture rotated with 25°, the resulted ACF can be checked in Figure **4.1** . The rotation of the spots are clear. To estimate



(a) Original (b) Rotated with 25°

Figure 4.1 Thresholded ACF of Figure **3.2**

the angle, calculating the displacement of a bright spot is enough. The closest point to the origin is selected. It is defined by line scanning from the center of the image, origin.

The line scanning starts from the origin, decreases the row number and increases the column number. The positions of the selected point from the original image and the rotated image are compared and the arctan of the displacement gives the rotation angle value. where N is the image size, RF and CF are the row and column numbers of the

Table 4.1 Logarithmic Thresholding Algorithm in C

```
double logThrCImage(double ci[N128][N128])
{
    int i,j;
    double maxim;
    maxim=0;
    for(i=0; i<N128; i++) {
        for(j=0; j<N128; j++) {
            ci[i][j]=1+ log(1+ci[i][j]);
            if(ci[i][j] > maxim)
                maxim = ci[i][j];
        }
    }
    for(i=0; i<N128; i++) {
        for(j=0; j<N128; j++) {
            if(ci[i][j] < FGThrVal * maxim)
                ci[i][j] = 0;
            else
                ci[i][j] = 1;
        }
    }
}
```

rotated spot. The distance to the origin is calculated by subtracting N / 2 from the row and column number. arctan of the ratio gives the angle in radians. Last, the result is subtracted from 90° because the angle found is the complementary angle of the rotation angle. The logThrCImage and findCImage functions are explained before. FFT2 is the computation of ACF with FFT. The absolute square of the image is calculated. It is thresholded, shifted, normalized and the brightest point is determined.

Table 4.2 Finding The Brightest Spot Nearest to Origin Algorithm in C

```
void findCImage(CImage* ci)
{
    int i, j;
    RF = N / 2;
    CF = N / 2;
    for (i = -20; i < 0; i++)
        for (j = 0; j <= 20; j++)
            if ((abs(i) > 4) & (abs(j) > 4) & (ci[RF+i][CF+j] > 0)) {
                RF += i;
                CF += j;
                return;
            }
}
```

Table 4.3 Determination of the Displacement of the Rotated Spot Algorithm in C

```
double test_vision(CImage* ci)
{
    FFT2(1, ci);
    absSqrCImage(ci);
    logThrCImage(ci)
    shiftCImage(ci);
    normalizeCImage(ci);
    findCImage(ci);
    return(90 - 180 * atan(-(CF - N / 2)/ ((double) RF - N / 2)) / PI);
}
```

The coordinate system transformation is left out which shortens the duration. But still it is not computationally efficient because of the floating point FFT calculations.

To overcome the computationally demanding autocorrelation computation, which is done by floating point FFT, 2-D autocorrelation computation via *Number Theoretic*

Transform is tested. The results of NTT are quite acceptable. To fasten the algorithm, divide and conquer method is applied to FFT computation.

4.1. Number Theoretic Transform

Different transform domain methods are used to reduce the excessive amount of computational effort that is required for direct computation of convolution and correlation. The methods used are Fast Fourier Transform (FFT), Number Theoretic Transform (NTT), Winograd Fourier Transform Algorithm (WFTA), and Agarwal-Cooley (AC) algorithms [43].

Using FFT algorithms to compute correlation is discussed in the previous chapter. However all FFT computations involve complex number multiplications that result floating-point multiplications. Because digital computations are restricted with finite word-length, round-off errors occur. [44].

Number Theoretic Transforms are also discrete Fourier transforms. Instead of the floating point numbers used in the calculation of FFT integer numbers are used. When it is compared to other methods, it is always preferable due to its exactness, in other terms it is free of round-of errors. The implementation is done by using modular arithmetic [43].

Recalling the FFT formulation in Equation 3.3, W_M is $\exp^{-j2\pi/M}$ where $W_M^M = \exp^{(-j2\pi/M)^M} = 1$. Instead of using the complex number W_M, the calculations are done in a new number field. In this number field, an integer 'G' is defined to replace with 'W_M' where $G^M = 1$. It is known that integers modulo a prime number 'p' form a number field [45]. If the number field is defined as modulo p, the FFT formulation in Equation 3.3 becomes :

$$F(k,l) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) * G_M^{xk} * G_N^{yl} \qquad mod(p) \tag{4.2}$$

where $G^M = 1(modp)$ and also $G^N = 1(modp)$. The image size is M*N. All computations are performed modulo 'p'.

4.1.1. Prime Numbers Algorithm

Definition A prime number is an integer p greater than one with the property that 1 and p are the only positive integers that divide p. There are many theorems to define *prime number*. According to *Wilson's Theorem*, p is prime if and only if :

$$((p-1)! + 1) \;\; \% \;\; p = 0 \tag{4.3}$$

The symbol % is used for remainder. For a prime number p, Fermat's little theorem states that if a is any integer that does not have p as a factor and p is prime [46] :

$$a^{p-1} = 1 \qquad (mod \;\; p), \qquad\qquad \forall \;\; integer \; a \tag{4.4}$$

The defined *modulo a prime number* field is Galois Field. GF(p) for any prime, p, this Galois Field has p elements which are the residue classes of integers modulo p. To be able to define GF(p) clearly, first the terms of Number Theory, group, ring and field are need to be defined.

A ***group*** is an algebraic system consisting of a set, an identity element, one operation and its inverse operation.

A ***ring*** is an algebraic system consisting of a set, an identity element, two operations and the inverse operation of the first operation.

A ***field*** is an algebraic system consisting of a set, an identity element for each operation, two operations and their respective inverse operations.

In the light of these definitions, the operation specifications and further definitions can be listed as :

First; an *R-module* is an abelian group M (R denotes a ring) which is a set G together with a binary operation: * on G such that
(i) $\forall$ *a, b* ϵ *G, there exists c* ϵ *G, such that a*b = c* (i.e., **closure**).
(ii) $\forall$ *a, b, c* ϵ *G, a*(b*c) = (a*b)*c (i.e., * is* ***associative****)*
(iii) *there exists e* ϵ *G (called the* ***identity element****), such that* $\forall$ *a* ϵ *G, a*e = a = e*a,*
(iv) $\forall$ *a* ϵ *G, there exists a*$\prime$ ϵ G *(called the* ***inverse of*** *a) such that* $a * a\prime = e = a\prime * a$,
(v) $\forall$ *a, b* ϵ *G, a*b = b*a (i.e., * is* ***commutative****).*

So if the abelian group *R-module* satisfies all the conditions above for * and + operations and also the following one:
(vi) $\forall$ *a, b, c* ϵ *G a*(b + c) = a*b + a*c (i.e., * is* ***distributive*** *over +).* it is a field [47].

Since the other laws are clear enough to verify, the Inverse Rule is going to be discussed in details for *modulo a prime number* field. The Inverse Rule requires that for every number x in the field, there exists a unique inverse number $-x$ also in the field, where

$$(-x) + x = x + (-x) = 0 \tag{4.5}$$

for addition and for every number x (except *zero*) in the field, there exists a unique inverse number x^{-1} also in the field, where

$$x^{-1} * x = x * x^{-1} = 1 \tag{4.6}$$

for multiplication. For the modulo prime number field, the multiplicative inverse means that $\frac{1}{x}$ is defined as :

$$x * \frac{1}{x} = 1 \qquad (mod\ \ p) \tag{4.7}$$

This rule will be used to define the backward transform of NTT.

To be able to define a number field using a prime number, first the prime number should be identified clearly. The required prime number, p should satisfy

$$p > 2NM2^{2m} + 1, \tag{4.8}$$

where N*M is the image size, and m is the number of bits used to store the intensity information. Note that, to be able to perform fast GT by usual "divide and conquer" type approach of FFT, we need the prime to be of the form

$$p = 2^{u+1}k + 1, \tag{4.9}$$

where $N = 2^u$ is the image size. The first algorithm to handle is a primeness test algorithm which is looking for a prime number satisfying the Equation 4.8 and 4.9.

The image size is set default as $64 * 64(N * M)$ for this project. The number of bits used is $m = 8$. The lower limit for p is $B = 2 * 64 * 64 * 256 * 256$. From Equation 4.9 p should be in the following form :

$$p = (2 * 64 * k) + 1, \tag{4.10}$$

The primeness test algorithm is implemented for those numbers $128 * k + 1$. The number k is used to assure that p is bigger than B and the test is re-implemented by increasing k one by one. The primeness test algorithm in Table **4.4** gets a candidate number cand. If the number is 2 or any other even number, it is declared that cand is not prime. Else, for odd numbers greater than 3, the number should verify that it has no divisors. Second

Table 4.4 Primeness Test Algorithm in C

```
/* Test Primeness */
int isPrime(INT64 cand) {
int count;
if (cand == 2)
    return 1; /*TRUE;*/
if (cand % 2 == 0)
    return 0; /*FALSE;*/
for (count = 3; count * count <= cand; count += 2)
    if (cand % count == 0)
        return 0; /*FALSE;*/
    return 1; /*TRUE;*/
}
```

step is to find the generator "G". G in Equation 4.2 must have the form $G = r^k (mod\ \ p)$, where r is a "primitive root" of p and k is $k = (p-1)/N$. "A primitive root of p is a number r such that any integer a between 1 and p-1 can be expressed by $a = r^k modp$, with k a nonnegative integer smaller that p-1"[48]. Since there is no even prime numbers except 2, if p is not equal to 2 it is an odd number. Then r is a primitive root of p if and only if

$$r^{(p-1)/q} > 1 \qquad (mod\ \ p), \qquad \forall\ primedivisors\ \ q\ \ of\ \ p-1 \tag{4.11}$$

So, the following two conditions guarantee that there exists at least one primitive root for prime number p.

$$r_k^{(p-1)/q_k} > 1 \qquad mod\ \ p \tag{4.12}$$

$$r_k^{p-1} = 1 \qquad mod\ \ p \tag{4.13}$$

for all prime factors/divisors q_k of p-1.

The order of a root r is the smallest positive x for which $r^x = 1(mod\ \ p)$. Notice that since

$$G^N = r^{N*k} = r^{(p-1)} = 1 \qquad (mod\ \ p), \tag{4.14}$$

G will behave similar to $e^{(-j*2*pi)/N}$

A primitive root for p must include all of the residue classes mod p. Residue classes for mod p can be defined as $0, 1, ..., p-1$ or $-\frac{p-1}{2}, ..., 0, ..., \frac{p-1}{2}$. The first residue class is preferred. None of the the first *p-1* powers of the primitive root are the same in modulo p, because except 0 there are *p-1* residue classes mod p. From Fermat's Little Theorem, it is known that powers of an integer form a maximum *p-1* length repeating cycle. So, for a primitive root r, the cycle should be as long as possible [49].

Let's see an example to find out primitive roots. For prime number $p = 11$, the prime factors of $p-1$ are 2 and 5.
$1^{10/2} = 1^5 = 1$ mod 11, $1^{10/5} = 1^2 = 1$ mod 11
$2^{10/2} = 2^5 = 10$ mod 11, $2^{10/5} = 2^2 = 4$ mod 11
$3^{10/2} = 3^5 = 1$ mod 11, $3^{10/5} = 3^2 = 9$ mod 11
$4^{10/2} = 4^5 = 1$ mod 11, $4^{10/5} = 4^2 = 8$ mod 11
$5^{10/2} = 5^5 = 1$ mod 11, $5^{10/5} = 5^2 = 3$ mod 11
$6^{10/2} = 6^5 = 10$ mod 11, $6^{10/5} = 6^2 = 3$ mod 11
$7^{10/2} = 7^5 = 10$ mod 11, $7^{10/5} = 7^2 = 5$ mod 11
$8^{10/2} = 8^5 = 10$ mod 11, $8^{10/5} = 8^2 = 9$ mod 11
$9^{10/2} = 9^5 = 1$ mod 11, $9^{10/5} = 9^2 = 7$ mod 11
$10^{10/2} = 10^5 = 10$ mod 11, $10^{10/5} = 10^2 = 1$ mod 11 From the numbers above 2, 6, 7 and 8 are satisfying the conditions in Equation 4.13. Let's check 2 for residue classes mod 11.
$2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 5$, $2^5 = 10$ mod 11
$2^6 = 9$, $2^7 = 7$, $2^8 = 3$, $2^9 = 6$, $2^10 = 1$ mod 11
So the result set of 2^k modulo 11 for k = 1, 2, ..., 10 involves the residue class 1, 2,...,

10. It is the least primitive root of prime 11. A prime number may have more than one primitive roots. The least prime number is taken to calculate a generator. Several

Table 4.5 Primitive Root Algorithm in C

```
INT64 primRoot(INT64 p, INT64* pf, int ne)
{
    INT64 ind, num, gen;
    INT64 pow;
    for (gen = 2; gen < p - 1; gen++) {
        for (ind = 0; ind < ne; ind++) {
            pow = (p - 1) / pf[ind];
            num = modp(gen, pow, p);
            if (num == 1)
                break;
            }
         if (num ! = 1)
            return(gen);
        }
}
```

Methods can be applied to find the primitive root of a prime. The algorithm implemented for this project uses prime factorization. First, the prime number factors are computed for *p - 1*. Prime factors are the divisors of a number which are all prime. For a prime number p = 23, the prime number of $p - 1 = 22$ are 2 and 11.

For all numbers less than *p-1* are checked out if Equation 4.15 is equal to 1.

$$X^{(p-1)/p} \qquad (modp), \qquad\qquad X = 2, 3, ..., p - 2 \tag{4.15}$$

where *pf* is an element of prime factors array. A number X for which this equation for all prime factors results are all different from 1 is a primitive root. The least primitive root is selected to generate the number *G*. The rest of the computation is done modulo p by replacing G with the exponential term of the FFT formulation.

The expression *modp(a, b, p)* in Table **4.6** represents a^b in modulo p. The algorithm is implemented from Equation 4.2 where instead of x and y, i and j are used for index variables of *f*. The inverse transform is done by

$$f(k,l) = \frac{1}{NM} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) * G_M^{-xk} * G_N^{-yl} \qquad mod(p) \tag{4.16}$$

G^{-1} is the multiplicative inverse of G defined in the modulo prime number field. It is calculated using the inverse rule declared in Equation 4.7.

$$X^{-1} = X^{p-2} \qquad mod(p) \tag{4.17}$$

gives the multiplicative inverse of X in modulo prime number field.

Table 4.6 Modular FFT Algorithm in C

```
INT64 myFFT(INT64 Xco[N128][N128], INT64 newX[N128][N128], INT64 p, INT64 G)
{
    int i, j, k, l, sumX1, sumX2;
    INT64 xg[N][N], xp[N][N];
    for (k = 0; k < N; k++)
     {
        for ( j = 0; j < N; j++)
         {
            sumX1 = 0;
            for (i = 0; i < N; i++)
             {
                xg[i][j] = modp(newX[i][j] * (modp(G, (k * i), p)), 1, p);
                sumX1 = modp(sumX1 + xg[i][j], 1, p);
            }
            xp[k][j] = modp(sumX1, 1, p);
        }
    }
    for (k = 0; k < N; k++)
     {
        for ( l = 0; l < N; l++)
         {
            sumX1 = 0;
            for (j = 0; j < N; j++)
             {
                xg[k][j] = modp(xp[k][j] * (modp(G, (l * j), p)), 1, p);
                sumX1 = modp(sumX1 + xg[k][j], 1, p);
            }
            Xco[k][l] = modp(sumX1, 1, p);
        }
    }
}
```

4.2. Divide and Conquer Algorithm of FFT

Divide and Conquer is an algorithm technique that divides the algorithm into two or more smaller parts. After solving each of these smaller parts recursively the solutions are combined to get the results for the whole algorithm [50].

The term "divide and conquer" has the meaning in engineering applications that; solution of a large problem can be obtained by dividing it into smaller ones and evaluating the solutions of the smaller parts in an organized way. The advantage of such an approach can be summarized as modular and systematic solutions. One of the most common examples can be noticed at the implementation of fast algorithms for discrete transforms [51].

The computation of the DFT is time consuming so that different algorithms are developed to be able to compute it in a more efficient way. The FFT, Fast Fourier Transform is an algorithm that computes the DFT extremely economic. It utilizes the periodicity and symmetry of trigonometric functions for computation with approximately $N \log_2 N$ operations instead of N^2 operations required for the DFT citenumbook. N is the size of the data. Direct computation of DFT requires N^2 complex multiplications and $N(N-1)$ complex additions. FFT algorithm was first developed in 1984 [53] and has been improved with modern digital computers. In 1965, J. W. Cooley and J. W. Tukey have introduced a computation of FFT using divide and conquer that today lots of algorithms are generated from their algorithm, '*Cooley-Tukey algorithm*' [54]. An alternative approach to Cooley-Tukey algorithm is '*Sandey-Tukey algorithm*'. The first algorithm is a member of Decimation in Time radix-2 algorithms and the second one is a member of Decimation in Frequency radix-2 algorithms.

The radix-2 algorithms are known as the simplest FFT algorithms. The decimation-in-frequency (DIF) radix-2 FFT divides the computation into even-indexed and odd-indexed outputs whereas the decimation-in-time (DIT) radix-2 FFT partitions the DFT into two half-length DFTs of the even-indexed and odd-indexed time samples [55].

The DIF algorithm is used to compute the FFT algorithm for this project. To be able to simplify the resulting algorithm, the size of the image should be in the form of :

$$N = 2^m \tag{4.18}$$

where m is an integer. The image size is N=64 which satisfies the condition. Remember that the formulation of DFT is :

$$F(l) = \sum_{k=0}^{N-1} x(k) W^{kl} \tag{4.19}$$

where $W = \exp^{-i(2\pi/N)}$ for l = 0, 1, ..., N - 1. Now, the sample is divided in half; from zero to N/2 and N/2 to N :

$$F(l) = \sum_{k=0}^{(N/2)-1} x(k) W^{kl} + \sum_{k=(N/2)-1}^{N} x(k) W^{kl} \tag{4.20}$$

Let's define a new variable, $m = k - \frac{N}{2}$ so that the second summation may consist with the first:

$$F(l) = \sum_{k=0}^{(N/2)-1} x(k) W^{kl} + \sum_{m=0}^{(N/2)-1} x(m + N/2) W^{l(m+N/2)} \tag{4.21}$$

and in a combined form:

$$F(l) = \sum_{k=0}^{(N/2)-1} (x(k) + \exp^{-i\pi l}) W^{kl} \tag{4.22}$$

Since $\exp^{-i\pi l} = (-1)^l$, for even points it is 1 and for odd ones it is -1. For even values of l, the Equation 4.22 becomes :

$$F(2l) = \sum_{k=0}^{(N/2)-1} (x(k) + x(k + N/2))W^{2kl} \tag{4.23}$$

and for odd l values:

$$F(2l + 1) = \sum_{k=0}^{\frac{N}{2}-1} (x(k) - x(k + N/2))W^{(2k+1)l} \tag{4.24}$$

for l = 0, 1, 2,..., (N/2)-1.

Therefore, N-point DFT computation via the decimation-in-frequency FFT reduces the number of multiplications and additions; $N/2 \log_2 N$ complex multiplications and $N \log_2 N$ complex additions. Decimation-in-frequency FFT is computed using Number Theoretic Transform as described before, so the complex calculations are replaced with modulo calculations which is another factor to reduce the computation time. Decomposing an N-point DFT into two N/2-point DFTs as a preprocessing stage of DIF is depicted in Figure **4.2** .

The illustration in Figure **4.3** shows that the output is in bit-reversed order. That is, if the frequency-sample index n is written as a binary number, the order is that binary number reversed. For example for N = 1, the bit order is 001, and the reversed order will be 100 which is equal to 4.

Pseudocode of FFT that used to generate a C code to compute decimation-in-frequency FFT is listed in Table **4.7** . The algorithm is generated using Chapra's Numerical Methods for Engineering book [52].

The complex parts are removed and modulo arithmetic computations are replaced for the project.

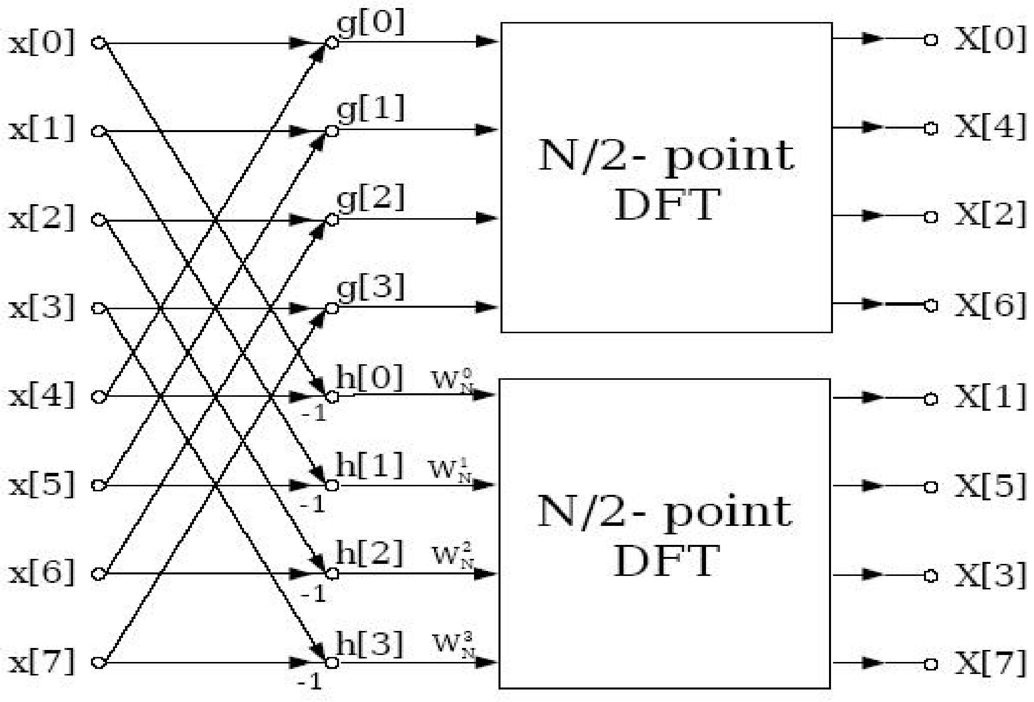


Figure 4.2 First stage in DIF of N=8

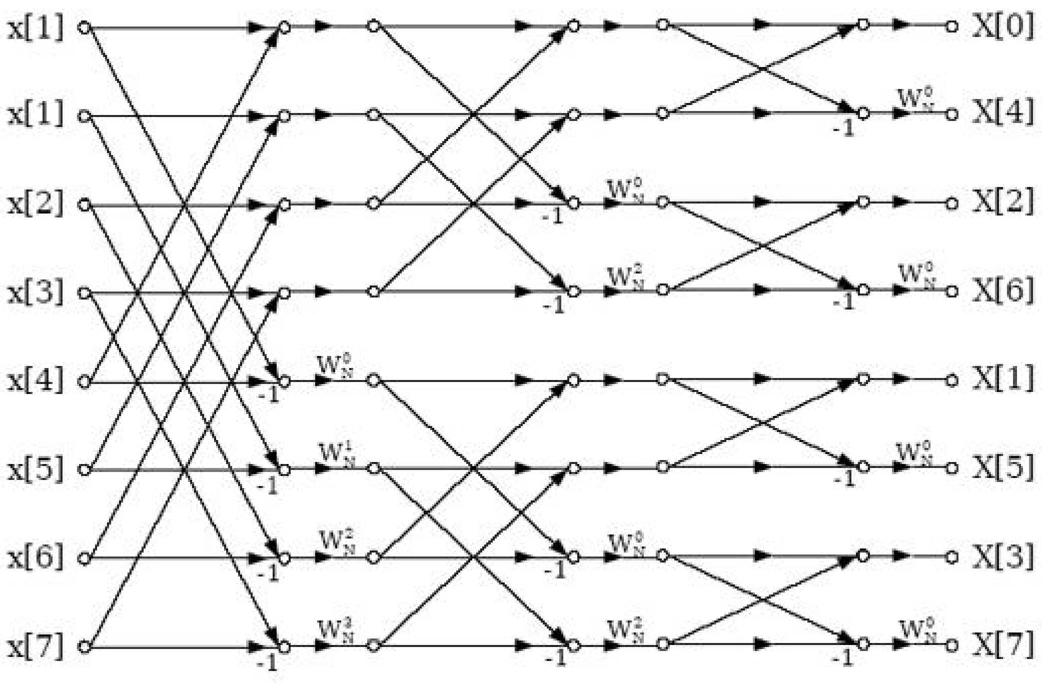


Figure 4.3 Flow Graph of Complete DIF with N=8

Table 4.7 Pseudocode to implement DIF Fast Fourier Transform. (a) FFT (b) Bit Reversal Routine

```
(a)
m = LOG(N)/LOG(2)
N2=N
DO k=1,m
    N1=N2
    N2=N2/2
    angle=0
    arg=2$\pi$/N1
    DO j = 0, N2-1
        c = cos(angle)
        s = -sin(angle)
        DO i = j, N -1, N1
            kk = i + N2
            xt = x(i) - x(kk)
            x(i) = x(i) + x(kk)
            yt = y(i) - y(kk)
            y(i) = y(i) + y(kk)
            x(kk) = xt * c - yt * s
            y(kk) = yt * c + xt * s
        END DO
        angle = (j + 1) * arg
    END DO
END DO
```

```
(b)
j = 0
DO i = 0, N-2
    IF (i ¡ j) THEN
        xt = $x_j$
        $x_j$ = $x_i$
        $x_i$ = xt
        yt = $y_j$
        $y_j$ = $x_i$
        $y_i$ = yt
    END IF
    k = N/2
    DO
        IF (k $\geq$ j + 1) EXIT
        j = j - k
        k = k/2
    END DO
    j = j + k
END DO
DO i = 0, N-1
    x(i) = x(i)/N
    y(i) = y(i)/N
END DO
```

CHAPTER 5

EXPERIMENTAL RESULTS AND PERFORMANCE ANALYSIS

Different rotation estimation algorithms are implemented for the project. The MATLAB codes of this study are basically used to determine the best approach that suits with the study. Two different computers, two different programming tool/language and two different operating systems are used for this project. The first part, Polar Transform Approach is implemented via MATLAB with a notebook that has a 1.60 GHz Pentium R(M) processor running WinXP with 704 MB RAM. Second part is implemented via C programming language on a desktop computer with a 2.80 GHz Pentium 4 processor running Linux with 512 MB RAM.

One of the problems faced during the Polar Transform implementation in MATLAB is defining Q_e value, the part of the polar form taken into account to compute the rotation angle. Some of the error results for different Q_e values can be followed from Table **5.1** . Negative and positive angle values are tested for $Q_e = Q/7, Q_e = 2.5 * Q/7, Q_e = Q/9$. The error values are generally acceptable. $Q_e = Q/9$ column has minimum errors with respect to other Q_e's for all angle values except $-9^\circ - -12^\circ$ range. Therefore a prior Q_e value is defined as Q/9. The resulting angle is checked and if it is between -6° and -16°, the computation of the angle is repeated by re-defining Q_e as $2.5 * Q/7$. The range and second Q_e value are determined using Table **5.1** . If the range is defined with $Q_e = Q/7$ the estimation error increases for rotation angles higher than 16° (in both directions; positive 16° and negative 16°). The error is over than 1°. With a general comparison, $Q_e = Q/9$ gives the least estimation error values for different rotation angles. So as a

Table 5.1 Error values for different angles and Q_e

ALFA	Positive			Negative		
Angle	$Q_e = Q/7$	$Q_e = 2.5 * Q/7$	$Q_e = Q/9$	$Q_e = Q/7$	$Q_e = 2.5 * Q/7$	$Q_e = Q/9$
1.4	0.661	0.435	0.273	0.709	0.828	0.262
2	0.488	0.009	0.017	0.457	0.508	0.315
3.3	0.213	0.298	0.006	0.162	0.067	0.396
4	0.259	0.216	0.222	0.351	0.343	0.183
5.7	0.275	0.088	0.174	0.122	0.573	0.097
8.1	0.169	0.418	0.017	0.224	0.298	0.631
10.6	0.403	0.779	0.440	0.268	0. 516	1.189
12.5	0.164	0.308	0.040	0.053	0.315	0.434
14.0	0.195	0.394	0.221	0.526	0.041	0.521
15.3	0.914	0.309	0.238	0.886	0.048	0.808
16.0	1.526	0.104	0.013	0.840	0.336	0.746
17.2	1.998	0.095	0.114	0.909	0.925	0.410
18.0	2.517	0.362	0.125	1.341	0.926	0.222
19.4	1.920	0.126	0.052	1.762	0.507	0.319
20.0	1.933	0.198	0.106	1.789	0.663	0.229
22.0	3.186	0.373	0.003	2.931	1.469	0.027
23.0	3.277	0.612	0.143	3.211	0.798	0.140
24.3	3.818	0.383	0.371	2.510	0.350	0.008

default Q_e, $Q_e = Q/9$ is chosen. $Q/9$ does not work well for angles between $-6°$ and $-16°$. If the estimated angle is in this range, the estimation algorithm is re-implemented to reach a better estimation with $Q_e = 2.5Q/7$. As a result, the error is minimized with setting the Q_e value (defining the estimation range) dynamically.

The error graphic in Figure **5.1** demonstrates the error results clearly. "+" and "*" signs refer to different Q_e's and the line is the defined Q_e value that catches the minimum error. The horizontal line is for rotation angle values and the vertical one is for error values.

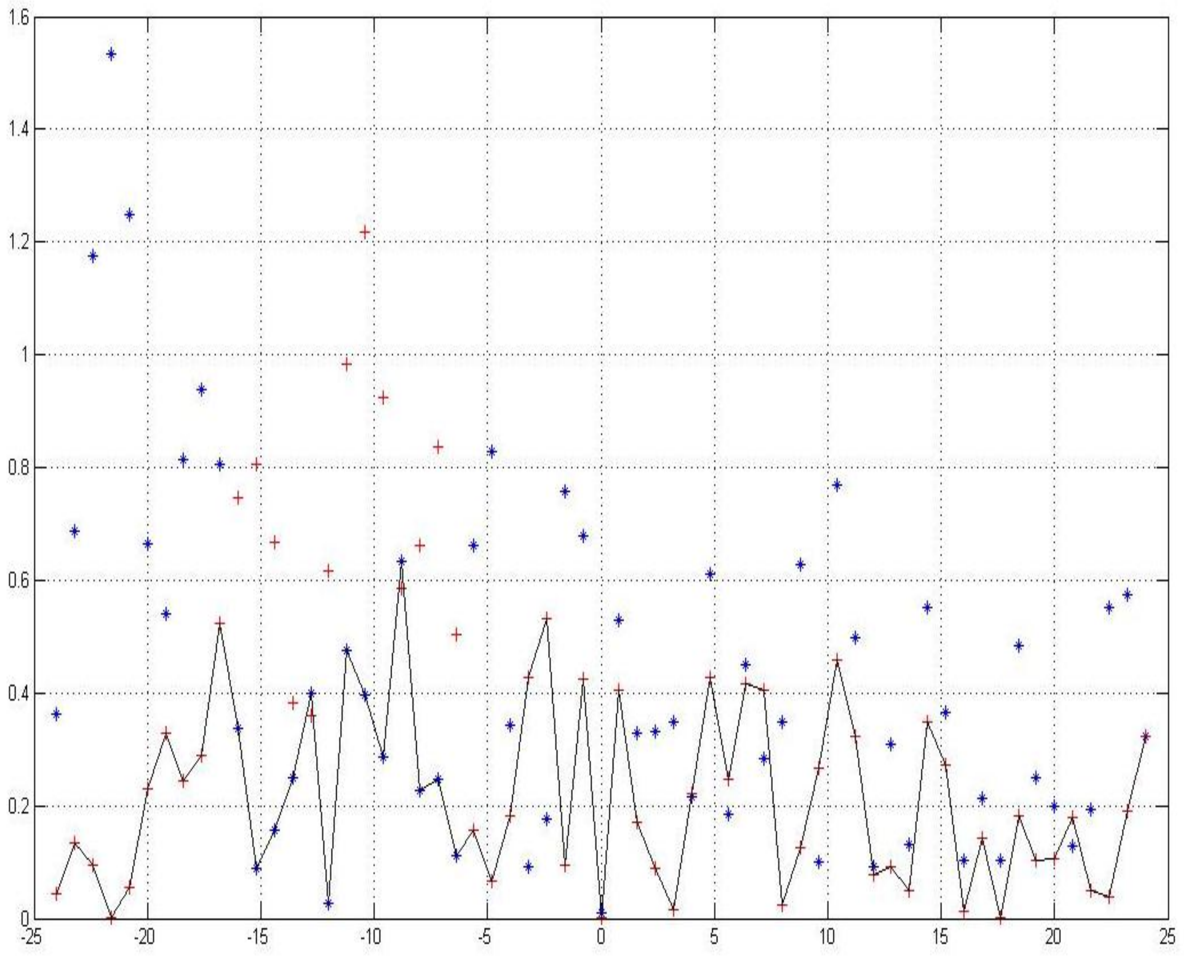


Figure 5.1 Error Graphic between $-25°$ - $25°$ for different Q_e's

The Polar Transform Approach gives reasonable results in a few seconds. For a 25° degree rotated texture, the output of the program is 25.4687. The estimation error results for different rotation angles are displayed in Figure **5.2** . It is seen that maximum error is 0.8325. This error is for angle equals to $-1.2°$. The estimation error is higher than 0.6 only for angles $-9.0, -5.4, -4.8, -1.8, -1.2, 4.8$ degree. The results can be improved with focusing on Q_e value. However the estimation process takes 2 or 3 seconds with a 280*280 image. The image will be used for this project is smaller, but the computation time is still not efficient. To decrease the computational time demanding, the algorithm is implemented to C and run over Linux.

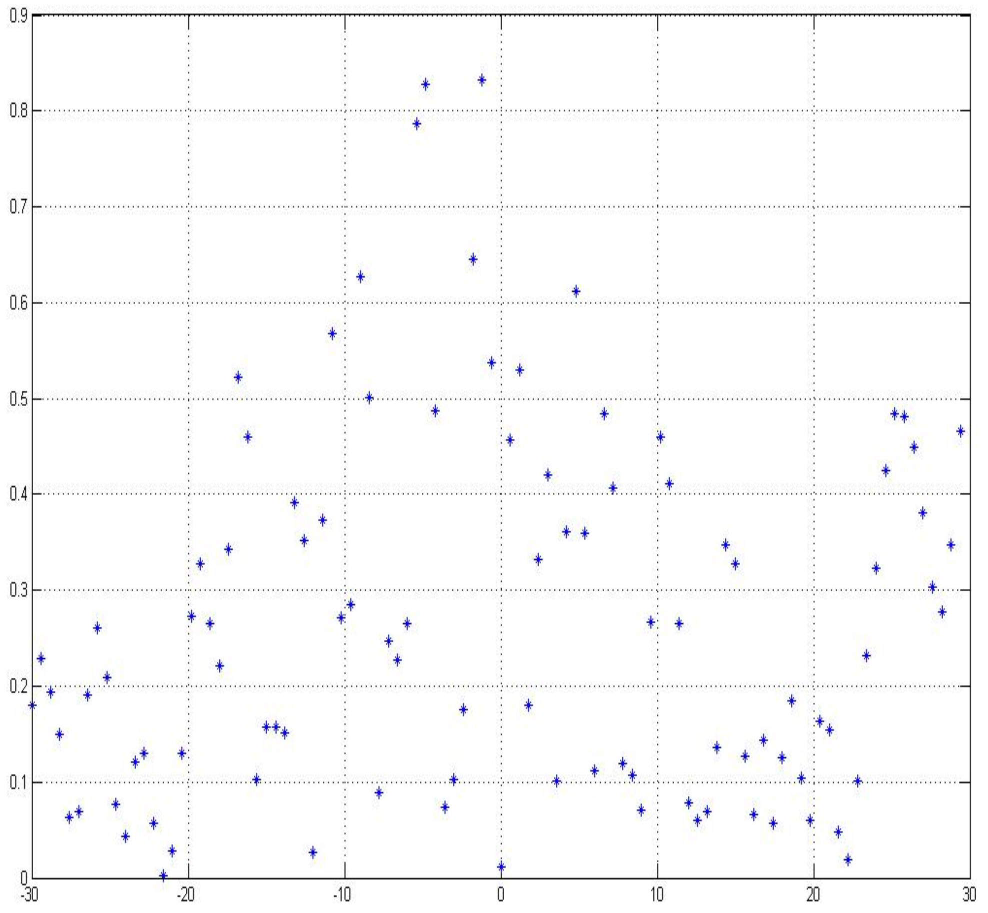


Figure 5.2 Estimation Error Graphic for rotation angle between $-30°$ - $30°$

Since the project is a real-time application, the algorithm is need to be simplified. The coordinate transformation is removed and the angle is estimated through phase of FFT. The error values of the estimation can not be determined exactly since the texture is rotated with unknown angles, but the results for FGT-Constellation method are reasonable. The algorithm gives output just in a few milliseconds as it is desired. To improve the algorithm Number Theoretic Transform and Divide and Conquer methods are studied.

The divide and conquer method has fastened the algorithm four times with respect to the classic computation as it has been expected. For a 64*64 image the computation of FFT lasts 23.85 seconds for the classical algorithm and 108.6 milliseconds for the divide and conquer algorithm.

Table 5.2 Computation Time for Different Size Images (Both use NTT)

Method	8*8	16*16	32*32	64*64
Classical	21.9 ms	239 ms	2.47 s	23.85 s
Div.Con	1.2 ms	5.6 ms	24.7 ms	108.6 ms

To apply NTT, a prime number is needed greater than 2*64*64*256*256 = 536870912 as mentioned in Equation 4.8. The prime number in the form of Equation 4.9 and greater than the mentioned limit is calculated as 536872321 = 2*64*4194315 + 1. The prime factors are 2, 3, 5, 31069. The primitive root is computed as 7 so the generator G is defined as 516347157. $G^N = 1$ modulo p. The inverse G or G^{-1} mod p equals to 532678006.

The Number Theoretic Transform method is implemented to leave out the complex terms and to be free from round off errors. At the same time, it is expected that the computation time may reduce.

NTT method algorithms show a real improvement at the computation time. The complex computation is two times slower than the modulo computation.

CHAPTER 6

CONCLUSIONS

The aim of this study is to determine the rotation angle of a texture for real-time applications. The studied methods are mainly around the Autocorrelation and Fourier Transform of the images since these functions rotate with a rotating image. The required computation method should be rotation-variant and translation-invariant since the rotation angle of the fabric will be determined free from translation.

The first method includes Polar Transformation to be able to calculate the rotation from the shift in Polar Coordinates. This part of the algorithm is implemented in MATLAB, the images are not captured from the camera. Instead, the saved images in the hard disk are used. The images are rotated via MATLAB. First, autocorrelation of the image is computed. Then ACF is transformed to the polar coordinates. The shift of the image in Polar Coordinate System will give the rotation angle. The output of the algorithm is compared with the input rotation angle. The estimation results are acceptable but the algorithm is time demanding.

As a second method, the ACF of the image is thresholded to reduce the computation size. Only the nearest point to the origin is taken into account to estimate the rotation. Since all points rotate with the same degree, this approach seems reasonable. It really reduced the computation time. The system estimates the angle simultaneously with the rotation. To determine the nearest point to the origin, different approaches are used. To draw a rectangle seems a good idea whereas it needs extra time. So the nearest point is selected by line scanning starting from the origin and by scanning row and column wise.

The displacement of the selected point gives the rotation angle. The rest of the study includes improvements on FGT-Constellation Method. ACF is calculated through FFT. Different approaches are tried to compute FFT. Decimation-in-frequency and Number Theoretic Transform are the used methods. The exponential term FFT, its divided and conquered version, FFT computation via modular arithmetic and its divided and conquered version are implemented to the images with different sizes to find out the fastest algorithm. These methods make change on the computation time, not on the estimation results.

To estimate the rotation angle, Polar Transform Approach gives really good results. But if decreasing computation time is more important than decreasing the estimation errors, the second method-FGT-Constellation is promising. The calculation of FFT with exponential terms and using decimation-in-frequency method gives good results from the computation time perspective.

CHAPTER APPENDIX A

MATLAB CODE OF POLAR TRANSFORM APPROACH

```
clear all
close all
tic
alfa=input('Please enter the rotation value: ');
I=imread('kumas2-1.bmp');
I=double(I(:,:,1));
Ishift=imread('kumas2-2.bmp');
Ishift=double(Ishift(:,:,1));
Irot=imrotate(Ishift,alfa3,'bilinear','crop');

[x,y]=size(I);
Icor=myxcorr2(I(:,:,1));
Inorm=abs(Icor)./max(max(Icor));
Icor2=myxcorr2(Irot(:,:,1));
Inorm2=abs(Icor2)./max(max(Icor2));
gammaCons=0.73;
In=Inorm.^gammaCons;
In2=Inorm2.^gammaCons;
[M,N]=size(In);
In=In(round(M/4:3*M/4),round(N/4:3*N/4));
In2=In2(round(M/4:3*M/4),round(N/4:3*N/4));
{[P,Q]}=size(In);
Qe=round(Q/9);
Q1=round((Q/2)-(Qe/2));
Q2=round((Q/2)+(Qe/2));
```

```
Jre1=convPolar(In);
iff1=fftshift(fft2(Jre1));
% FFTSHIFT is useful for visualizing the Fourier transform
% with the zero-frequency component in the middle of the spectrum.
Jre2=convPolar(In2);
iff2=fftshift(fft2(Jre2));

fftRatio=[];
kLP=round(P/3); % TAKE CARE
for l=1:Q
   fftRatio=[fftRatio iff2(round(P/2)-kLP:round(P/2)+kLP,l) \
            iff1(round(P/2)-kLP:round(P/2)+kLP,l)];
end
ang=angle(fftRatio(Q1+1:Q2)./fftRatio(Q1:Q2-1));
% angle gives the result in radians;
Nc=sum(ang);  \% = 2*pi*dl
dl=Nc/2/pi;
dtc3=dl*(2*pi/(Q2-Q1))*180/pi;

% If the angle is found between -6 and -17 for Qe=2.5*Q/9
% the angle is going to be calculated again to minimize the error.
if(dtc3<=6)
    if(dtc3>=-17)
        Qe=round(2.5*Q/7); % LOOK
        Q1=round((Q/2)-(Qe/2));
        Q2=round((Q/2)+(Qe/2));
        Jre1=convPolar(In);
        iff1=fftshift(fft2(Jre1));
        Jre2=convPolar(In2);
        iff2=fftshift(fft2(Jre2));
        fftRatio=[];
        kLP=round(P/3);
        for l=1:Q
           fftRatio=[fftRatio iff2(round(P/2)-kLP:round(P/2)+kLP,l) \
                     iff1(round(P/2)-kLP:round(P/2)+kLP,l)];
        end
        ang=angle(fftRatio(Q1+1:Q2)./fftRatio(Q1:Q2-1));
        Nc=sum(ang);  % = 2*pi*dl
        dl=Nc/2/pi;
        dtc3=dl*(2*pi/(Q2-Q1))*180/pi;
    end
end
X=180*(angle(fftRatio))/pi;
figure();
plot(X)
hold on
grid on
plot([Q1 Q1],[-100 100],'r')
 plot([Q2 Q2],[-100 100],'r')
title('Phase of The FFTRatio and The Range');
 err3=abs(dtc3-alfa3)
 toc
```

CHAPTER APPENDIX B

C CODE of COMPUTATION ACF, (FFT computed via NTT)

```
#include<stdio.h>

#include<math.h>

#include <stdlib.h>
#include <time.h>
#include <sys/times.h>
#include <sys/wait.h>

#define INT64 long long int
#define N128    128
#define N64     64

INT64 myInvFFT(INT64 xy_inv[N128][N128], INT64 xy[N128][N128], INT64
p, INT64 G);

INT64 myFFT(INT64 X_co[N128][N128], INT64 newX[N128][N128], INT64 p,
INT64 G);

void update_table(INT64*, INT64, INT64*, int*); int
find_prime_factors(INT64, INT64*, int*);
int main(int, char**);
INT64 prim_root(INT64, INT64*, int);

INT64 modp(INT64, INT64, INT64);
```

```
int is_prime(INT64 cand);

int main(int argc, char** argv) {
    int ne, i, m, inde, j;
    INT64 pf[100], myX[N64][N64], myY[N64][N64], X_co[N128][N128];
    INT64 Y_co[N128][N128], xg[N128][N128], yg[N128][N128];
    INT64 X_inv[N128][N128], Y_inv[N128][N128], newX[N128][N128];
    INT64 newY[N128][N128], xy[N128][N128], xy_inv[N128][N128];
    INT64 a = N128, b = 1, M = 536870912LL;/*2*64*64*256*256*/
    INT64 k = (M-b)/a + 1;
    INT64 Kn, pri_g, G, sumX, sumY, G_inv, iCo;
    long ratio;
    clock_t time1;
    clock_t time2;

    ratio =1000/CLK_TCK;
    time1 = clock();
    while (!is_prime(a * k + b))
        k++;
    INT64 p = a * k + b;
    INT64 n = p - 1;
    Kn = n / N128;
    find_prime_factors(n, pf, &ne);
    pri_g = prim_root(p, pf, ne);
    G = modp(pri_g , Kn, p);
    myFFT(X_co, newX, p, G); // The image is newX, FFT of newX is X_co;
    myFFT(Y_co, newY, p, G);
    for (i = 0; i < N128; i++)
    {
        for (j = 0; j < N128; j++)
        {
            xy[i][j] = modp(X_co[i][j] * Y_co[i][j], 1, p);
            xy[i][j] = modp(xy[i][j], 1, p);
        }
    }
    myInvFFT(xy_inv, xy, p, G);
    time2 = clock();
    printf("Took %ld ms\n", (ratio * (long)time2 - ratio * (long)time1)/10000 );
}
 /* primitive Root Function*/
 INT64 prim_root(INT64 p, INT64* pf,
int ne) {
    INT64 ind, num, gen;
    INT64 pow;
    for (gen = 2; gen < p - 1; gen++) {
        for (ind = 0; ind < ne; ind++) {
            pow = (p - 1) / pf[ind];
            num = modp(gen, pow, p);
            if (num == 1)
                break;
        }
        if (num != 1)
            //printf("\nPRIM_ROOT=%lld\n", gen);
            return(gen);
    }
}
/* Calculate Powers for modulo*/
INT64 modp(INT64 x, INT64 n, INT64
pr) {
        INT64 r1, r2, r;
        INT64 i;
    if (n == 0)
        return(1);
    else if (n == 1)
        return(x % pr);
    else {
        r1 = modp(x, n % 2, pr);
        r2 = modp(x, n / 2, pr);
            r = (r2 * r2) % pr;
        r = (r * r1) % pr;
        return(r);
    }
}
void update_table(INT64* np, INT64 p, INT64* pf, int* nep) {
    if ((*np) % p == 0) {
        pf[(*nep)++] = p;
        *np = (*np) / p;
    }
}
 /* Find Prime Divisors */
 int find_prime_factors(INT64 n, INT64*
pf, int* nep) {
    INT64 p;
    *nep = 0;
    p = 2;
    while (n % p == 0)
        update_table(&n, p, pf, nep);
   for (p = 3; p * p <= n; p += 2)
        while (n % p == 0)
        update_table(&n, p, pf, nep);
   if (n > 1)
    pf[(*nep)++] = n;
}

/* Test Primeness */
int is_prime(INT64 cand) {
    int count;
    if (cand == 2)
        return 1; /*TRUE;*/
    if (cand % 2 == 0)
        return 0; /*FALSE;*/
    for (count = 3; count * count <= cand; count += 2)
        if (cand % count == 0)
            return 0; /*FALSE;*/
    return 1; /*TRUE;*/
}
 INT64 myFFT(INT64 X_co[N128][N128], INT64 newX[N128][N128], INT64
p, INT64 G) {
    int i, j, k, l, sumX1, sumX2;
    INT64 xg[N128][N128], xp[N128][N128];
    for (k = 0; k < N128; k++) {
        for ( j = 0; j < N128; j++) {
            sumX1 = 0;
            for (i = 0; i < N128; i++) {
                xg[i][j] = modp(newX[i][j] * (modp(G, (k * i), p)), 1, p);
                sumX1 = modp(sumX1 + xg[i][j], 1, p);
            }
            xp[k][j] = modp(sumX1, 1, p);
        }
    }
    for (k = 0; k < N128; k++) {
        for ( l = 0; l < N128; l++) {
            sumX1 = 0;
            for (j = 0; j < N128; j++) {
                xg[k][j] = modp(xp[k][j] * (modp(G, (l * j), p)), 1, p);
                sumX1 = modp(sumX1 + xg[k][j], 1, p);
            }
            X_co[k][l] = modp(sumX1, 1, p);
        }
    }
}

INT64 myInvFFT(INT64 xy_inv[N128][N128], INT64 xy[N128][N128], INT64
p, INT64 G) {
    INT64 G_inv, iCo, xg[N128][N128], xp[N128][N128];
    int i, j, k ,l , sumX1, sumX2;
    G_inv = modp (G, p -2, p);
    iCo = modp (N128, p-2, p);
    for (k = 0; k < N128; k++) {
        for ( j = 0; j < N128; j++) {
            sumX1 = 0;
            for (i = 0; i < N128; i++) {
                xg[i][j] = modp(xy[i][j] * (modp(G_inv, (k * i), p)), 1, p);
                sumX1 = modp(sumX1 + xg[i][j], 1, p);
            }
            xp[k][j] = modp(iCo * sumX1, 1, p);
        }
    }
    for (k = 0; k < N128; k++) {
        for ( l = 0; l < N128; l++) {
            sumX1 = 0;
            for (j = 0; j < N128; j++) {
                xg[k][j] = modp(xp[k][j] * (modp(G_inv, (l * j), p)), 1, p);
                sumX1 = modp(sumX1 + xg[k][j], 1, p);
            }
            xy_inv[k][l] = modp((iCo * sumX1), 1, p);
        }
    }
}
```

CHAPTER APPENDIX C

C CODE FOR COMPUTATION of FFT via DIF TECHNIQUE

```
 INT64 myFFT(INT64 newX[N128][N128], INT64 p, INT64 G, int m, int dir)
{
    int i, j, k, kk, sumX1, N1, N2, ia, row, col;
    INT64 iCo, st, c, pro, gL;
    INT64 modpow[N128];
    gL = G;
    iCo = 1;
    if (dir == -1)
         iCo = modp (N128, p-2, p);
    pro = 1;
    for (ia = 0; ia < N128; ia++) {
         modpow[ia] = pro;
         pro = modp(pro * gL, 1, p);
   }

   if (dir == -1) {
         pro = 1;
         gL = modpow[N128 - 1];
         for (ia = 0; ia < N128; ia++) {
             modpow[ia] = pro;
             pro = modp(pro * gL, 1, p);
         }
   }
   for (row = 0; row < N128; row++) {
         N2 = N128;
```

```
for (k = 0; k < m; k++) {
    N1 = N2;
    N2 = N2 / 2;
    ia = 0;
    for (j = 0; j < N2; j++) {
        c = modpow[ia];
        for (i = j; i < N128; i += N1) {
            kk = i + N2;
            st = modSub(newX[row][i], newX[row][kk], p);
            newX[row][i] = modp(newX[row][i] + newX[row][kk], 1 , p);
            newX[row][kk] = modp(st * c, 1, p);
            newX[row][kk] = modp(newX[row][kk], 1, p);
            ia = (j + 1) * (N128 / N1);
        }
    }
}
j = 0;
for (i = 0; i < N128 - 1; i++) {
    if (i < j) {
        st = newX[row][j];
        newX[row][j] = newX[row][i];
        newX[row][i] = st;
    }
    k = N128 / 2;
    while (1) {
        if (k >= j + 1)
            break;
        j = j - k;
        k = k / 2;
    }
```

```
         j = j + k;
     }
     for (i = 0; i < N128; i++)
         newX[row][i] = modp(iCo * newX[row][i], 1, p);
}
for (col = 0; col < N128; col++)
{
    N2 = N128;
    for (k = 0; k < m; k++)
    {
        N1 = N2;
        N2 = N2 / 2;
        ia = 0;
        for (j = 0; j < N2; j++)
        {
            c = modpow[ia];
            for (i = j; i < N128; i += N1)
            {
                kk = i + N2;
                st = mod_sub(newX[i][col], newX[kk][col], p);
                newX[i][col] = modp(newX[i][col] + newX[kk][col], 1 , p);
                newX[kk][col] = modp(st * c, 1, p);
                newX[kk][col] = modp(newX[kk][col], 1, p);
                ia = (j + 1) * (N128 / N1);
            }
        }
    }
    j = 0;
    for (i = 0; i < N128 - 1; i++)
    {
            if (i < j)
            {
                st = newX[j][col];
                newX[j][col] = newX[i][col];
                newX[i][col] = st;
            }
            k = N128 / 2;
            while (1)
            {
                if (k >= j + 1)
                    break;
                j = j - k;
                k = k / 2;
            }
            j = j + k;
        }
        for (i = 0; i < N128; i++)
            newX[i][col] = modp(iCo * newX[i][col], 1, p);
    }
}
```

REFERENCES

[1] Kim W. Y. and Kim Y. S., "Robust Rotation Angle Estimator", IEEE TRANS. PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 21, NO. 8, AUGUST 1999

[2] Erhardt + Leimer S.r.I.- Company
http://www.erhardt-leimer.it/prodotti/raddrizzatrama.html

[3] Fuji H, Almeida SP, Dowling JE. Rotational matched spatial filter for biological pattern recognition. Appl Opt 1980;19:1190-5.

[4] Boshra M, Bhanu B. Predicting object recognition performance under data uncertainty, occlusion and clutter. IEEE International Conference on Image Processing, 1998. pp. 556-60.

[5] Iftekharuddin KM, Ahmed F, Karim MA. Amplitude-coupled MACE for automatic target recognition applications. Opt Eng 1996;35:1009- 14.

[6] Diab SL, Karim MA, Iftekharuddin KM. Multiobject detection of targets with fine details, scale and translation variations. Opt Eng 1998;37:876-83.

[7] Iftekharuddin KM, Razzaque MA. Constraints in distortion invariant target recognition system simulation. Proc SPIE 2000;4414:20-31.

[8] Brown LG. A survey of image registration techniques. ACM Comput Surveys 1992;24:325-76.

[9] W.Y. Kim and P. Yuan, "A Practical Pattern Recognition System for Translation, Scale and Rotation Invariance," Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR), pp. 391-396, Seattle, Wash., June 1994.

[10] A. Khotanzad and Y.H. Hong, "Rotation Invariant Image Recognition Using Features Selected via a Systematic Method", Pattern Recognition, vol. 23, no. 10, pp. 1,089-1,101, 1990.

[11] B.S. Reddy and B.N. Chatteji. An fft-based technique for translation, rotation, and

scale-invariant image registration. IEEE Trans. Pattern Analysis and Machine Intelligence, 5(8):126-1270, August 1996.

[12] C.D. Kuglin and D.C. Hines, "The phase correlation image alignment method," in Proc. Int. Conf. on Cybernetics and Society, 1975, pp. 163-165.

[13] R. Fisher, S. Perkins, A. Walker and E. Wolfart, HYPERMEDIA IMAGE PROCESSING REFERENCE, 2003
http://homepages.inf.ed.ac.uk/rbf/HIPR2/fourier.htm

[14] E. D. Castro and C. Morandi. Registration of translated and rotated images using finite fourier transforms. IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI9 (5):700–703, Sept. 1987.

[15] Cideciyan, A. V., Jacobson, S. G., Kemp, C. M., Knighton, R. W., and Nagel, J. H., 1992. Registration of high resolution images of the retina. SPIE Medical Imaging VI: Image Processing 1652, 310-322.

[16] D. Casasent and D. Psaltis. Position, rotation, and scaleinvariant optical correlation. Applied Optics, 15:1793-1799, 1976.

[17] W.K. Pratt. Digital Image Processing. John Wiley - Sons, New York, 1978.

[18] Q. Chen, M. Defrise, and E Deconinck. Symmetric phaseonly matched filtering of fourier-mellin transforms for image registration and recognition. IEEE Trans. Pattern Analysis and Machine Intelligence, 16(12): 1156-1168, December 1994.

[19] Wolberg G., Zokai S., ROBUST IMAGE REGISTRATION USING LOG-POLAR TRANSFORM. IEEE, 2000.

[20] Oppenheim AV, Schafer RW. Discrete-time signal processing. Englewood Cli6s, NJ: Prentice-Hall, 1989

[21] Silverman J. H., "A Friendly Introduction to Number Theory" , 3rd Edition - Pearson Prentice Hall, 2006.

[22] University of Illionis at Urbana-Champaign, Department of Mathematics, "Guide to Graduate Study in Number Theory", 2002.
http://www.math.uiuc.edu/ResearchAreas/numbertheory/guide.html

[23] Rusin D., The Mathematical Atlas, 2006.
http://www.math.niu.edu/ rusin/known-math/index/11-XX.html

[24] Gudvangen S., ”Practical Applications of Number Theoretic Transforms”, Norwegian Signal Processing Symposium, Norway, 1999.

[25] A. Bouridane, A. Pajayakrit, S. Dlay, and A. Holt, ”CMOS VLSI circuits of pipeline sections for 32 and 64-point Fermat number transformers” Integration, vol. 8, pp. 51-64, 1989.

[26] S. Gudvangen and A. Patel, ”Rapid synthesis of a macro- pipelined CMOS ASIC for the Fermat number transform”, in Proc. Norwegian Signal Processing Symposium (NORSIG), Stavanger, Norway, pp. 143-148, 1-2 Sept. 1995.

[27] R. Vander Kraats and A. Venetsanopoulos, ”Hardware for two- dimensional digital filtering using Fermat number transforms”, IEEE Trans. on Acoustics, Speech, and Signal Processing, vol. ASSP-30, pp. 155-162, April 1982.

[28] N. Yamane, Y. Morikawa, and H. Hamada, ”A fast image filtering processor using the Fermat number transform”, Systems and Computers in Japan, vol. 18, no. 1, pp. 67-78, 1987. Translated from Denshi Tsushin Gakkai Ronbunshi, Vol. 69-D, No. 2, Feb. 1986, pp. 198-207.

[29] Toivonen T., ”Number Theoretic Transform -Based Block Motion Estimation”, Department of Electrical Engineering, University of Oulu, Oulu, Finland. Diploma Thesis, 2002.

[30] A. Schonhage and V. Strassen, ”Schnelle multiplication großer Zahlen”, Computing, vol. 7, pp. 281-292, 1971.

[31] V. Pan, ”How can we speed up matrix multiplication?”, SIAM Review, vol. 26, pp. 393-415, July 1984.

[32] A. Yagle, ”Fast algorithms for matrix multiplication using pseudo-number-theoretic transforms”, IEEE Trans. on Signal Processing, vol. 43, pp. 71-76, Jan. 1995.

[33] D. Bini, M. Capovani, F. Romani, and G. Lotti, ”$O(n^{2.7799})$ complexity for n*n approximate matrix multiplication”, Information Processing Letters, vol. 1, pp. 234-235, June 1979.

[34] J. Sullivan and J. Adams, ”A fast algorithm for solving Toeplitz systems of equations”, in Proc. Int. Symp. on Circuits and Systems (ISCAS), pp. 373-376, 1994.

[35] B. Harms and S. Keller-McNulty, ”Error-free solution to a Toeplitz system of equa-

tions", IEEE Trans. on Signal Process- ing, vol. 39, pp. 1212-1215, May 1991.

[36] R. Blahut, "Theory and practice of error control codes", Addison-Wesley Publ. Comp., 1984.

[37] B. Arambepola and S. Choomchuay, "Algorithms and architectures for Reed-Solomon codes", GEC J. of Research, vol. 9, no. 3, pp. 172-184, 1992.

[38] D. Kolba and T. Parks, "A prime factor FFT algorithm using high-speed convolution", IEEE Trans. on Acoustics, Speech, and Signal Processing, vol. ASSP-25, pp. 281-294, Aug. 1977.

[39] Deselaers T., Keysers D., Ney H., "Features for Image Retrieval - A Quantitative Comparison", In DAGM 2004, Pattern Recognition, 26th DAGM Symposium, Lecture Notes in Computer Science, pages 228-236, Tübingen, Germany, September 2004.

[40] Y. M. Wang and H. Zhang, "Detecting image orientation based on low-level visual content," Computer Vision and Image Understanding (CVIU), vol. 93, no. 3, pp. 328-346, 2004.

[41] Z. Liu and S. Wada, "Translation, Rotation and Scale Invariant Texture Characterization Method for Retrieval", Proc. 4th IEEE International Symposium on Signal Processing and Information Technology, 2004.

[42] Crandall, R. E. , "Integer convolution via split-radix fast Galois transform", 1999.

[43] Bhattacharya, M. and Astola, J., "Number Theoretic Transform Modulo K.
$2^n + 1$, A Prime", Signal Processig X Theories and Applications, EUSIPCO, Vol. 4, 4-8 September 2000, Tampere, Finland, pp. 2529-2532, (2000)

[44] T.P. Harte and R. Hanka, "Number Theoretic Transforms in Neural Network Image Classification", Proc. 1st International Workshop on Statistical Techniques in Pattern Recognition, IAPR, Prague, June 9-11, 1997.

[45] Tommila, M., Apfloat, A High Performance Arbitrary Precision Arithmetic Package for C++ and Java
http://www.apfloat.org/ntt.html

[46] Ribenboim P., "The new book of prime number records, Springer", 1995.

[47] Shoup V., "A Computational Introduction to Number Theory and Algebra", 2005.

[48] Silva Toms Oliveira e http://www.ieeta.pt/ tos/

[49] Stromquist, W., Elementary Number Theory Notes
http://www.brynmawr.edu/math/people/stromquist/numbers/primitive.html

[50] National Institute of Standards and Technology,
http://www.nist.gov/dads/HTML/divideAndConquer.html

[51] Guoan B., Zeng Y., Transforms and Fast Algorithms for Signal Analysis and Representations, .

[52] Chapra S. C. and Cahale R. P., Numerical Methods for Engineering with Software and Programming Applications, .

[53] Heidemann M. T., Johnson D. H., and Burrus C. S., Gauss and the History of the Fast Fourier Transform, IEEE ASSP Magazine, pp. 14 -21, October 1984.

[54] J.W. Cooley and J.W. Tukey, "An algorithm for the machine calculation of complex Fourier series," Math. Comp., 19 (1965), pp. 297301.

[55] Jones D., "Decimation-in-Frequency (DIF) Radix-2 FFT," Connexions, September 17, 2006, http://cnx.org/content/m12018/1.6/.