DESIGN AND IMPLEMENTATION OF A SMART CAMERA AS AN EMBEDDED SYSTEM

by

Cihan ULAŞ

June 2007

DESIGN AND IMPLEMENTATION OF A SMART CAMERA AS AN EMBEDDED SYSTEM

by

Cihan ULAŞ

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

June 2007
Istanbul, Turkey

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

_____
Prof. Dr. Muhammet KÖKSAL
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. Onur Toker _____

Assist. Prof. Dr. Atakan Kurt _____

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

June 2007

DESIGN AND IMPLEMENTATION OF A SMART CAMERA AS AN EMBEDDED SYSTEM

Cihan ULAŞ

M. S. Thesis - Electronics Engineering
June 2007

Supervisor: Assoc. Prof. Dr. Onur Toker

ABSTRACT

In the last few decades, smart cameras started to play a very important role in many applications, such as medical applications, traffic surveillance systems and industrial applications. There are several smart camera architectures reported in the literature. However, in this study, we designed our own smart camera architecture and named it as "FU-SmartCam".

This thesis is about the FU-Smart Camera (Fatih University Smart Camera) architecture and its application to an industrial problem. Rotation estimation for the weft-straightening machines is implemented.

Two different methods are proposed for rotation estimation. The first method is based on FFT computation and the second one is based on statistical feature extraction. The advantages and disadvantages of both approaches are discussed and illustrated via examples. Matlab programs are used to compare the results of rotation estimation algorithms. FFT based algorithms also implemented in C and deployed on the Smart Camera.

Keywords: Smart Cameras, Embedded Systems, 2D FFT, Statistical Feature Extraction, Rotation Angle Estimation, Embedded Linux, Texture Analysis.

GÖMÜLÜ SİSTEM OLARAK AKILLI KAMERA TASARIMI VE UYGULAMASI

Cihan ULAŞ

Yüksek Lisan Tezi – Elektronik Mühendisliği
Haziran 2007

Tez Yöneticisi: Doç. Dr. Onur TOKER

ÖZ

Son yıllarda, Akıllı kameralar medikal uygulamalar, trafik izleme ve endüstriyel uygulamalar gibi bir çok alanda önemli rol oynamaya başladı. Literatürde sunulan bir kaç akıllı kamera mimarisi bulunmaktadır, ancak bu uygulamada biz FU-SmartCam diye adlandırılan kendi mimarimizi dizayn ettik.

Bu tez FU-SmartCam (Fatih Üniversitesi Akıllı Kamera) mimarisi ve endüstrideki uygulaması üzerinedir. Atkı düzeltme makinası için kumaştaki dönmeyi hesaplayan uygulama gerçekleştirilmiştir.

Kumaş resmindeki dönme açısının hesaplanması için iki metot önerildi. FFT ve statiksel özelliklere dayanan iki yöntemin avantaj ve dezavantajları tartışıldı ve örnekler ile gösterildi. Dönme açısının hesaplanmasında ve deney sonuçlarının karşılaştırılmasında Matlab programı kullanıldı. Ayrıca FFT metoduna dayalı yöntem C programlama dili ile yazıldı ve Akıllı Kamera üzerinde test edildi.

Anahtar Kelimeler: Akıllı Kameralar, Gömülü Sistemler, 2D FFT, Statiksel Özellik Çıkarımı, Dönme Açısı Hesaplama, Gömülü Linux, Kumaş Analizi.

Dedicated to my parents

ACKNOWLEDGEMENT

Firstly I would like to thank my supervisor Assoc. Prof. Dr. Onur TOKER for his guidance, continuous support, and enlightening experience through my thesis work. I am also grateful that he taught me the importance of patience.

I also thank to Prof Dr. Kemal FİDANBOYLU, Prof Dr. Muhammet KÖKSAL, Assoc. Prof. Erkan İMAL and the other faculty members for their encouragement and support.

Finally I thank my mother and my father for supporting me through all these years.

TABLE OF CONTENTS

LIST OF TABLES

TABLE

LIST OF FIGURES

FIGURE

LIST OF SYMBOLS AND ABBREVIATIONS

SYMBOL/ABBREVIATION

Pd	Gray Level Co-Occurrence Matrix
i, j	The entry of Co-Occurrence Matrix
d	Displacement Vector
r, s, t, v	Pixel indices
I	Image matrix
ρ	Autocorrelation function
M	Robert and Laplace Operators
F_w	1D Fourier Transform
w	Window function
$\Delta t \Delta u$	Time-Bandwidth product
$W_{f,a}$	Wavelet Transform
h	Impulse response
u_0	Frequency of the sinusoidal wave
ϕ	Phase of the sinusoidal wave
$R(\theta)$	Rotation matrix
x, y	Location of the brightest point
I	DC component region on the FFT
I'	Brightest point searching region

ADC	Analog Digital Converter
AF	Address Family
BTTV	Bullet Train to Vegas
CMOS	Complementary Metal Oxide Semiconductor
DFT	Discrete Fourier Transform
DFFT	Discrete Fast Fourier Transform
DOOG	Differences of Offset Gaussian
DSP	Digital Signal Processing/Processor
DT	Discrete Time
FFT	Fast Fourier Transform
FIFO	First Input First Output
FPGA	Field Programmable Gate Array
FT	Fourier Transform
GIPS	Giga Instruction Per Second
GLCM	Gray Level Co-Occurrence Matrices
HCI	Human Computer Interface
HDD	Hard Disk Drive
HLT	Hybrid Label Tree
HMT	Hidden Markov Tree
ICA	Independent Component Analysis
Inca	Intelligent Camera
IP	Image Processing/ Internet Protocol/ Implicit Polynomial
I/O	Input/Output
LAN	Local Area Network
LS	Least Square
MRF	Markov Random Field
OS	Operating System
PLD	Programmable Logic Device
PC	Personnel Computer
PF	Protocol Family
RGB	Red Green Blue
SBC	Single Board Computer
SDL	Simple Direct Media
SIMD	Single Instruction Multiple Data

SmartCam	Smart Camera
TCP	Transmission Control Protocol
TELNET	Telecommunications Network
TI	Texas Instrument
XDAIS	Express DSP Algorithm Standard
VLIW	Very Long Instruction Word

CHAPTER 1

INTRODUCTION

A “smart camera” is generally a video camera combined with a computer vision system in a tiny package. An embedded system is designed for a specific function and performs pre-defined tasks. There are several reasons for designing a smart camera as embedded, such as cost, simplicity, integration, and reliability (Yaghmour, 2003).

Smart cameras are equipped with a high-performance onboard computing communication video capture units. By providing access to more than one view through cooperation among each camera, networks of embedded cameras can potentially support more complex and challenging applications such as smart rooms, surveillance systems, tracking of an object or people, and motion analysis than a single camera (Bramberger et al., 2006).

The question often comes up as to what is the most appropriate approach to take in implementing a vision system using a smart camera or sort of PC based approach. It is very obvious that microprocessors, DSPs and FPGAs are getting faster; therefore, smart cameras are getting more powerful and “smarter”. Thus, they can compete with the more “traditional” approaches to machine vision systems (Maclean, 2005).

An important property of the smart camera is that it reduces the amount of data generated to the *'data of interest'* by making use of embedded image processing algorithms. The data of interest might be, for example, defective areas of the product being inspected. Multiple cameras can route their data to a single frame grabber and computer due to the reduction of data stream, thus dramatically reducing system cost and increasing inspection bandwidth capability. This smart camera also makes use of an on-board microprocessor for communication with the inspection systems' host computer and for internal control functions (Sousa, 2003).

.The main part of the smart cameras is embedded system. Embedded system is desingned for specific tasks and completely dedicated to the device or system it controls. Unlike a general-purpose PC, an embedded system performs one or a few pre-defined tasks, usually with very specific requirements. Since the system is dedicated to specific tasks, design engineers can optimize it, reducing the size and cost of the product. Embedded systems are often mass-produced, benefiting from economies of scale (Yaghmour, 2003).

The reason of using Linux operating systems (OS) is that it can be used in many computers and embedded systems. Linux is interchangeably used in reference to the Linux kernel, a Linux system, or a Linux distribution. Other advantages of Linux are modularity and structure, ease of fixing, extensibility, configurability and error recovery. When source access problems arise, the open source and free software communities seek to replace the "faulty" software with an open source version providing similar capabilities. This contrasts with traditional embedded OSes, where the source code isn't available or must be purchased for very large sums of money. The advantages of having the code available are the possibility of fixing the code without exterior help and the capability of digging into the code to understand its operation. Fixes for security weaknesses and performance bottlenecks, for example, are often very quickly available once the problem has been publicized. With traditional embedded OSes you have to contact the vendor, alert them of the problem, and await a fix. Most of the time, people simply find workarounds instead of waiting for fixes. For sufficiently large projects, managers even resort to purchasing access to the code to alleviate outside dependencies (Yaghmour, 2003).

An embedded Linux system simply designates an embedded system based on the Linux kernel and does not imply the use of any specific library or user tools with this kernel. An embedded Linux distribution may include: a development framework for embedded Linux systems, various software applications tailored for usage in an embedded system, or both. Development framework distributions include various development tools that facilitate the development of embedded systems. This may include special source browsers, cross-compilers, debuggers, project management software, boot image builders, and so on. These distributions are meant to be installed on the development host. Tailored embedded distributions provide a set of applications to be used within the target embedded system. This might include special libraries, executions, and configuration files to be used on the target. A method may also be provided to simplify the generation of root file systems for the target system (Yaghmour, 2003).

In the texture industry, there is a well known mechanism which is weft-straightening machine. This mechanism simply straights the texture if the texture pattern goes wrong. There can be two types of distortion; the first one is linear distortion and the second one is circular distortion. This distortion has to be fixed just before drying of the texture. In industry this problem was solved by using optical systems. The aim of this thesis is to find a suitable algorithm for such a system and build smart camera architecture.

We study for the rotation estimation algorithms on a texture and their implementation in a Vortex86 SBC (Single Board Computer) to overcome the problem that I mentioned above.

The second chapter consists of the literature survey on the smart camera architectures, position estimation and texture analysis. Although there are many researches in the area of pattern recognition for rotation invariant systems, there are very few study on the rotation estimation for the textures. We tried two different methods for rotation estimation on a texture image.

The third chapter proposes the Fast Fourier Transform (FFT) based rotation estimation algorithm. The rotation on texture image is equivalent to rotation on the FFT. Based on this idea, we observe the brightest point in the image and get the brightest point

location. Since the brightest point on the FFT will turn with rotation, we can estimate the rotation angle by tracing this brightest point location.

Fourth chapter introduces the statistical based rotation estimation algorithm. The statistical features are observed with the rotation. The aim of the algorithm is to find the linear change with the rotation. Two parameters modeling and six parameters modeling are compared to each other. Chapter five is devoted to experimental setup for testing the FFT and statistical based rotation estimation algorithms as well as real time embedded Linux installation. Chapter six shows the experimental result of the FFT and statistical based rotation estimation approximations. Finally, conclusions for the thesis are presented in chapter seven.

CHAPTER 2

LITARATURE SURVEY

Smart Cameras as an embedded system are greatly used in many applications, including industrial inspection, robot vision, traffic control, automotive control, surveillance, security systems and medical imaging. This thesis mainly covers two main topics which are smart camera architectures and rotation estimation algorithms for texture images. The smart cameras and their architectures will be in the first section. Following sections contain pose estimation (position estimation) and texture analysis.

2.1 SMART CAMERA AND ITS ARCHITECTURES IN LITERATURE

2.1.1 Smart Cameras as Embedded Systems

Smart Cameras can be expressed as high performance embedded systems combining video sensing, video processing, and communicating within a single device. Wayne Wolf and colleagues described the required computing and communication performance and the real-time and quality-of-service (QoS) requirements of the image-processing algorithms executed on a single embedded smart camera (Wolf et al., 2002)

An important study on gesture recognition with real-time smart cameras is designed by Lu (Lu, 2004). The designed smart camera project can be implemented not only for gesture

recognition but also detecting and tracking of objects. They proposed two background elimination algorithms in the dissertation to increase the performance of smart camera systems in changing background and varying lighting condition environment. The software implementation of the smart camera systems is then taken up, including the selection of platforms and the optimization process that leads to a five times speedup. New hardware architecture is then proposed that it can provide more processing power than current platforms. The new architecture exhibits more than ten times average speedup over a traditional architecture.

Foote and Kimber proposed a FlyCam practical panoramic video and automatic camera control (Foote and Kimber, 2000). They improve computationally and substantially inexpensive methods for panoramic video imaging. Combining images, digitally, from an array of inexpensive video cameras results in a wide-field panoramic camera from inexpensive ready to use hardware. They present methods that both correct lens distortion and seamlessly merge images into a panoramic video image. Electronically selecting a region of this, results in a rapidly steerable “virtual camera”. Since the camera is fixed with respect to the background, simple motion analysis can be used to track objects and people of interest. They present methods of motion analysis and algorithms for automatic camera control that imitate the behavior of a human operator, using inexpensive and widely available hardware.

Sato et al. present a paper which describes the design and implementation of a hybrid intelligent surveillance system consisting of an embedded system and a personal computer PC-based system (Sato et al., 2006). The embedded system performs some of the image processing tasks and sends the processed data to a PC. The PC tracks persons and recognizes two-person interactions by using a grayscale side-view image sequence captured by a stationary camera.

Since the computer vision and image processing algorithms need computational powered computer systems, there are various architectures based on this approach. The best known smart camera architectures will be explained in this subsection.

2.1.2 Philips “Inca” Camera Architecture

The company of Philips developed a smart camera for machine vision. The Philips Inca (Intelligent Camera) is widely used in industry, automotive and surveillance systems. . The role of a digital intelligent camera in automating industrial photogrammetry is presented by Dold (Dold, 1998) and a face recognition system is studied with Philips Inca (Broers et al., 2004). This camera houses a CMOS sensor, a parallel processor for pixel crunching and a DSP for the high level programs. The architecture will be explained with the application of face recognition.

Harry Broers et al. proposed a face recognition algorithm with Philips SmartCam (Broers et al., 2004). Face recognition is divided into two parts. Face detection and face recognition. The detection part is face-oriented (high level processing). It finds the faces in the scene. In order to reduce the amount of work, the image needs to be pre-processed by a number of low level operations. These operations are at pixel level. This allows massive data-level parallelism. Thus, detection part involves low level and high level image processing (IP).

Figure 2.1 Philips Inca camera.

The recognition part uses high level IP and only works on a few faces per second. On the other hand, it has a high amount of operations in an iterative way while a database is scanned.

The different points of view of these algorithms tasks (low and high level IP) need a dual processor approach. The low level IP approach of the face detection part is mapped on a massively parallel processor “Xetal” working in SIMD (Single Instruction Multiple Data) mode (Hjelmas and Low, 2001). The high level IP approach of the detection and recognition part is mapped on a high-performance fully programmable DSP core “Trimedia” (Trimedia, 2003).

Two processors can be simply connected in series (in Figure 2.2). The Xetal performs face detection pre-processing. The Xetal is a low power high performance digital signal processor (Kleihorst et al., 2001). The Trimedia performs the actual face detection and recognition. For more information about the implementation of Trimedia processor one can look at the study of Slavenburg (Slavenburg, 1996). For the face detection, Haar-Face detection algorithm is initially applied to captured image (Leinhart and Maydt, 2002). The initial point of the problem is to detect (segment) the skin colored regions in the image (Majoor, 2000). This algorithm is highly talented to detect even hand drawn faces in the different light conditions. Before the image is transferred to the DSP (Trimedia) image is pre-processed. First Xetal converts the RGB colored image to a gray level image. Then, Xetal performs lighting correction to improve the quality of the image. Xetal also performs canny edge processing to reduce the number of faces candidates (Viola and Jones, 2001). Consequently, Xetal computes two so-called integral images as described before. One for the light corrected image and one for the canny edge image. The Trimedia takes the original gray level image and these two integral images from the three communication channels. As it is seen, Trimedia processor saves valuable time by using a parallel processor. After all possible face candidates are obtained, a grouping algorithm is applied to reduce a group of face candidates into one positive detection system. After the faces on the captured images are detected, the face recognition is performed.

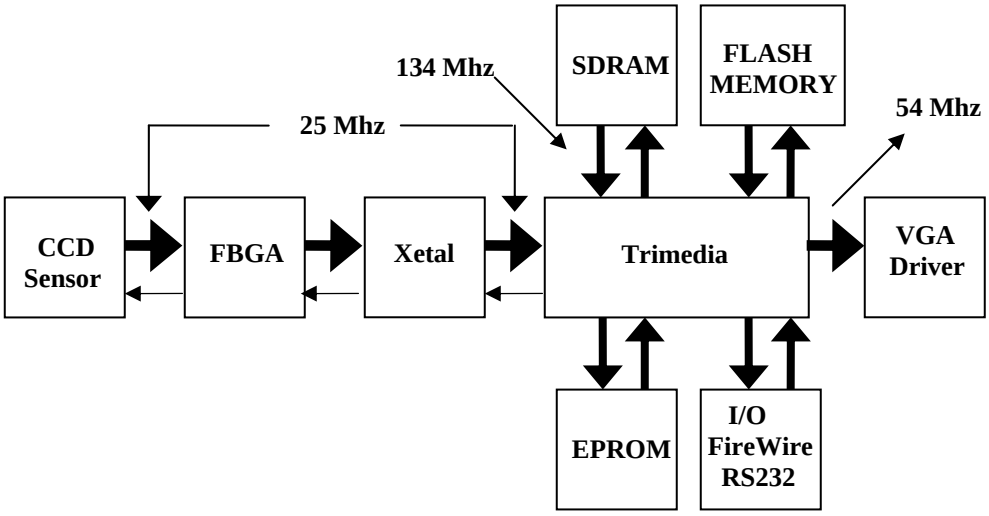


Figure 2.2 Philips Inca architecture.

Only the normalized images are used for recognition, namely the eyes and part of the nose region, covering 96x40 pixels. Hairline, mouth and ears are avoided as they can differ because of changing hairstyle, background, shaving conditions and moving lips. The registered images are forwarded directly to the input of the neural network in the recognition phase after being normalized in gray-level. For face recognition, a Radial Basis Function (RBF) neural network is used.

Hammerstrom and Lulich present the design rationale for CNAPS, a specialized one-dimensional (1-D) processor array developed by Adaptive Solutions Inc. (Hammerstrom and Lulich, 1996). They discuss the problem of Amdahl's law which severely constrains special-purpose architectures (Gustafson, 1998). They also discussed specific architectural decisions such as the kind of parallelism, the computational precision of the processors, on-chip versus off-chip processor memory, and-most importantly-the interprocessor communication architecture. They argue that, for their particular set of applications, a 1-D architecture gives the best “bang for the buck”, even when compared to the more traditional two-dimensional (2-D) architecture. Secondly, they describe how several simple algorithms map to the CNAPS array. The CNAPS array is described in the Hammerstrom and his

colleagues book (Hammerstrom, 1993). Their results show that the CNAPS 1-D array offers excellent performance over a range of IP algorithms. They also briefly look at the performance of CNAPS as a pattern recognition engine because many image processing and pattern recognition problems are closely related.

2.1.3 Scalable Smart Camera Architecture

Michael Bramberger et al. proposed that "The Scalable Smart Camera Architecture" is divided into two parts as Hardware and Software (Bramberger et al., 2006). They suggest a hardware architecture as seen in Figure 2.3.

The video sensor represents the first stage in the smart cameras overall data flow. The sensor captures incoming light and transforms it into electrical signals that can be transferred to processing unit. A CMOS sensor best fulfills the requirements for a video sensor. These sensors feature high dynamics due to their logarithmic characteristics and provide on-chip ADCs and amplifiers.

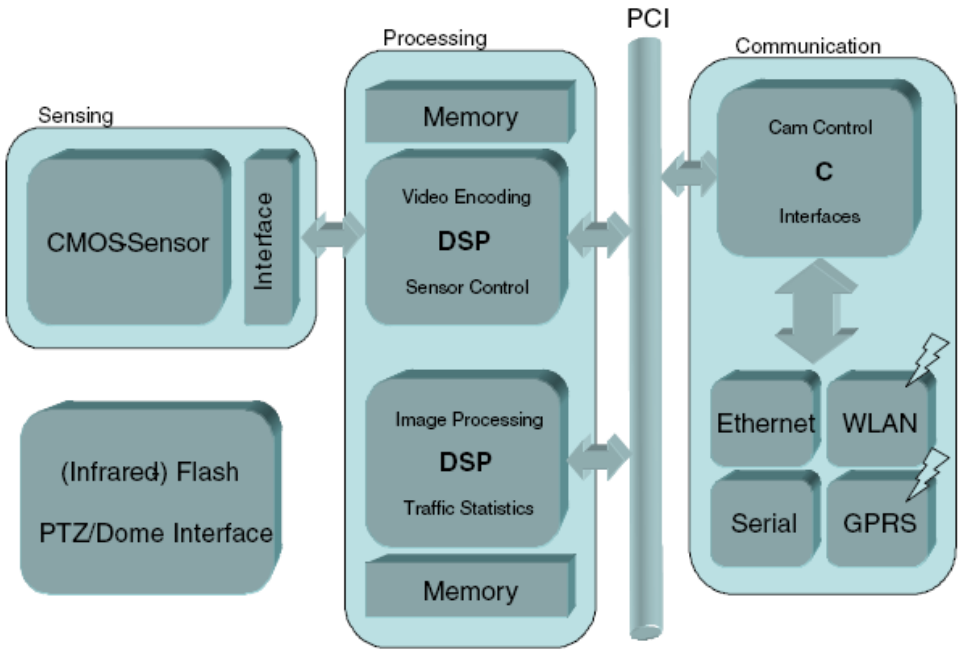


Figure 2.3 Hardware architecture of the scalable smart camera.

The second stage in the overall data flow is the processing unit. Due to the high performance on-board image and video processing, the requirements on the computing performance are very high. A rough estimation results in 10 GIPS computing performance. These performance requirements together with the various constraints of the embedded system solution are fulfilled with digital signal processors (DSP). The smart camera is equipped with two TMS320C6415 DSPs from Texas Instruments (TI) running at 600 MHz (Markandey, 2002). Both DSPs are loosely coupled via the on-board PCI bus, while each processor is connected to its own local memory. The various tasks are statically mapped to the DSPs to avoid overhead induced by a global scheduler. The video sensor is connected to one DSP via a FIFO memory to relax the timing between sensor and DSP. The image is then transferred into the DSP's external memory and via the PCI bus to the other components (DSP and network processor).

The final stage of the smart camera's overall data flow is represented by the communications unit. The unit is primarily composed of an Intel XScale IXP425 processor which directly manages most onboard communications like PCI, Ethernet, USB, and serial communications. Wireless LAN and GSM/GPRS are connected using a generic interface. This interface enables the connection of various peripherals or communication systems with low effort.

The software architecture of the smart camera is basically divided into two parts; DSPs configured to basically run computation intensive tasks like video compression (MPEG-4 simple profile), image analysis, or some parameters calculation. Since reconfigurability and scalability are important issues, the DSPs are running on Texas Instruments' Reference Framework 5 (RF5) in combination with TI's XDAIS algorithm standard, which enables the exchange and reconfiguration of algorithms during runtime. All reconfiguration and control actions are controlled by the system control processor. XScale processor is primarily used for system control and communication purposes. Therefore a standard operating system makes the development of internal and external communication services like web-services, proprietary control connections, or PCI- communications easier. Hence (Embedded-) Linux has been chosen to be used. XScale processor is primarily used for system control and communication purposes.

2.2 LITERATURE SURVEY ON POSE ESTIMATION

Pose estimation and object matching are two classical problems in pattern recognition and computer vision. Pose (Position) of rotated object is calculated relative to an original or known object. Similarly, in object matching an object database is needed such that the unknown object is matched to one candidate object in this database. These methods are widely used automated manufacturing and robotics. Since we are looking for the rotation estimation, we will focus on this pose estimation rather than object matching.

Cem Unsalan proposed a least squares based parameter estimation method depending on implicit polynomial (IP) representations directly (Unsalan, 2007). This approach can be applied to both 2D and 3D implicit polynomial representation. Implicit Polynomial representations have many desirable properties over parametric representations. In this paper, the IP fit method which is suggested by Tasdizen was used. (Tasdizen, 2000). Object representation with IP is based on finding appropriate parameters. The pose estimation and object matching methods depends upon IP representation of the original and rotated objects. Second order and fourth order IP is used in this study. It is stated that the increment of the fit order start to capture the effects of the disturbance more, so increasing the degree of the polynomial decrease the performance. It is not possible to give a specific fit order that performs best under colored noise, missing points, or affine transformation. Finally, it is claimed that fit order should be neither high nor low.

Marques et al. proposed an algorithm for the optimal alignment of a pair of 2D shapes defined by point sequence (Marques et al., 1997). The proposed algorithm provides closed-form expressions for the estimation of the initial point, scale, and pose parameters. In this study, a shape is taken as a reference image. Then, this image is rotated with a certain degree and the rotation of the image is estimated. The experimental results of the study show that the proposed method provides reliable estimates for the unknown parameters even under strong shape deformations. For example, no error is observed in the initial point estimates in the tests presented in the study. It is also concluded that the shape alignment

algorithm proposed in the paper performs better than the Hough transform when applied to the same data (Illingworth and Kittler, 1988).

An important problem is that a mobile robot exploring an unknown environment has no absolute frame of reference for its position, other than features it detects through its sensors. Lu and Milios developed two new iterative algorithms to register a range scan to a previous scan so as to compute relative robot positions in an unknown environment (Lu and Milios, 1997). The first algorithm is based on matching data points with tangent directions in two scans and minimizing a distance function in order to solve the displacement between the scans. The second algorithm establishes correspondences between points in the two scans and then solves the point-to-point least-squares problem to compute the relative pose of the two scans.

An approach for pose estimation based on multi-camera system with known internal camera parameters is proposed by Frahm et al (Frahm et al., 2004). They only assume for the multi-camera system that the cameras of the system have fixed orientations and translations between each other. In contrast to existing approaches for reconstruction from multi-camera systems they introduce rigid motion estimation for the multi-camera system itself using all information of all cameras simultaneously even in the case of non overlapping views of the cameras. Furthermore they introduce a technique to estimate the pose parameters of the multi-camera system automatically.

Real-time 3-D pose estimation is useful in a variety of situations. In manufacturing environments, it can be used in feedback control loops to allow a mechanism (e.g. a robot) to perform an operation (e.g. grasping) on a moving part. In the area of Human Computer Interaction (HCI), real-time pose estimation can be useful for tracking movements of a body part for subsequent interpretation as input to a computer. In medicine, a variety of problems involve the need to register pre-operative, volumetric data with the corresponding anatomy of the actual patient. A real-time 3-D pose estimation approach is described in the paper which is written by David A. Simon et al. (Simon et al, 1994).

To be able to increase the performance of the pose estimation algorithms, Gregory Shakhnarovich et al. proposed fast pose estimation algorithm with parameter sensitive

hashing (Shakhnarovich et al., 2003). They present a new algorithm that learns a set of hashing functions that efficiently index examples relevant to a particular estimation task. Their algorithm extends a recently developed method for locality-sensitive hashing, which finds approximate neighbors in time sublinear in the number of examples. This method depends critically on the choice of hash functions; they show how to find the set of hash functions that are optimally relevant to a particular estimation problem. Experiments demonstrate that the resulting algorithm, which is called Parameter-Sensitive Hashing, can rapidly and accurately estimate the articulated pose of human figures from a large database of example images.

2.3 LITERATURE SURVEY ON TEXTURE ANALYSIS

Texture analysis methods are used in many applications. These applications can counted as medical image processing, automated inspection, document processing, and remote sensing. There are many researches in literature about texture analysis. Texture analysis studies such as texture classification, segmentation, defect detection and texture analysis methods will be given in this section.

2.3.1 Texture Classification

Almost every computer vision and image processing book contains significant part devoted to texture analysis (Sonka et al., 1993). Irene Epifanio and Guillermo Ayala proposed a global framework for texture classification based on random closed set theory (Epifanio and Ayala, 2002). In this approach, a binary texture is considered as an outcome of a random closed set. Some distributional descriptors of this stochastic model are used as texture features in order to classify the binary texture. If a grayscale texture has to be classified, then the original texture is reduced to a multivariate random closed set where each component (a different random set) corresponds with those pixels verifying a local property. Again, some functional descriptors of the multivariate random closed set defined from the texture can be used as texture features to describe and classify the grayscale texture.

Chellappa and Chatterjee present two feature extraction methods for the classification of textures using two-dimensional (2-D) Markov random field (MRF) models (Chellappa, and Chatterjee, 1985). It is assumed that the given $M \times M$ texture is generated by a Gaussian MRF model. In the first method, the least square (LS) estimates of model parameters are used as features. In the second method, using the notion of sufficient statistics, it is shown that the sample correlations over a symmetric window including the origin are optimal features for classification. Simple minimum distance classifiers using these two feature sets yield good classification accuracies for a seven class problem.

Porter and Canagarajah proposed novel feature extraction schemes for texture classification, which are wavelet Gabor filter and GMRF based schemes (Porter and Canagarajah, 1997). The schemes are shown to give a high level of classification accuracy compared to most existing schemes, using both fewer features (four) and a smaller area of analysis (16 x 16). Furthermore, unlike most existing schemes, the proposed schemes are shown to be rotation invariant and demonstrate a high level of robustness to noise. The performances of the three schemes are compared, indicating that the wavelet-based approach is the most accurate, exhibits the best noise performance and has the lowest computational complexity.

Another classification method using color, texture and regions is presented by Cheng and Chen (Cheng and Chen, 2003). Image-based features related to color and local edge patterns are used to prune irrelevant database images for each query image. The proposed region matching is then applied to find the match to the query image from among the set of candidate images in the database. The dissimilarity of each pair of images can be calculated on the basis of the matching results. Finally, all the database images in the candidate set can be sorted by ascending dissimilarity values. The main contribution of this paper is to select proper features for representing color, texture and region, which, in turn, are used to achieve effective classification results. More important, all features used in the proposed method, no matter color or texture, are presented in the simple form of histogram, yet leading to effective results.

A framework for comparing texture classification algorithms is proposed by Smith and Burns (Smith and Burns, 1997). The framework contains several suites of texture classification problems, a standard functionality for algorithms, and a method for computing a score for each algorithm. They use the framework to demonstrate the peaking phenomenon in texture classification algorithms.

2.3.2 Texture Segmentation

Sun et al. proposed a multiscale Bayesian texture segmentation algorithm that is based on a complex wavelet domain hidden Markov tree (HMT) model and a hybrid label tree (HLT) model (Sun et al., 2004). The HMT model is used to characterize the statistics of the magnitudes of complex wavelet coefficients. The HLT model is used to fuse the interscale and intrascale context information. In the HLT, the interscale information is fused according to the label transition probability directly resolved by an EM algorithm. The intrascale context information is also fused so as to smooth out the variations in the homogeneous regions.

Panda and Chatterji present a texture segmentation algorithm based on the multi-channel filtering theory (Panda and Chatterji, 1997). The channels are characterized by a bank of Gabor like tuned modulated basis filters. They chose scale changeable exponential bases of compact support to derive such filters. It is seen that the tuned modulated basis filters closely approximate the Gabor elementary function. Perfect reconstruction of the input image from its filtered images is shown. Computation and storage requirements are considerably reduced. Texture features are obtained by subjecting each (selected) filtered image to a nonlinear transformation and computing a measure of "energy" in a window around each pixel.

An adaptive computational model for texture segmentation is proposed by Caelli (Caelli, 1998). Extensions to current models for texture segmentation are presented in the study. The underlying detector (filter) mechanisms are allowed to adapt to the incoming signal in terms of their dynamical response range and associativities. This system converges on new “texton” profiles of minimal dimensionality that are used to classify texture regions by a minimum distance classifier in the texture feature space. These three

processes of convolution, cooperativity, and classification are individually analyzed and compared with some observations from human texture discrimination experiments.

Thomas Hofmann et al. present a novel optimization framework for unsupervised texture segmentation that relies on statistical tests as a measure of homogeneity (Hofmann et al., 1998) Texture segmentation is formulated as a data clustering problem based on sparse proximity data. Dissimilarities of pairs of textured regions are computed from a multiscale Gabor filter image representation. They discuss and compare a class of clustering objective functions which is systematically derived from invariance principles. As a general optimization framework, they propose deterministic annealing based on a mean-field approximation. The canonical way to derive clustering algorithms within this framework as well as an efficient implementation of mean-field annealing and the closely related Gibbs sampler are presented.

2.3.3 Pattern Regularity and Finding Structural Defects

D. Chetverikov gives a summary of our research on pattern regularity (Chetverikov, 2000). Periodic structures are perceived by humans as regular in a wide range of viewing angles. This observation motivates the development of a regularity based feature vector whose affine invariance is justified theoretically and tested experimentally. The vector is derived from the interaction map of a pattern. Several alternative but closely related definitions of the interaction map are discussed. The maximal regularity, a component of the feature vector, is shown to be consistent with human judgment on regularity. This feature can be implemented as a run filter, allowing for regularity based image filtering. Three applications of the regularity approach are presented. First, it is used for affine-invariant texture classification. Then, detection of periodic structures in aerial images is demonstrated. Finally, the texture inspection problem is addressed and structural defects are found as locations of low regularity.

Another texture defect detection method is proposed by Dimitry Chetverikov and Krisztian Gede (Chetverikov and Gede, 1997). In this paper, it is illustrated that structural texture imperfections of the diverse origin can be detected in the framework of a novel approach based on the FBIM structural filtering. The initial experiments of the study

indicate that adaptivity to variations in directionality and size is strongly desirable. The FBIM approach is especially proper for detection of structural defects in more or less regular textures. It is not efficient when applied to irregular patterns.

Sezer et al. address the raw textile defect detection problem using independent components approach with insights from human vision system (Sezer et al, 2007). Human vision system is known to have specialized receptive fields that respond to certain type of input signals. Orientation-selective bar cells and grating cells are examples of receptive fields in the primary visual cortex that are selective to periodic- and aperiodic-patterns, respectively. Regularity and anisotropy are two high-level features of texture perception, and they assume that disruption in regularity and/or orientation field of the texture pattern causes structural defects. In the paper, it is observed that independent components extracted from texture images give bar or grating cell like results depending on the structure of the texture.

2.3.4 Texture Analysis Methods

Tuceryan and Jain proposed that distinguishing the perceived qualities of texture in an image is an important step towards building mathematical models for texture (Tuceryan and Jain, 1998). The intensity variations in an image which characterize texture are generally due to some underlying physical variation in the scene (such as pebbles on a beach or waves in water). Modeling this physical variation is very difficult, so texture is usually characterized by the two-dimensional variations in the intensities present in the image. This explains the fact that no precise, general definition of texture exists in the computer vision literature. In spite of this, there are a number of intuitive properties of texture which are generally assumed to be true.

The texture analysis methods are divided into four main topics which are Statistical Methods, Geometrical Methods, Model Based Methods and Signal Processing Methods.

2.3.4.1 Statistical Methods

The use of statistical methods is very early researches proposed in machine vision literature. In statistical methods literature, there are two subtopics which are co-occurrence matrix and autocorrelation (Tuceryan and Jain, 1998).

Spatial gray level co-occurrence estimates image properties related to second-order statistics. Haralick et al. suggested the use of gray level co-occurrence matrices (GLCM) which have become one of the most well-known and widely used texture features (Haralick et al., 1973). The GxG gray level co-occurrence matrix P_d for a displacement vector $d = (d_x, d_y)$ is defined as follows. The entry (i, j) of P_d is the number of occurrences of the pair of gray levels i and j which are a distance d apart. Formally, it is given as

$$P_d(i, j) = |\{((r, s), (t, v)): I(r, s) = i, I(t, v) = j\}| \quad (2.3.1)$$

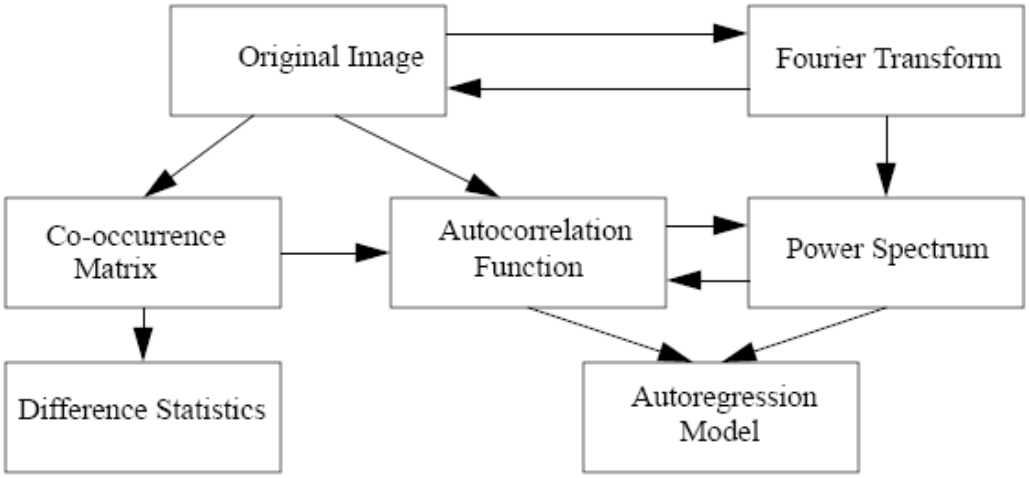


Figure 2.4 Interrelation between various second-order statistics and input image.

Where $(r, s),(t, v) \in N \times N$, $(t, v)=(r+d_x, s+d_y)$ and $|.|$ is the cardinality of a set.

As a property of a co-occurrence matrix, it is not symmetrical but a symmetric matrix can be computed as;

$$P = P_d + P_{-d} \tag{2.3.2}$$

They proposed a number of useful texture features that can be computed from the co-occurrence matrix. Table 2.1 lists some of these features. In the list, μ_x and μ_y, are the means and, σ_x and σ_y are the standard deviations of $P_d(x)$ and $P_d(y)$ where,

$$P_d(x) = \sum_{j=0} \mathrm{P_d}(x, j), \qquad P_d(y) = \sum_{i=0} \mathrm{P_d}(i, y) \tag{2.3.3}$$

The co-occurrence matrix features suffer from a number of difficulties. There is no well established method of selecting the displacement vector d and computing and computing co-occurrence matrices for different values of d is not feasible. For a given d, a large number of features can be computed from the co-occurrence matrix. This means that some sort of feature selection method must be used to select the most relevant features. The co-occurrence matrix-based texture features have also been primarily used in texture classification tasks and not in segmentation tasks.

Table 2.1 Some texture features from gray level co-occurrence matrices.

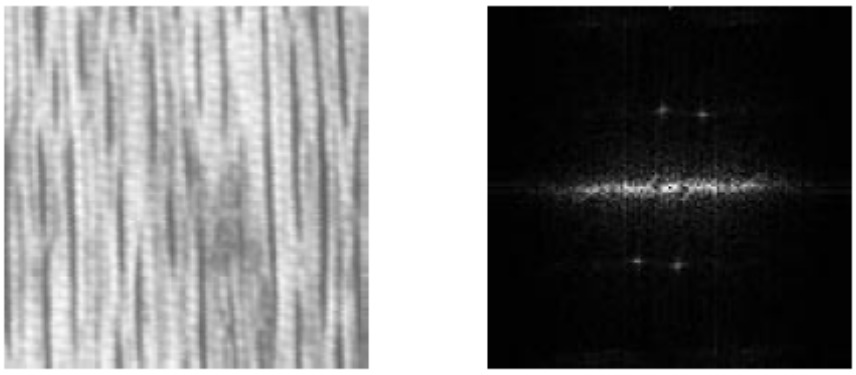
Texture Feature	**Formula**
Energy	$\sum_i \sum_j P_{\mathbf{d}}^2(i, j)$
Entropy	$-\sum_i \sum_j P_{\mathbf{d}}(i, j) \log P_{\mathbf{d}}(i, j)$
Contrast	$\sum_i \sum_j (i-j)^2 P_{\mathbf{d}}(i, j)$
Homogeneity	$\sum_i \sum_j \frac{P_{\mathbf{d}}(i, j)}{1 + \lvert i-j \rvert}$
Correlation	$\frac{\sum_i \sum_j (i-\mu_x)(j-\mu_y) P_{\mathbf{d}}(i, j)}{\sigma_x \sigma_y}$

An important property of many textures is the repetitive nature of the placement of texture elements in the image. The autocorrelation function of an image I (x, y) is defined follows.

$$\rho(x,y) = \frac{\sum_{u=0}^{N}\sum_{v=0}^{N} I(u,v)I(u+x,v+y)}{\sum_{u=0}^{N}\sum_{v=0}^{N} I^2(u,v)} \qquad (2.3.4)$$

This function is related to the size of the texture primitive (i.e. the fineness of the texture). If the texture is coarse, then the autocorrelation function will drop off slowly. Otherwise, it will drop off very rapidly. For regular textures, the autocorrelation function will exhibit peaks and valleys.

The autocorrelation function is also related to the power spectrum of the Fourier transform (see Figure 2.5). Consider the image function in the spatial domain I(x, y) and its Fourier Transform F (u, v). The quantity $|F(u,v)|^2$ is defined as the power spectrum where |.| is modulus of the complex number.



(a) (b)

Figure 2.5 Texture features from the power spectrum;
(a) a texture image (b) power spectrum of the image.

2.3.4.2 Geometrical Methods

The class of texture analysis methods that falls under the heading of geometrical methods is characterized by their definition of texture as being composed of “texture elements” or primitives (Tuceryan and Jain, 1990). The method of analysis usually depends upon the geometric properties of these texture elements. Once the texture elements are identified in the image, there are two major approaches to analyzing the texture. One computes statistical properties from the extracted texture elements and utilizes these as texture features. The other tries to extract the placement rule that describes the texture. The latter approach may involve geometric or syntactic methods of analyzing texture.

Tuceryan and Jain proposed the extraction of texture tokens by using the properties of the Voronoi tessellation of the given image (Tuceryan and Jain, 1990). Voronoi tessellation has been proposed because of its desirable properties in defining local spatial neighborhoods and because the local spatial distributions of tokens are reflected in the shapes of the Voronoi polygons.

Zucker has proposed a method in which he regards the observable textures (real textures) as distorted versions of ideal textures (Zucker, 1976). The placement rule is defined for the ideal texture by a graph that is isomorphic to a regular or semi regular tessellation. These graphs are then transformed to generate the observable texture. Which of the regular tessellations is used as the placement rule is inferred from the observable texture. This is done by computing a two-dimensional histogram of the relative positions of the detected texture tokens.

Another approach to modeling texture by structural means is described by Fu (Fu, 1982). In this approach the texture image is regarded as texture primitives arranged according to a placement rule. The primitive can be as simple as a single pixel that can take a gray value, but it is usually a collection of pixels. The placement rule is defined by a tree grammar. A texture is then viewed as a string in the language defined by the grammar whose terminal symbols are the texture primitives. An advantage of this method is that it can be used for texture generation as well as texture analysis.

2.3.4.3 Model Based Methods

Model based texture analysis methods are based on the construction of an image model that can be used not only to describe texture, but also to synthesize it. The model parameters capture the essential perceived qualities of texture.

Ohanian and Dubes have studied the performance of various texture features (Ohanian and Dubes, 1992). They studied the texture features with the performance criteria "which features optimized the classification rate?" They compared four fractal features, sixteen co-occurrence features, four Markov random field features, and Gabor features. In this study, it is used Whitney's forward selection method for feature selection. The evaluation was done on four classes of images: Gauss Markov random field images, fractal images, leather images, and painted surfaces (Whitney, 1971).

2.3.4.4 Signal Processing Methods

Psychophysical research has given evidence that the human brain does a frequency analysis of the image. Texture is especially suited for this type of analysis because of its properties. This subsection will review the various techniques of texture analysis that rely on signal processing techniques. Most techniques try to compute certain features from filtered images which are then used in either classification or segmentation tasks.

Spatial domain filters are the most direct way to capture image texture properties. Earlier attempts at defining such methods concentrated on measuring the edge density per unit area. Fine textures tend to have a higher density of edges per unit area than coarser textures. The measurement of edgeness is usually computed by simple edge masks such as the Robert's operator or the Laplacian operator. The two orthogonal masks for the Robert's operator and one digital realization of the Laplacian are given below.

Robert Operators,

$$M_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \qquad M_2 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, \tag{2.3.5}$$

Laplacian Operator,

$$M_1 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}, \tag{2.3.6}$$

The edgeness measure can be computed over an image area by computing a magnitude from the responses of Roberts's masks or from the response of the Laplacian mask.

Malik and Perona proposed spatial filtering to model the preattentive texture perception in human visual system (Malik and Perona, 1990). Their proposed model consists of three stages: (a) convolution of the image with a bank of even-symmetric filters followed by half-wave rectification, (b) inhibition of spurious responses in a localized area, and (c) detection of the boundaries between the different textures. The even-symmetric filters they used consist of differences of offset Gaussian (DOOG) functions. The half-wave rectification and inhibition (implemented as leaders-take-all strategy) are methods of introducing nonlinearity into the computation of texture features. Nonlinearity is needed in order to discriminate texture pairs with identical mean brightness and identical second-order statistics. The texture boundary detection is done by straightforward edge detection method applied to the feature images obtained from stage (b). This method works on a variety of texture examples and is able to discriminate natural as well as synthetic textures with carefully controlled properties.

Unser and Eden have also looked at texture features that are obtained from spatial filters and a nonlinear operator (Unser and Eden, 1990). Reed and Wechsler review a number of spatial/spatial frequency domain filter techniques for segmenting textured images (Reed and Wechsler, 1990).

2.3.4.5 Gabor and Wavelet Methods

Gabor and Wavelet methods are the most commonly methods used in texture analysis (Tuceryan and Jain, 1998). The Gabor and Wavelet method is produced from the window

Fourier Transform. The Fourier transform is an analysis of the global frequency content in the signal. Many applications require the analysis to be localized in the spatial domain. This is usually handled by introducing spatial dependency into the Fourier analysis. The classical way of doing this is through the window Fourier Transform. The window Fourier Transform (or short-time Fourier Transform) of a one-dimensional signal f(x) is defined as follows:

$$F_w(u,\zeta) = \int_{-\infty}^{\infty} f(x) w(x-\zeta) e^{-j2\pi ux} dx \tag{2.3.7}$$

When the window function is Gaussian, the transform becomes a Gabor transform. The limits on the resolution in the time and frequency domain of the window Fourier Transform are determined by the *time-bandwidth product* or the *Heisenberg uncertainty inequality* given by:

$$\Delta t \Delta u \geq \frac{1}{4\pi} \tag{2.3.8}$$

Once a window is chosen for the window Fourier Transform, the time-frequency resolution is fixed over the entire time-frequency plane. To overcome the resolution limitation of the window Fourier Transform, one lets the Δt and Δu vary in the time-frequency domain. Intuitively, the time resolution must increase as the central frequency of the analyzing filter is increased. That is, the relative bandwidth is kept constant in a logarithmic scale. This is accomplished by using a window whose width changes as the frequency changes.

Recall that when a function $f(x)$ is scaled in time by a which is expressed as $f(at)$, the function is contracted if $a > 1$ and is expanded when $a < 1$. Using this property, the wavelet transform can be written as:

$$W_{f,a}(u,\zeta) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} f(t) h^* \left(\frac{t-\zeta}{a} \right) dt \tag{2.3.9}$$

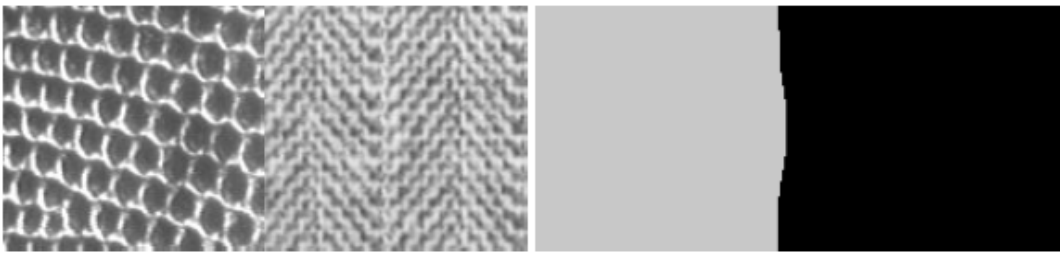


Figure 2.6 The segmentation results using moment based texture features.

(a) A texture pair consisting of reptile skin and herringbone pattern from the Brodatz album. (b) The resulting segmentation.

Here, the impulse response of the filter bank is defined as the scaled versions of the same prototype function $h(t)$. Now, setting in Equation (2.2.8);

$$h(t) = w(t)e^{-j2\pi ut} \tag{2.3.10}$$

The wavelet model for the texture analysis is obtained. Usually scaling factor *a* will be based on the frequency of the filter.

Daugman proposed the use of Gabor filters in the modeling of the receptive fields of simple cells in the visual cortex of some mammals (Daugman, 1980). The proposal to use the Gabor filters in texture analysis was made by Turner (Turner, 1986). Jain and Farrokhnia used it successfully in segmentation and classification of textured images (Jain and Farrokhnia, 1991). Gabor filters have some desirable optimality properties. Daugman showed that for two dimensional Gabor functions, the uncertainty relations $\Delta x \Delta u \geq \pi / 4$ and $\Delta y \Delta v \geq \pi / 4$ attain the minimum value. Here Δx and Δy are effective widths in the spatial domain. Furthermore, Δu and Δv are effective bandwidths in the frequency domain.

A two-dimensional Gabor function consists of a sinusoidal plane wave of a certain *frequency* and *orientation* modulated by a Gaussian envelope. It is given by,

$$f(x, y) = \exp\left\{-\frac{1}{2}\left[\frac{x^2}{\sigma_x^2} + \frac{y^2}{\sigma_y^2}\right]\right\}\cos(2\pi u_0 x + \phi) \tag{2.3.11}$$

where u_0 and ϕ are the frequency and the phase of sinusoidal wave. The values σ_x and σ_y are the sizes of the Gaussian envelope in the *x* and *y* directions, respectively. The Gabor function at an arbitrary orientation θ_0 can be obtained from Equation (2.3.11) by a rigid rotation of the *x*-*y* plane by θ_0.

The Gabor filter is a frequency and orientation selective filter. This can be seen from the Fourier domain analysis of the function. When the phase ϕ is 0, the Fourier transform of the resulting even-symmetric Gabor function $f(x, y)$ is given by

$$F(u,v) = A\left\{\exp\left(-\frac{1}{2}\left[\frac{(u-u_0)^2}{\sigma_u^2}+\frac{v^2}{\sigma_v^2}\right]\right)+\exp\left(-\frac{1}{2}\left[\frac{(u+u_0)^2}{\sigma_u^2}+\frac{v^2}{\sigma_v^2}\right]\right)\right\} \qquad (2.3.12)$$

where $\sigma_u = 1/(2\pi\sigma_x), \sigma_v = 1/(2\pi\sigma_y)$, and $A = 2\pi\sigma_x\sigma_y$. This function is real-valued and has two lobes in the spatial frequency domain, one centered around u_0 and another centered around $-u_0$. For a Gabor filter of a particular orientation, the lobes in the frequency domain are also appropriately rotated.

CHAPTER 3

FOURIER TRANSFORM BASED ROTATION ESTIMATION ALGORITHM

To be able to estimate rotation angle on the target image, FFT (Fast Fourier Transform) based approximation will be discussed in this section. Also, the definitions of Fourier Transform (FT), Fast Fourier Transform (FFT) and Discrete Time Fourier Transform (DFT) is discussed in this chapter.

The Fourier transform is a mathematical method which is used to expand signals into a spectrum of sinusoidal components to provide signal analysis and system performance. In certain applications the Fourier transform is used for spectral analysis, or for spectrum shaping that adjusts the relative contributions of different frequency components in the filtered result. In other applications, the Fourier transform is important for its ability to decompose the input signal into uncorrelated components, so that signal processing can be more effectively implemented on the individual spectral components. Decorrelating properties of the Fourier transform are important in frequency domain adaptive filtering, subband coding, image compression, and transform coding.

Classical Fourier methods such as the Fourier series and the Fourier integral are used for continuous-time (CT) signals and systems, i.e., systems in which the signals are defined at all values of t on the continuum $-\infty < t < \infty$. A more recently developed set of discrete Fourier methods, including the discrete-time (DT) Fourier transform and the Discrete

Fourier transform (DFT), are extensions of basic Fourier concepts for DT signals and systems. A DT signal is defined only for integer values of n in the range of $-\infty < n < \infty$.

Fast Fourier Transform is the decimation-in-time method to speed the computation of FT or DFT of a sequence. This method is one of breaking the N-point transform into two (N/2)-point transforms, breaking each (N/2)-point transform into two (N/4)-point transforms, and continuing the above process until we obtain the two point transform. FFT program, which is written in Matlab, can be found in Appendix A.

3.1 2D DISCRETE FOURIER TRANSFORM (DFT) REPRESENTATION

The 2D Discrete Fourier Transform is the series expansion of an image function in terms of "cosine" image basis function. The definition of the transform and its inverse are given as;

$$F(u,v) = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) e^{-j2\pi(ux+vy)/N} \tag{3.1.1}$$

$$f(x,y) = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} F(u,v) e^{+j2\pi(ux+vy)/N} \tag{3.1.2}$$

Brayer illustrated the basis functions for the Fourier Transforms (FT) in his study (Brayer, 2006). The main idea of the FT is to represent all images as a summation of cosine-like images (see Figure 3.1).

The images are a pure horizontal cosine of 8 cycles and a pure vertical cosine of 32 cycles. It is very important to figure out that FT for each image has a single component, represented by 2 bright spots symmetrically placed about the center of the FT image. The center of the image is the origin of the frequency coordinate system. The u-axis runs left to right through the center and represents the horizontal component of frequency. The v-axis runs bottom to top through the center and represents the vertical component of frequency. In both cases there is a dot at the center that represents the (0, 0) frequency term or average

value of the image. Images usually have a large average value (like 128) and lots of low frequency information so FT images usually have a bright blob of components near the center. Notice that high frequencies in the vertical direction will cause bright dots away from the center in the vertical direction. And those high frequencies in the horizontal direction will cause bright dots away from the center in the horizontal direction.

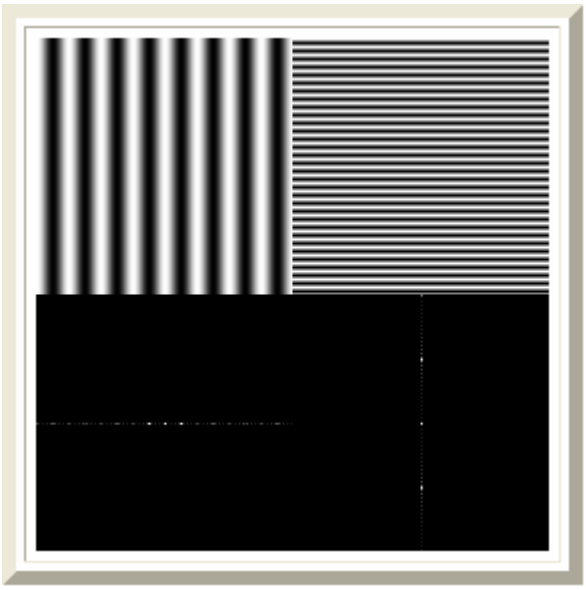


Figure 3.1 High and low frequency information of sinusoidal pattern image.

There is another example in Figure 3.2, the image composed of 2D cosines with both horizontal and vertical components. The one on the left has 4 cycles horizontally and 16 cycles vertically. The one on the right has 32 cycles horizontally and 2 cycles vertically (Note: It can be seen a gray band when the function goes through gray = 128 which happens twice/cycle). It is obvious to see the symmetry on the FTs of the images. For all real images the FT is symmetrical about the origin so the 1st and 3rd quadrants are the same and the 2nd and 4th quadrants are the same. If the image is symmetrical about the x-axis (as the cosine images are) 4-fold symmetry results.

After the basis function explanations, we would like to show the rotation and edge effects on the FT of the images.

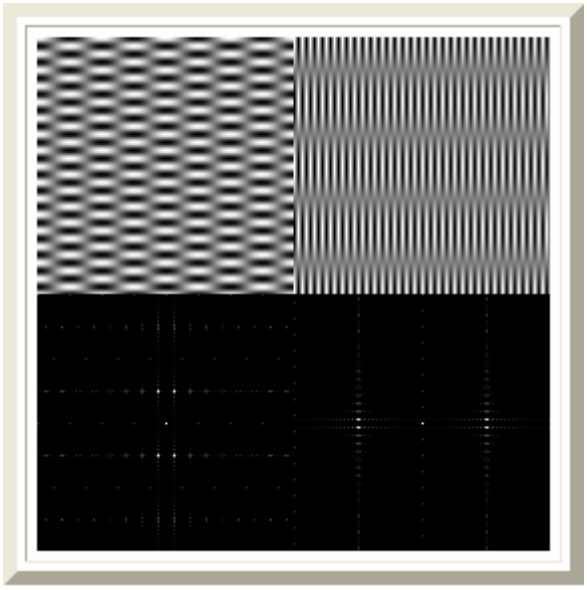


Figure 3.2 FT of a more complicated pattern image.

3.2 ROTOTION AND EDGE EFFECTS ON FOURIER TRANSFORM

In the text, whenever FT (Fourier Transform) is mentioned, FFT (Fast Fourier Transform) should be understood since FFT is used for speeding up the calculations. Rotation of the image corresponds the rotation of its FT. To prove this;

A rotation matrix is defined as;

$$R(\theta)=\begin{bmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{bmatrix}, \qquad R(\theta)=\begin{bmatrix}\cos\theta & \sin\theta\\ -\sin\theta & \cos\theta\end{bmatrix} \tag{3.2.1}$$

$$\begin{bmatrix}x'\\ y'\end{bmatrix}=\begin{bmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{bmatrix}\begin{bmatrix}x\\ y\end{bmatrix}, \qquad \begin{bmatrix}x\\ y\end{bmatrix}=\begin{bmatrix}\cos\theta & \sin\theta\\ -\sin\theta & \cos\theta\end{bmatrix}\begin{bmatrix}x'\\ y'\end{bmatrix} \tag{3.2.2}$$

$$\begin{aligned}x &= x'\cos\theta + y'\sin\theta\\ y &= -x'\sin\theta + y'\cos\theta\end{aligned} \tag{3.2.3}$$

From the Fourier transform definition;

$$F(u',v') = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x',y') e^{-j2\pi(ux+vy)/N} \tag{3.2.4}$$

$$\begin{aligned} ux + vy &= u(x'\cos\theta + y'\sin\theta) + v(-x'\sin\theta + y'\cos\theta) \\ ux + vy &= x'(u\cos\theta - v\sin\theta) + y'(u\sin\theta + v\cos\theta) \\ ux + vy &= u'x' + v'y' \end{aligned} \tag{3.2.5}$$

Therefore, the rotation of the image results in equivalent rotation of its FT.

$$F(u',v') = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x',y') e^{-j2\pi(u'x'+v'y')/N} \tag{3.2.6}$$

To be able to see the edge affects on the Fourier transform let us analyze the image shown in Figure 3.3.

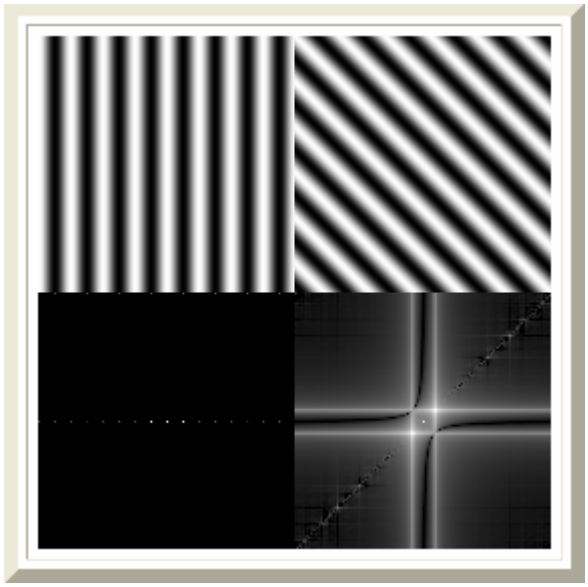


Figure 3.3 Edge effect.

As it is seen in Figure 3.3, the 45 degree rotation has very complicated and undesired result on the FT of the image. The horizontal and vertical components occur on the FT. This event is known as strong “edge effect” between neighbors of a periodic array. Edge

effects are significantly reduced by "windowing" the image with a function that slowly tapers off to a medium gray at the edge. The result can be seen in Figure 3.4

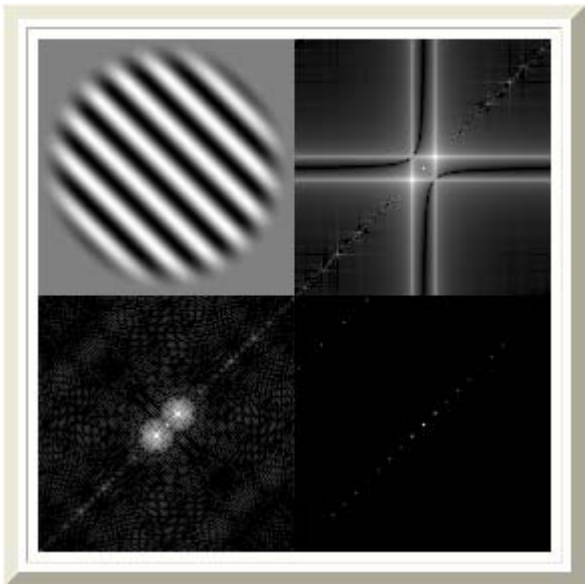


Figure 3.4 The windowed image and its FT.

The ideal case at right-down figure and the real FT can be seen on the left-down figure. Although it does not give perfect reduction, it significantly eliminates the horizontal and vertical components. There are several kinds of data windowing methods. These are Triangle (Fejer, Bartlet), Hamming window, Blackman window, Blackman-Harris window, Centered Gaussian, Centered Kaiser-Bessel window (Harris et al., 1978).

3.3 TEXTURE IMAGE AND ITS FOURIER TRANSFORM (FT) ANALYSIS

Texture image pattern is generally in a periodic format. Therefore, Fourier Transform of the image also has the periodic structure. In this subsection, we will explain how FT is used to estimate the rotation.

3.3.1 Image Size Modification

Before we compute FT of the image, we must modify the image size to make sure the size of the image matrix is square. The reason of size modification is to comply with FFT requirements. This modification cuts the texture image from center pixel to previously defined size (n). The algorithm is as follows;

- Get the size of target image.
- Find the center of the image.
- Cut the image from center for n size width and n size height

The aim of getting the size of the target image is to find the image width and height. After the width and height is found, the center of the image is can be obtained. Finally, the image is cut from center for n sizes to keep the size of the image square.

Another important result of the size modification is to convert RGB image to Gray level image. Since we are dealing with the pattern, we can omit the color information.

3.3.2 FFT of the Texture Image

After we get the square matrix gray-level image, we take the FFT of the texture image. Then, we emphasize or de-emphasize the bright points by thresholding the FT of the image. The complete Matlab code can be found in Appendix A.

The image and its Fourier Transform can be seen in Figure 3.5. The rotation of points can be seen from the figures. To get a clear response on the figure, we invert the color from black to white.

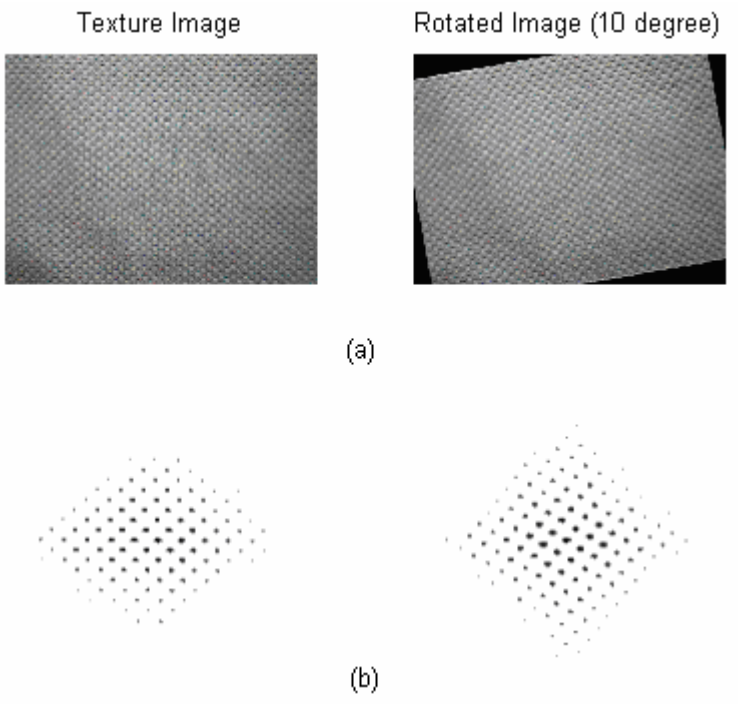


Figure 3.5 (a) Rotated texture image and (b) Its FFT.

3.3.3 Estimating the Rotation from FFT of the image

In order to estimate rotation from FFT of the image, the following algorithm is proposed.

- Define a box in the neighbor of the center image. It can be 20-25 pixel depending on the image size.
- Search and find the brightest point.
- Get the brightest point location (x and y).
- Compute the angle.

The box and searching region can be seen from the Figure 3.6. The area “I” represent the dc component region, so we omit this region while searching the brightest point. The searched area is actually “I`”. The regions of the “I” can be 3-5 pixels depending on the dc component size. When we are searching the brightest point, we are actually looking for the greatest pixel value of the FFT taken image. Then we get brightest point location and compute the rotation angle by applying the following well known formula;

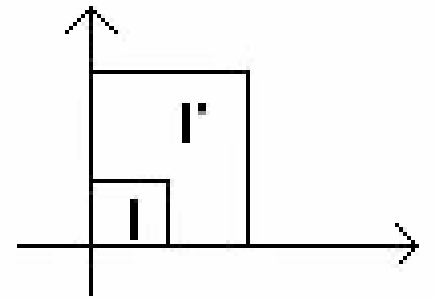


Figure 3.6 Determination of the brightest point in the searched region

$$\theta = \frac{180}{\pi} \tan^{-1}\left(\frac{y}{x}\right), \tag{3.3.1}$$

where x and y represent the brightest point location.

CHAPTER 4

STATISTICAL FEATURES BASED ROTATION ESTIMATION ALGORITHM

In order to avoid the computational difficulty of FT (Fourier Transform), we have computed different statistical features of textures, and observed how they change with texture rotation. Statistical parameters which vary significantly and linearly with rotation are best candidates for such an approach. Currently, we have studied the computation of 2-D model based parameters, 1-D model based parameter, mean of the standard deviations on the x axis, y axis and diagonal axes of the texture image. The computations of the statistical features will be discussed in the first section of this chapter. In the second part, the statistical based rotation estimation algorithm will be explained.

The FFT based rotation estimation algorithm has some disadvantages, such as computational difficulties and rotation range. Honec et al. proposed in their study that even assuming all the exponential terms are pre-calculated and stored in a table, the total number of complex multiplications needed to evaluate the 2D DFT is N^4 (Honec et al., 2001) The number of complex additions is also N^4. Two techniques can be employed to reduce the operation count of the 2D-DFT transform. First, the row-column decomposition method partitions the 2D-DFT into many one-dimensional DFTs. Row-column decomposition reduces the number of complex multiplications from N to $2N^3$. The second technique for reducing the operation count of the 2D-FFT transform is the fast fourier transform (FFT). The FFT is a shortcut evaluation of the DFT. The FFT is used to evaluate the one-dimensional DFTs produced by the row-

column decomposition. The number of complex multiplications is reduced from $2N^3$ for direct evaluation of each DFT to $N^2 \log N$ for FFT evaluation using the row-column decomposition method.

Another drawback of the FFT is the restricted working region. The rotations which are outside of the region of $-45 \le \theta \le 45$ can not be estimated. Even this is not a problem for the weft-straightening problem; it should be found another method which is more time conservative and also work in large working region. Statistical based rotation estimation algorithms corresponds these requirements. The statistical based rotation estimation algorithm also has some disadvantages with respect to FFT based method. These disadvantages will be explained in this chapter as well as the statistical based rotation estimation algorithm method.

4.1 STATISTICAL FEATURES

In this section, the computation of the statistical features will be discussed. We will start with two parameters modeling approximation.

4.1.1 2-D Model Based Parameters

A pixel value can expressed with the following equation.

$$\hat{t}(x, y) = \alpha \cdot t(x, y - dy) + \beta \cdot t(x - dx, y) \tag{4.1.1}$$

where α and β are the model parameters. Apart from these parameters, there are also two variables which are dx and dy. This variables are the distances from the current pixel to previous pixel position. There are totally 4 parameters to be expressed. In the above equation, we try to minimize the following function.

$$f = \sum_{x=1}^{N} \sum_{y=1}^{N} [-\hat{t}(x, y) + \alpha \cdot t(x, y - dy) + \beta \cdot t(x - dx, y)]^2 \tag{4.1.2}$$

$$\frac{df}{d\alpha} = 0, \qquad \frac{df}{d\beta} = 0 \tag{4.1.3}$$

Solving for α and β, we optain,

$$\alpha = \frac{A_{22} \cdot B_1 - A_{12} \cdot B_2}{A_{11} \cdot A_{22} - A_{21} \cdot A_{12}}, \qquad \beta = \frac{A_{12} \cdot B_1 - A_{11} \cdot B_2}{A_{12} \cdot A_{21} - A_{11} \cdot A_{22}}, \tag{4.1.4}$$

where $A_{11} = \sum_{x=dx}^{M-1} \sum_{y=dy}^{N-1} t^2(x-dx, y), \quad A_{12} = \sum_{x=dx}^{M-1} \sum_{y=dy}^{N-1} t(x-dx, y) \cdot t(x, y-dy),$ (4.1.5)

$$A_{21} = \sum_{x=dx}^{M-1} \sum_{y=dy}^{N-1} t^2(x-dx, y), \quad A_{22} = \sum_{x=dx}^{M-1} \sum_{y=dy}^{N-1} t(x-dx, y) \cdot t(x, y-dy), \tag{4.1.6}$$

$$B_1 = \sum_{x=dx}^{M-1} \sum_{y=dy}^{N-1} t(x, y-dy) \cdot t(x, y), \quad B_2 = \sum_{x=dx}^{M-1} \sum_{y=dy}^{N-1} t(x, y) \cdot t(x-dx, y), \tag{4.1.7}$$

4.1.2 1-D Model Based Parameter

1-D model based parameter approximation is similar to 2-D model based parameters approximation. A pixel can be expressed as follows;

$$\hat{t}(x, y) = a \cdot t(x-dx, y-dy) \tag{4.1.8}$$

In this situation a has the following equality,

$$a = \frac{\sum_{x=dx}^{M-1} \sum_{y=dy}^{N-1} t(x-dx, y-dy) \cdot t(x, y)}{\sum_{x=dx}^{M-1} \sum_{y=dy}^{N-1} t(x-dx, y-dy) \cdot t(x-dx, y)} \tag{4.1.9}$$

4.1.3 Mean of Standard Deviations Parallel to the *X* Axis

The mean of the standard deviations along the *x* axis can be expressed as follows;

$$\Phi_x = \frac{\sum_{i=1}^{H} \sigma_{x_i}}{H}, \qquad I = \frac{\Phi_x}{M} \tag{4.1.10}$$

where Φ_x is the mean of standard deviations along the *x* axis of the texture image and I is statitical parameter which is independent of illumination disturbance. In order to eleminate the effects of the enviroment Φ_x is divided to M , which is the average pixel intensity of the texture. σ_{x_i} is the standard deviations of the each row. W and H stand for the image sizes as pixel value. W represents the width of the image, and H represents the height of the image.

4.1.4 Mean of Standard Deviations Parallel to the *Y* Axis

Similarly, the mean of the standard deviations along the *y* axis can be expressed as follows;

$$\Phi_y = \frac{\sum_{i=1}^{W} \sigma_{y_j}}{W}, \qquad J = \frac{\Phi_y}{M} \tag{4.1.11}$$

where Φy is the mean of standard deviations on the y axis of the texture image and J is statitical parameter which is independent of the light disturbance. σ_{y_j} is the standard deviations of the each columns.

4.1.5 Mean of Standard Deviations Along the Diagonal Axes

Let Φ_{d_1} and Φ_{d_2} be the means of standard deviations along the first and second diagonal axes, and M be the average pixel intensity of the texture. We define a new statistical parameter (*K*) as follows;

$$K = \frac{\Phi_{d_1} + \Phi_{d_2}}{2M} \tag{4.1.12}$$

4.2 STATISTICAL PARAMETERS BASED ROTATION ESTIMATION

The main idea of the theorem is to find the best statistical parameters which change significantly and linearly with rotation. To be able to find the best candidates we plot and look at the figures which give the desired result. To show the importance of the usage of more than one statistical feature, we divide the statistical based approach into two parts. In the first section, we analyze using 2 model parameters, which is the most important one. In the second section, we analyze using all parameters to show the improvement.

4.2.1 Statistical Rotation Estimation by Using Two Model Parameters

In this subsection, we will use the 2 model statistical parameters to estimate rotation angle. The algorithm of the system is as follows.

The most important part is to generate a look up table. The look-up table has the following structure

Table= [α β θ],

The look up table is generated by rotating the image in a desired region. We compute the 2 model parameters α and β for each value of θ. After the look-up table is generated, we analyze the system in the same region with more detail. For example if we generate the look-up table from -30 degree to +30 degree with 0.5 step rotation, we analyze the system in the same region but with 0.1 step of rotations. This method determines the performance of the system.

Another important point is the problem of choosing *dx* and *dy*. We have to choose distance variables *dx* and *dy* properly because it significantly affects the parameters-rotation linearity. To decide which distance values are best candidates for the target texture image, we implement two methods. The first one is to draw the parameter-

rotation graph for each *dx* and *dy* combination and look at the linearity with naked eyes. The second and more professional one is to draw θ-error graph and compute the least square error. The error stands for the difference between estimated rotation and real rotation. As an example, the best *dx* and *dy* values for the small image can be seen in the Figure 4.1 and Figure 4.2, respectively. As we mentioned, model parameters α and β linearly change with the rotation. The θ-error graph can be seen in the Figure 4.3.

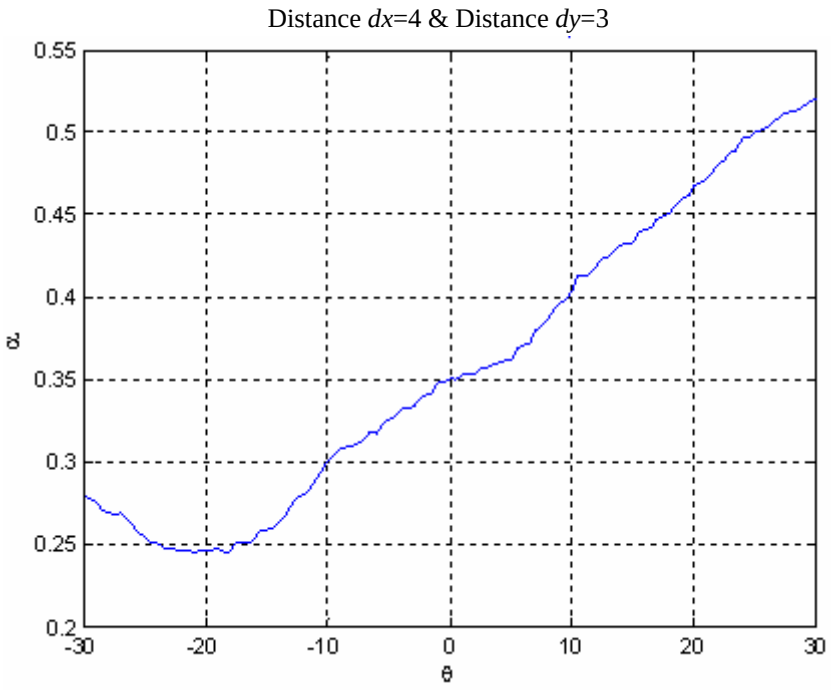


Figure 4.1 *α versus θ* plot.

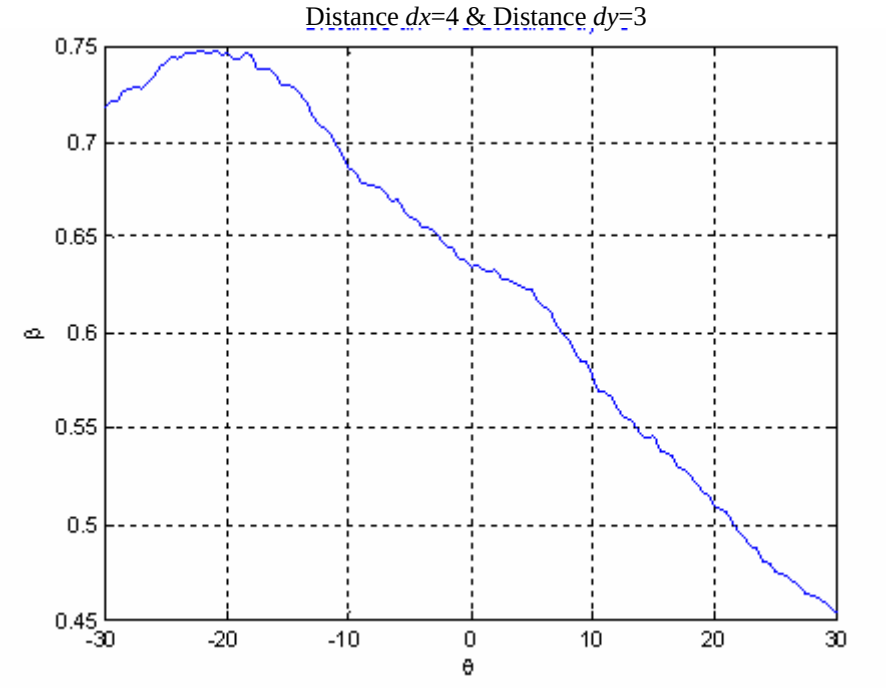


Figure 4.2 β *versus* θ plot.

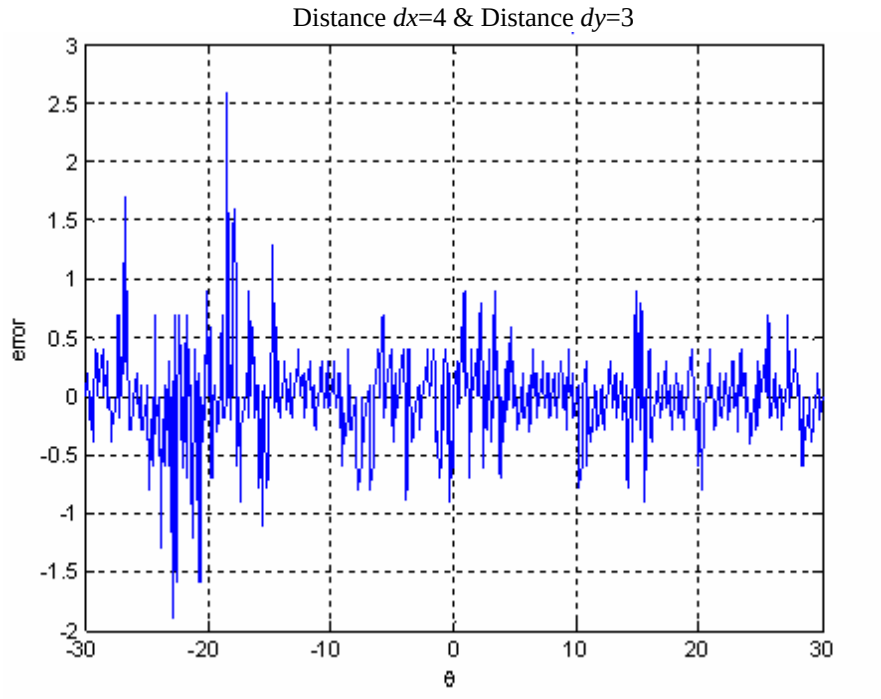


Figure 4.3 *Error* versus θ plot

The nearest neighborhood method is used to estimate rotation angle. Actually, the decision error is given by the following equation:

$$e = \sum_{i=1}^{N} w_i \left| \mu_i - \mu_{v,i} \right|$$

where $\mu_{v,i}$ is the array of statistical parameters, where the first index is for the rotation angle, and the second index is for the parameter number. The measured parameters are μ_i, and they are obtained by processing the captured the texture image. Finally, w_i's are weighting factors which are used to emphasize certain statistical parameters over others. Selection of these weighting factors is also an important issue, and affects the overall system performance. In our performance analysis tests, we have selected equal weights; however a further analysis can be done for optimum weight selection as well.

As seen from Figure 4.1 through Figure 4.3, in the region of -20 to -30, there is an undesired result on the *Error* versus θ plot in Figure 4.3. To reduce the error and get better results, we should increase the statistical features.

4.2.2 Statistical Features Based Rotation Estimation by Using Six Model Parameters

To get better results in rotation estimation algorithm, we used one model parameter, the mean of the standard deviations along the *x, y* and diagonal axes in addition to the two model parameters.

The idea is very similar to the two model rotation estimation algorithm; the first difference is that we generate the look table with all these statistical parameters which is explained in section 4.1. The other difference is the computational time. In this method, the processing time increases due to the size of the table of the parameters. Table has the following structure;

Table= [I J K a α β θ];

where I stands for the mean of standard deviations on the *x* axis, J stands for the mean of standard deviations on the *y* axis, K stands for the mean standard deviations on the diagonal axes, a stands for the one model parameter, and α and β stand for the two model parameters. Finally, θ stands for the rotation angle. We compute each

statistical feature and generate the look-up table for each rotation. The decision mechanism is the same as the two model parameter rotation estimation algorithm. The nearest neighborhood method is applied to find the rotation.

CHAPTER 5

EXPERIMANTAL SETUP

In this section, we explain how the algorithm is implemented and improved in a tiny embedded system. This process is very overwhelming and contains a lot of details. The process consists of two phases.

The first one is the development phase. we start with the development phase which is implemented and simulated in Matlab. The Matlab has many sophisticated functions and methods to improve the algorithm. However, we prefer to utilize the built-in subroutines.

The second phase is the implementation phase. The algotithm is implemented in a desktop PC and then in a single board PC (SBPC) which is also known as Industrial PC. This PC is like regular PC but it is designed for the industry. It is very small and convenient to use in many applications. It has serial ports, Ethernet card, TV card, keyboard mouse sockets. Namely, it has almost every properties of the standard PC. As an OS, we used embedded Linux operating system (Xlinux).

5.1 DEVELOPMENT PHASE OF THE ALGORITHMS

As explained in the previous chapters, there are two main algorithms that we study in this thesis. The first one is FFT based approximation and second one is statistical based approximation.

5.1.1 FFT Computation Based Rotation Estimation Algorithm

To test the camera in Matlab, following command was used:

vidobj = videoinput('winvideo', 1);

This command creates the video object. All camera properties can be obtained from this object. To view and open the window for displaying the vision we used the following command (see Figure 5.1).

preview(vidobj)

After we get an acceptable vision, we can proceed with capturing process of an image from the camera.

To get a snapshot from the camera, we used the following command.

I=getsnapshot (vidobj);

Figure 5.1 Video previews for testing.

Next, we snap an image in the size of 320x240. This size depends on the camera. After we get the image matrix into “I” variable, we continue with the modify image subroutine given as :

Im=modifyimage(I,n);

The modify image subprogram cut the image (I) into *nxn* pixel from center to side. This makes the image matrix square and gray scale. The code can be found in Appendix A. After we get the square matrix we apply the *fft2new* algorithm.

Ixd=fft2new(Im,threshold,gamma);

After executing this code, we obtain the FFT matrix of the image and store it in *Ixd* matrix.

The brightest point in predefined region is found as;

[xI yI]=findbrghtpnt(Ixd) ;

where *xI* and *yI* contain the brightest point locations.

Finally, the rotation angle is estimated with the following well known formula:

*theta=180/pi*atan(xI/yI);*

5.1.2 Statistical Parameters Based Rotation Estimation Algorithm

We divide statistical method into two parts. These are two model parameter based rotation estimation and six model parameters based rotation estimation. They are exactly same algorithm but six model parameters use more statistical features to get better performance. The algorithm is given below :

- Generate a look-up table by rotating the image in a desired region and step interval.
- Connect camera to the system. Check if the system recognizes the camera.
- Get a snapshot from the camera.
- Modify the image size and convert to gray.

- Obtain the model parameters.
- Look up the table and find the best rotation angle by using nearest neighborhood method.

5.1.2.1 Generating Look-Up Table and Parameter Extraction

We use the following function to generate look-up table:

table=gettable(I, range, dx, dy)

This function takes the image, range and distance values as input variables and generates the table as an output. The *gettable* function for two model statistical approach is given as follows:

```
function table_real=gettable(I, range, dx, dy)
        n=round (size (I, 1)/4);
        xr=[];
        theta=[];
                for t=range
                        XR = imrotate (I, t, 'bilinear', 'crop');
                        XR=modifyimage (XR,n);XR=double(XR);
                        xr=[xr getparameters(XR, dx, dy)];
                        theta=[theta t];
                end

table_real=[xr(1,:);xr(2,:);theta];
```

where *n* is used to obtain square image matrix. Image is rotated in a pre-assigned range and for all rotation angle statistical parameters is calculated and stored in an *xr* array . Finally the table matrix is generated. The crucial function is the *getparameters* function. It computes the statistical features of the image. The *getparameters* function for two model rotation estimation is given as follows;

```
function x=getparameters(X, dx, dy)

[M, N]=size(X);

myA11=sum(sum(X([1:M-dx],[dy+1:N]).*X([1:M-dx],[dy+1:N])));
myA12=sum(sum(X([1:M-dx],[dy+1:N]).*X([dx+1:M],[1:N-dy])));
myA21=sum(sum(X([dx+1:M],[1:N-dy]).*X([1:M-dx],[dy+1:N])));
myA22=sum (sum(X([dx+1:M],[1:N-dy]).*X([dx+1:M],[1:N-dy])));
myB11=sum(sum(X([dx+1:M],[dy+1:N]).*X([1:M-dx],[dy+1:N])));
```

```
myB21=sum(sum(X([dx+1:M],[dy+1:N]).*X([dx+1:M],[1:N-dy])));

A= [myA11 myA12; myA21 myA22] ;
B= [myB11; myB21];
x=inv (A)*B;
```

The mathematical analysis can be found in Chapter 3. For the six model parameters rotation estimation algorithm, we change the *gettable* function as follows;

```
function table_real=gettable(I, range, dx, dy)
    n=round (size (I, 1)/4);
    table=[];
    theta=[];
        for t=range
            XR = imrotate (I, t, 'bilinear', 'crop');
            XR=modifyimage(XR,n);XR=double(XR);
            xdir1=xdir (XR);
            ydir1=ydir (XR);
            diagdir1=diagdir (XR);
             mod1=getmodpar (XR, dx, dy);
            mod2=getparameters (XR, dx, dy);
            xr=[xdir1 ydir1 diagdir1 mod1 mod2];
            table=[table; xr] ;
        end
    table_real=[table range']';
```

As seen from the program sequence, in addition to 2 model parameters we add other statistical parameters to the look-up table as explained in Chapter 3. After we generate the table, we analyze the system and its performance.

5.1.2.2 Testing the System and Camera Setting

The system was tested using two different ways. The first one is that we decrease step interval and compute new statistical features and plot the theta-error graph. The second one is that we connect a webcam to computer by using Matlab and try to understand how the estimated rotation changes. The experimentation of the second method is not so efficient because while we are rotating the camera (or texture), we exactly do not know how much we really rotate the camera or texture. Another drawback of the second method is that we can not rotate the camera with our hand perfectly. While we are rotating the camera, we also change the direction of the camera unwillingly. The first method also has some drawbacks such as rotation is done by

computer and there is no light effects and noise on the texture image. However, it gives the general idea about how statistical parameters change with the rotation. Therefore, the experimental results are obtained mostly using the first method.

Camera Settings and snapshot are exactly the same as the FFT based approximation. The modification of the image size and conversion to the gray level image is also the same as FFT based approximation.

To analyze the system, we used the GUI (Global User Interface) to make the usage of the program easier. Figure 5.2 shows the main menu. There are six sections which are;

- Get image
- Inspect for the best statistical distance (*dx* and *dy*)
- Rotate image
- Calculate Rotation
- Theta-Error Plotter
- Theta-Error Plotter for different distance

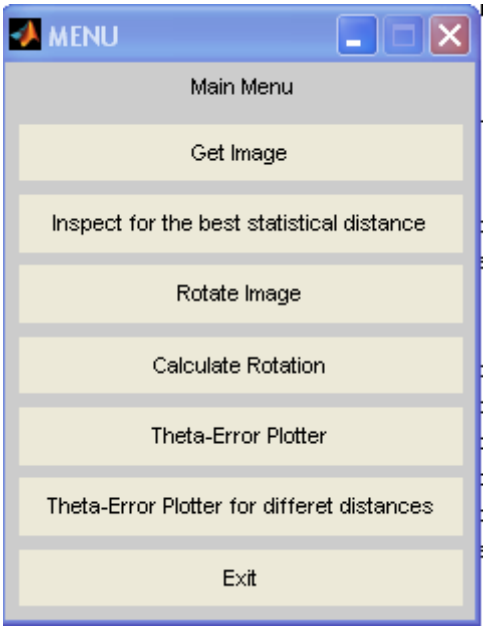


Figure 5.2 Testing menu.

***Get Image* function:**

Get file image function basically gets the image path and name for the *imread* command. The image path is taken from the application folder or captured from the camera. To get the image path from the folder;

```
[file_name file_path] = uigetfile ('*.bmp');
I = imread ([file_path,file_name]);
```

To capture the image from a camera;

```
vidobj = videoinput('winvideo', 1);
preview(vidobj)
I=getsnapshot(vidobj);
```

***Inspect for the best statistical distances* function:**

The aim of the *inspect for the best statistical distance* function is to plot the 2 model and one model parameters and rotation graphs for the combinations of *dx* and *dy*. Let's assume that the region is *dx=1:10* and *dy=10* then we will get 100 plot as a result and we look at the linearity to choose best statistical distance *dx* and *dy* values.

```
drangex=input('Distance Range for dx: ');
drangey=input('Distance Range for dy: ');

for d1=drangex
        for d2=drangey
                table=gettable(I,range,d1,d2);
                mod1=table(4,:);
                mod21=table(5,:);
                mod22=table(6,:);
                figure('Name','1 Model Parameter','NumberTitle','off')
                plot(range,mod1,'b')
                xlabel ('theta')
                ylabel ('a0')
                title(['Distance dx= ',int2str(d1),' & Distance…
                y=',int2str(d2)],'Color','b')

                figure('Name','2 Model Parameters','NumberTitle','off')
                plot(range,mod21,'b')
                xlabel ('theta')
                ylabel ('a1')
                title('Mean of Diag Left-Right-Mean')
```

```
                title(['Distance dx= ',int2str(d1),' & Distance…
                y=',int2str(d2)],'Color','b')

                figure('Name','2 Model… Parameters','NumberTitle','off')
                plot(range,mod22,'b')
                xlabel ('theta')
                ylabel ('a2')
                title('Mean of Diag Left-Right-Mean')
                title(['Distance dx= ',int2str(d1),' & Distance…
                dy=',int2str(d2)],'Color','b')
                disp('Press any key for the next distance...')
                pause;
                close all;
                clc;
        end
end
```

Rotate Image **function:**

Using this function, the image can be rotated for a desired angle and *getparameters* function is executed to get statistical parameters. The distance variables *(dx, dy)* have to be entered.

```
t=input('Enter the rotation angle: ');
xr=rotandgetpar(I,t,dx,dy)
```

Calculate Rotation **function:**

In order to estimate the rotation, we enter the table and *xr* as an input. The nearest neighborhood method is applied while finding the best match.

```
theta=gettheta(table,xr)

function theta=gettheta(table,xr)
        [m,n]=size(table);
        minerror=10;
                for i=1:n
                 %xR=table(1:8,i);
                xR=table(1:2,i);
                error =norm(xR-xr);
  if error < minerror
  minerror=error;
  %theta=table(9,i);
```

```
    theta=table(3,i);
    end
  end
```

***Theta Error Plotter* function:**

Theta-Error function is used to plot the theta-error graph by using the following code.

```
rot=input('Enter the table range(Example:0:0.5:180):');
err=[] ;
for t=rot
    xr=rotandgetpar(I,t,dx,dy);
    thetac=gettheta(table,xr);
    err=[err (t-thetac)];
end
disp(mean(err));
figure
plot(rot,err)
axis auto
xlabel('theta')
ylabel('error')
title(['Distance dx= ',int2str(dx),' & Distance… dy=',int2str(dy)], 'Color', 'b')
```

***Theta-Error Plotter for Different Distances* function:**

We use the theta-error plotter with the combinations of various *dx* and *dy* distance values. Then we apply least square error method to find the best distance *dx* and *dy* values.

```
rot=input('Enter the rotation range(Example:0:1:180):');
drangex=input('Distance Range dx: ');
drangey=input('Distance Range dy: ');
disp('This process may take a few minutes...')
minerr=10000;
c=clock;
disp(['Started at :' int2str(c(1,4)) ':' int2str(c(1,5)) ':'…
int2str(c(1,6))])
    for dx=drangex
        for dy=drangey
            table=gettable(I,range,dx,dy);
            err=[] ;
                for t=rot
                    xr=rotandgetpar(I,t,dx,dy);
                    thetac=gettheta(table,xr);
```

```
                        err=[err (t-thetac)];
                    end
                pwrerr=sum(err.^2);
                    if pwrerr<minerr
                        minerr=pwrerr;
                        bestdx=dx;
                        bestdy=dy;
                        besterr=err;
                    end
                disp(dx*dy)
                c=clock;
                disp([int2str(c(1,4)) ':' int2str(c(1,5)) ':' …
                int2str(c(1,6))])
            end
        end
        disp(['Finished at : ' int2str(c(1,4)) ':'…
        int2str(c(1,5)) ':' int2str(c(1,6))])
        figure
        plot(rot,besterr)
        xlabel ('rot')
        ylabel ('err')
        title(['Best Distances  dx=… ',int2str(bestdx),' &…
        Distance dy= ',int2str(bestdy)],'Color','b')
    axis auto
```

After we find the best distance values for the texture image, we can implement the algorithm in the embedded system. The above method is same for two-model and five-model statistical features however the result is very different.

5.2 IMPLEMENTATION OF THE ALGORITHMS ON THE LINUX SYSTEM

Up to now, we have investigated the FFT and statistical based rotation estimation algorithms. We are interested in the software side of the project. However, the significant part of the problem is the implementation part. This section covers the installation of Linux and how to switch from a conventional Linux to embedded system.

5.2.1 Installation of Linux and Embedded Linux Operating System

We firstly setup a Debian Linux OS on a desktop PC. The procedure we go over is as follows,

- Install Debian Kernel (2.4-27-2-386)

- Install the base system from CD-ROM or DVD-ROM
- Complete installation via CD-ROM or Internet
- HDD Partitioning as flows

Device Boot	Start	End	Blocks	Id	System
/dev/hda1 *	1	1155	9277506	83	Linux
/dev/hda2	1156	2498	10787647+	83	Linux

- Installation of TELNET and FTP server
- Kernel Recompilation
- Compile the Xlinux Kernel (Embedded Linux OS) with BTTV driver
- Boot the new system and test BTTV driver (install xawtv program)
- Configure the Video For Linux (V4Linux) parameters
- SDL Library and libfg installation
- Compile the project.c source file
- Run and test the system in Desktop PC

After we setup the Debian Linux system, we developed the project application code. In this process, we wrote the C code of the program that was used in Matlab. It is basically a conversion of the codes from Matlab to C. Since there is no direct conversion from Matlab code to C code in our case, we need to write the code again by using our own function and libraries. We use SDL libraries for the video applications. SDL library provides us easy to use functions while dealing with the video application.

After we write our code and make sure that the algorithm works fine with the Debian Linux system. Then we continue with the installation of the executable file in to Vortex86 SBC (Single Board Computer.) The hierarchy is shown in Figure 5.3.

The procedure that we applied in Debian Linux Desktop is the same with the Vortex86 SBC. One difference is that here is no need to install SDL libraries in to SBC and compile the program. The executable file is transferred via ftp to embedded system. Once this is done, a problem is encountered. The problem is that since some of the “.so” object files are missing in the SBC, we need to transfer these files from Debian Desktop PC to SBC, too. The other difference is that we install Xlinux system on SBC. These

system works very fast with respect to debian system while booting the system. It takes about 10-12 seconds to boot system as soon as the energy is given.

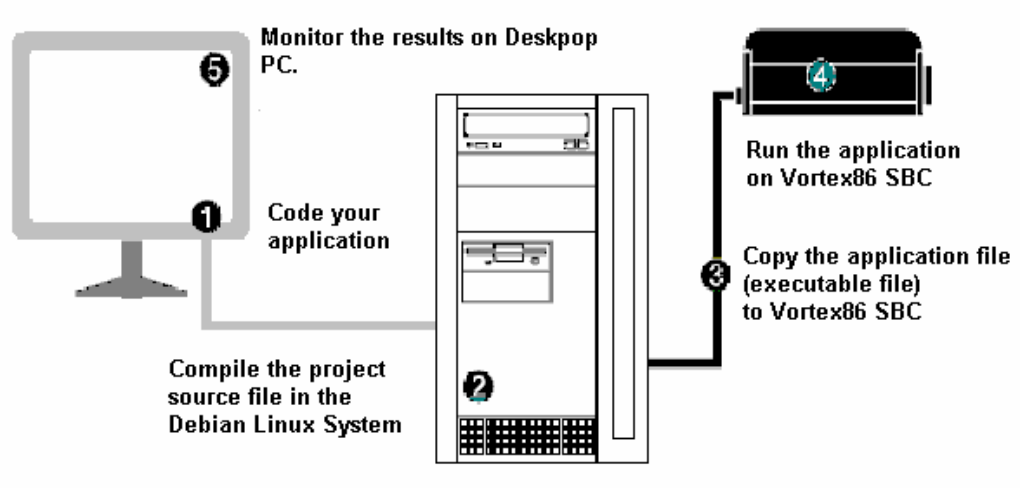


Figure 5.3 Application developments and implementation hierarchy.

5.2.2 Smart Camera System Architecture and Networking

Another important part of the experimental setup is the networking. Since the cameras are connected to each other with a network, this is also another topic to be covered. The servers will be “Smart Cameras”, which has Embedded Linux OS and client can be either Linux or Windows loaded systems. Each server will compute the rotation and will wait for the client to make request. As soon as the client make request, server starts to send vision results to the client. There are totally 4 or 6 Smart Cameras (SmartCam) depending upon the requirements as shown in Figure 5.4. Each SmartCam has their vision area and connected to a network with a hub. There is only one client and get rotation information from each server (or SmartCam). The final decision is made in this client PC. The motors are activated based on the final decision. Since our target is to estimate rotation angle, we will not discuss the driving motor part. Connection oriented TCP/IP flowchart can be seen in Figure 5.5 (Stevens, 1990).

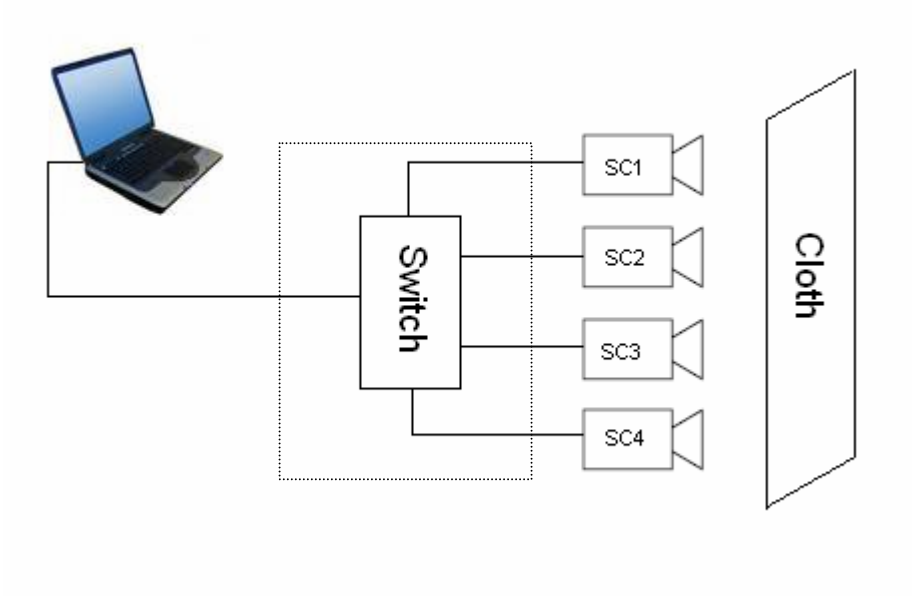


Figure 5.4 Weft-Straightening system architecture.

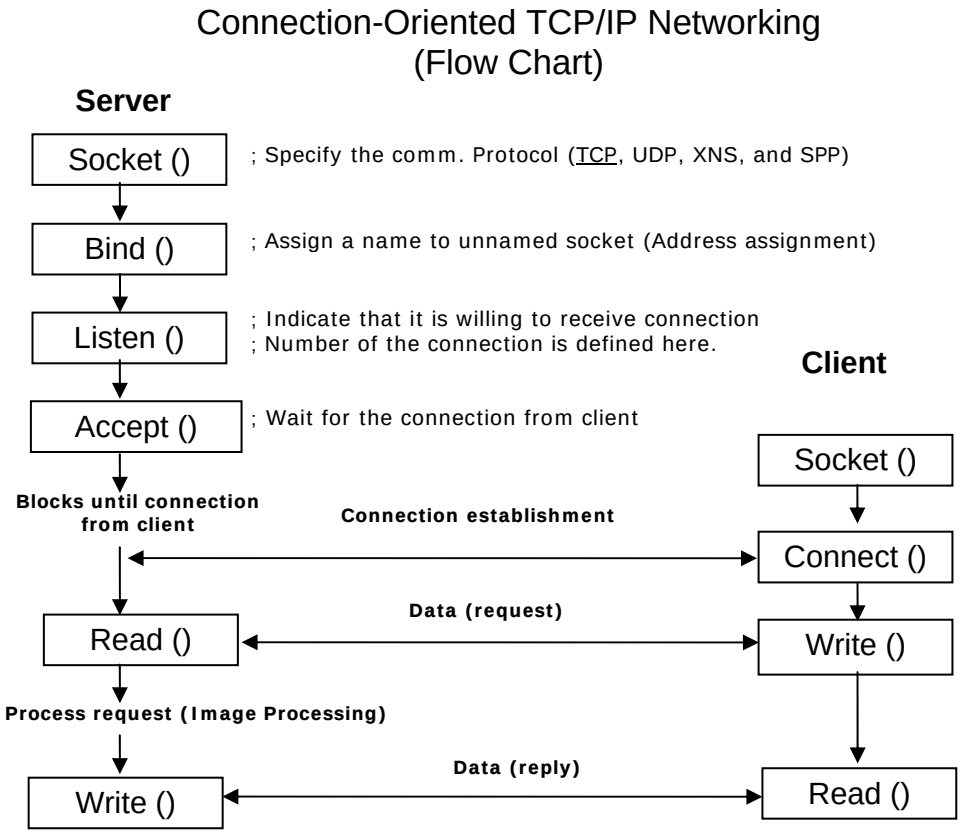


Figure 5.5 TCP/IP networking

5.2.3 Elementary Socket System Calls

In this section the elementary system calls which are used in TCP/IP networking will be explained in detail.

5.2.3.1 "Socket" System Call

The network I/O, is established by calling the socket system call and specifying the type of communication protocol desired (Internet TCP, Internet UDP, XNS SPP, etc.).

 # include<sys/types.h>
 # include<sys/socket.h>

int socket (int family, int type, int protocol);

The family is one of;

AF_UNIX	*UNIX internal protocol*
AF_INSET	*Internet Protocol*
AF_NS	*Xerox NS protocol*
AF_IMPLINK	*IMP link layer*

The AF_ prefix stands for "address family." There is another set of terms that is defined, starting with a PF_ prefix, which stands for "protocol family." Either term for a given family can be used, as they are equivalent.

The socket system call returns an integer value, similar to a file descriptor. It is called a socket descriptor, or a *sockfd*. To obtain this socket descriptor, we specify the address family and socket type (stream, datagram, etc.). For an association;

[Protocol, local-addr, local-process, foreign-addr, foreign-process]

All the socket system call specifies is one element of this 5-tuble, the protocol. Before the socket descriptor is of any real use, the remaining four elements of the

association must be specified. The socket system calls and association elements can be seen in Table 5.1 (Stevens, 1990).

Table 5.1 Socket system calls and association elements

	Protocol	Local addr, local-process	Foreign addr, foreign process
Server	Socket ()	Bind()	Listen(), Accept()
Client	Socket()	Connect()	

5.2.3.2 "Bind" System Call

The bind system call assigns a name to an unnamed socket.

 # include<sys/types.h>

 # include<sys/socket.h>

*int bind (int sockfd, int sockaddr *myaddr, , int addrlen);*

The second argument is a pointer to a protocol-specific address and the third one is the size of this address structure. There are three use of bind.

Servers register their well known address with system. It tells the system "this is my address and any messages received for this address are to be given to me." Both connection-oriented and connectionless servers need to do this before accepting client requests.

A client can register a specific address for itself. A connectionless client needs to assure that the system assigns it some unique address, so that the other end (the server) has a valid return address to send its responses to.

5.2.3.3 "Connect" System Call

A client process connects a socket descriptor following the socket system call to establish a connection with a server.

 # include<sys/types.h>

 # include<sys/socket.h>

*int connect (int sockfd, int sockaddr *myaddr, , int addrlen);*

The *sockfd* is socket descriptor that was returned by the socket system call. The second arguments are a pointer to a socket address, and its size, as described earlier.

For the connection oriented protocols, the connect system call result in the actual establishment of a connection between the local system and the foreign system.

Messages are typically exchanged between the two systems and specific parameters relating to the conversation might be agreed on. In these cases the connect system call does not return until the connection is established, or an error is returned to the process.

5.2.3.4 "Listen" System Call

This call is used by a connection oriented server to indicate that is willing to receive connection.

int listen (int sockfd, int backlog);

It is usually executed after both socket and bind system calls, and immediately before the accept system call. The *backlog* argument specifies how many connection requests can be queued by the system while it waits for the server to execute the accept system call. This argument usually specified as 5, the maximum value currently.

5.2.3.5 "Accept" System Call

After a connection oriented server executes the listen system call described above, an actual connection from some client process is waited for by having the server execute the *accept* system call.

*int accept (int sockfd, struct sockaddr *peer, int *addrlen);*

Accept system call takes the first connection request on the queue and creates another socket with the same properties as socket. If there are no connection requests pending, this call blocks the caller until one arrives.

CHAPTER 6

TEST RESULTS

FFT and Statistical based rotation estimation algorithm test results are presented in this chapter.

6.1 FFT COMPUTATION BASED ROTATION ESTIMATION ALGORITHM

The Matlab simulation results will be explained in this section. Although we test the system in real time, we use the Matlab simulation results since Matlab has powerful graphical interface and graphic functions. The real time system does not need any graphical plots, hence Matlab simulation results are used. From our experiences, the Matlab result and real time system result are very close to each other.

6.1.1 Rotation of an Image in Matlab

The Image has to be zoomed enough to get a clear vision response on the FFT domain. There are some rotated image samples which can be seen in Figure 6.1.

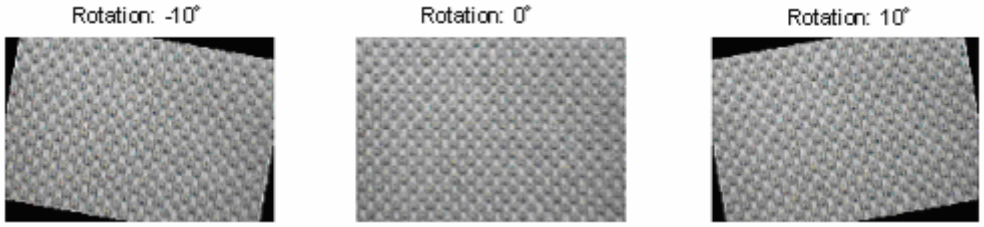


Figure 6.1 Rotated image figures.

6.1.2 The FFT of the Texture Image

The rotated FFTs of the images can be seen in Figure 6.2. The result shows that the rotation on the spatial domain correspond the rotation on the frequency domain. This phenomenon is proved in Chapter 3.

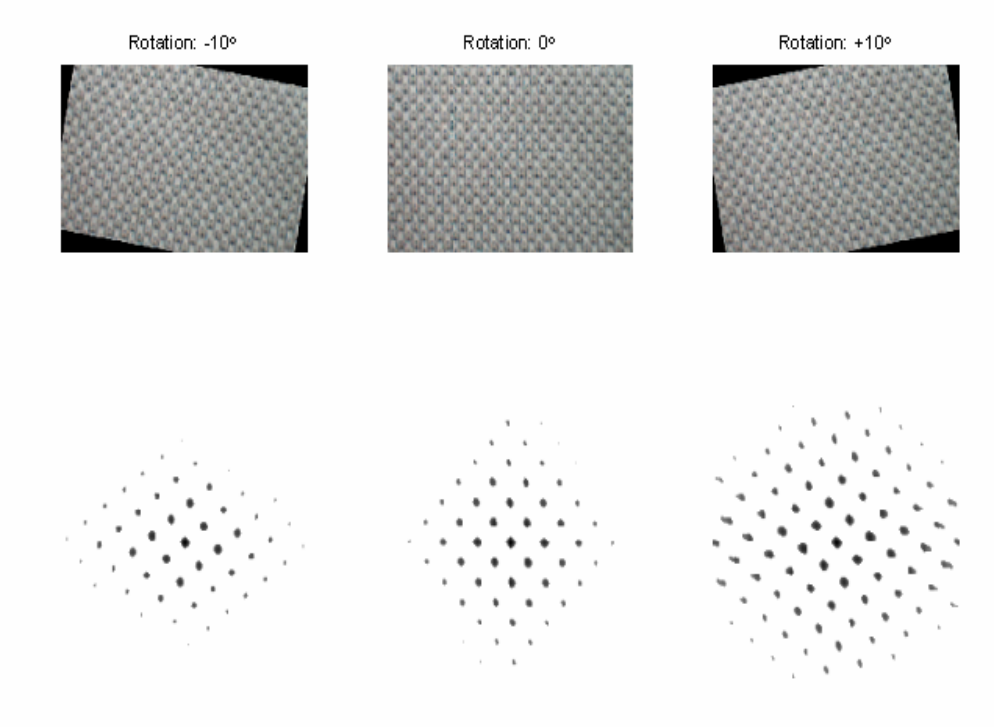


Figure 6.2 FFTs of the rotated images**.**

6.1.3 Finding the Brightest Point

After we apply the *find brightest point* algorithm, we get the results which are displayed in Figure 6.3. *Finding brightest point* algorithm was given in Chapter 5 and the Matlab code is shown in Appendix A.

Finally we compute the rotation from the brightest point position. For this example, we estimated -10.4915 for -10 rotation angle, 0 for 0 rotation angle and 10.4915 for +10 rotation angle, respectively. It seems that there is almost 0.5 degree error on the estimation. This is very acceptable for weft-straightening machine.

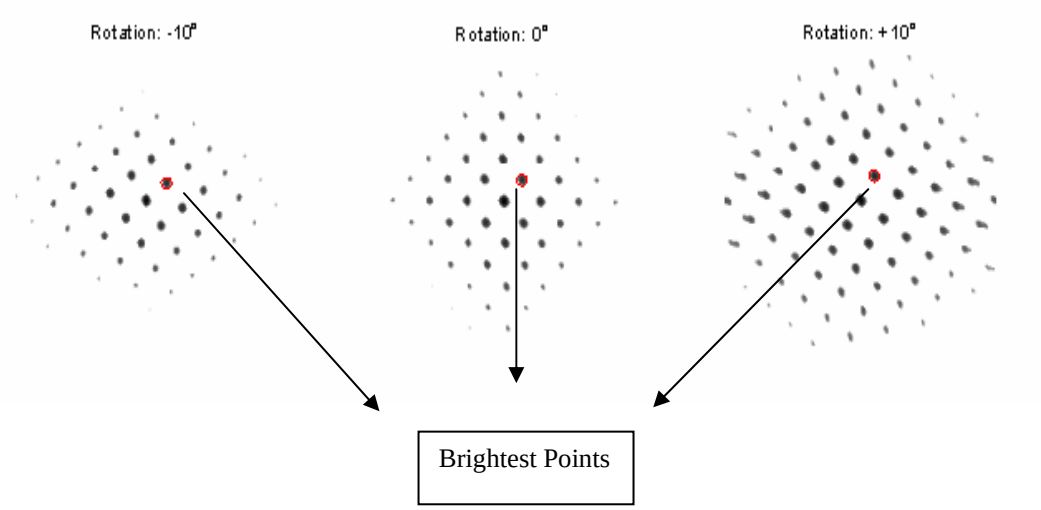


Figure 6.3 Finding the brightest point.

6.1.4 Theta-Error Plots

In order to analyze the algorithm, the image is tested in the rotation region of theta= -35:0.1:35 and each error which corresponds to the related theta is plotted. The rotation is done by Matlab imrotate function and “bilinear-crop” method is applied. The result is shown in Figure 6.4. There are two main reasons why we chose this region.

- The region is acceptable for texture weft-straightening machine and is never exceeded after the system is setup by a technician.
- The FFT based rotation estimation algorithm can not estimate the rotation which is out of this region due to the periodicity of the texture image FFT pattern.

By plotting the theta-error graph, we can see that which angle generates much error. As seen from the Figure 6.4. The maximum error is about 3 degrees when the actual rotation is 13.5 degrees and 27.3 degrees.

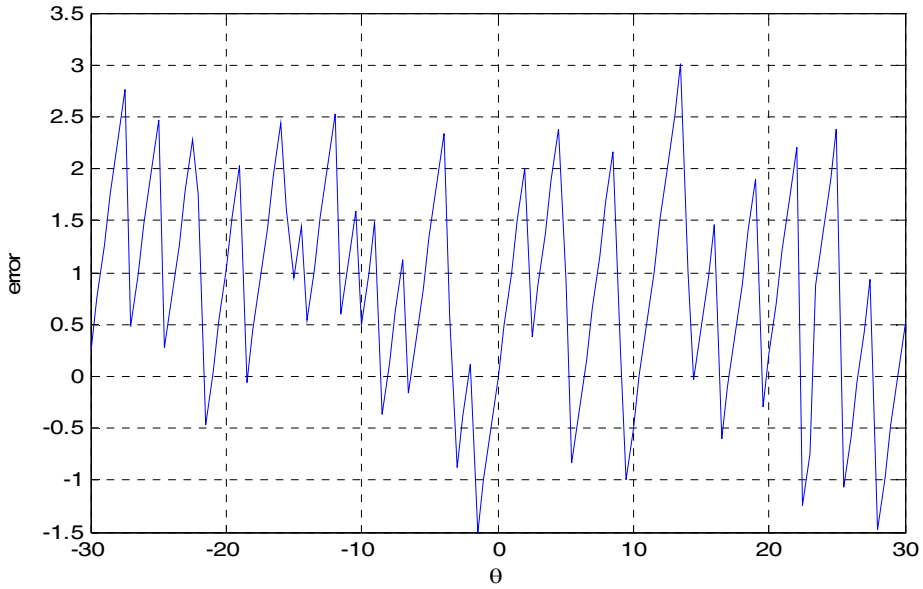


Figure 6.4 *Error versus* θ plot.

6.1.5 The Drawback of the FFT Based Approach

We expected that the system works fine in the region of theta= -45:45 since we are searching the first area on the x-y axis. However, the algorithm gives reasonable response in the region of theta= -40:40. The system loses its functionality outside of this region. This problem can be seen from Figure 6.5.

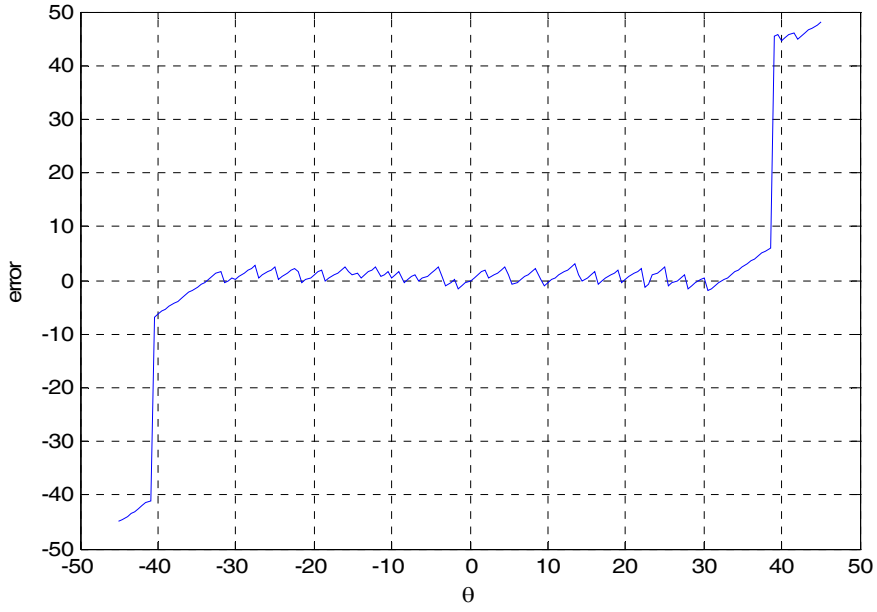


Figure 6.5 The drawback of the FFT algorithm.

6.2 STATISTICAL FEATURES BASED ROTATION ESTIMATION

Two parameters and six parameters modeling methods are discussed in this section. There are three sizes of texture images that we used for experimentation. The sizes of the images are 140x100, 180x150 and 440x325 (see Figure 6.6).

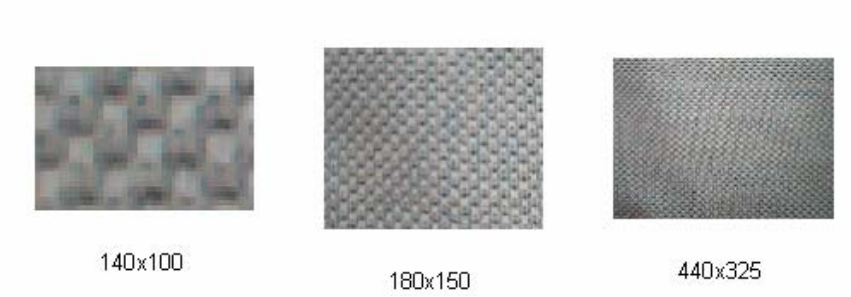


Figure 6.6 Different size images used in the experiment.

6.2.1 2-D Model Based Parameters

For *dx=1* and *dy=1* the 2 model parameters (α and β) changes with the rotation can be seen from the following figures. These *dx* and *dy* are arbitrarily chosen values for just illustration. Later on we find the optimum *dx* and *dy* values and show the effect of the distance variables. The following figures show how the model parameters vary with rotation.

As we see from these figures, the model parameters almost change linearly with the rotation. However, the *dx* and *dy* values has to be chosen properly. Although *dx=dy=1* gives reasonable response, we have to search for the optimum *dx* and *dy* values. As seen from the Figure 6.7, there is a bounce around zero degree rotation. Therefore, the estimation error will be high in this region. We could not analyze the reason of this situation. The situation does not occur for all the *dx* and *dy* values. This can be seen from the further illustrations.

Figure: 6.7 α versus θ plot for 440x325 size image.

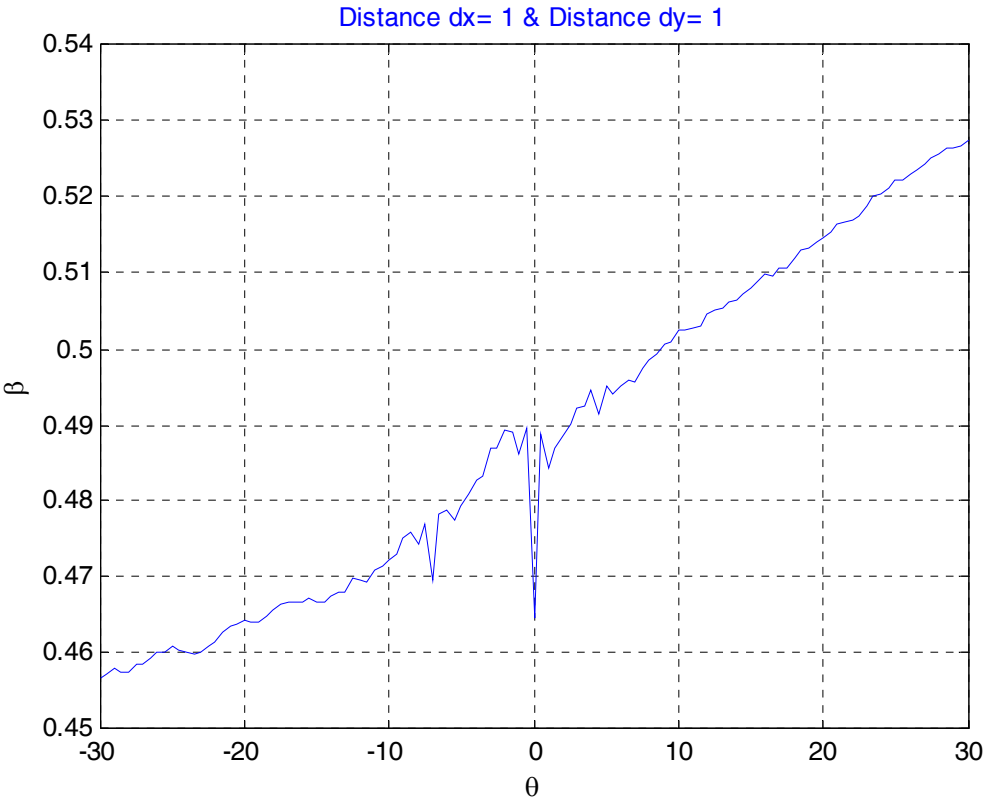


Figure: 6.8 β versus θ plot for 440x325 size image.

6.2.1.1 Usage of Theta-Error Plotter

We use theta-error plotter program to see the difference between estimated error and actual error for *dx=dy=1*. It is shown from the result that the error is very large around -20 and 0 degree. Obviously, this is not a desired result; therefore, we need to find the optimum *dx* and *dy* values to improve the performance.

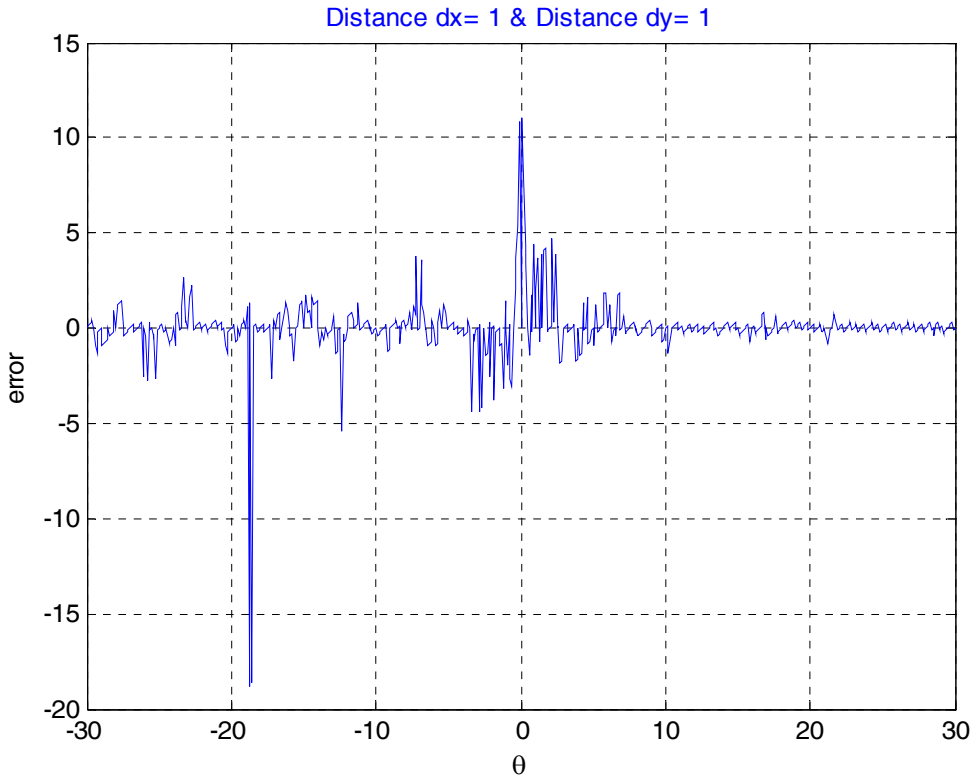


Figure 6.9 *Error versus θ* plot for 440x325 size image.

6.2.1.2 The Optimum Distance Values (dx, dy)

We execute the program that we explained in previous chapter, and we found the optimum distance values for the small size, middle size and large size images as follows. For a small size (45x30) image, the optimum distances are *dx=4* and *dy=3*. Please also note that the error reaches its maximum value in non linear region.

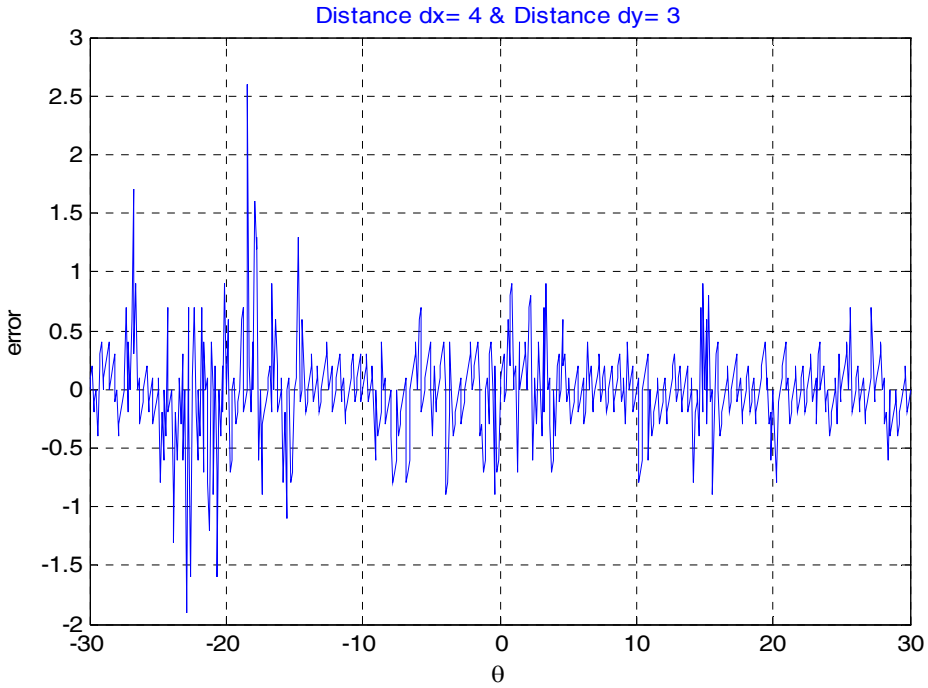


Figure 6.10 *Error versus* θ plot for the optimum distance values of small size image.

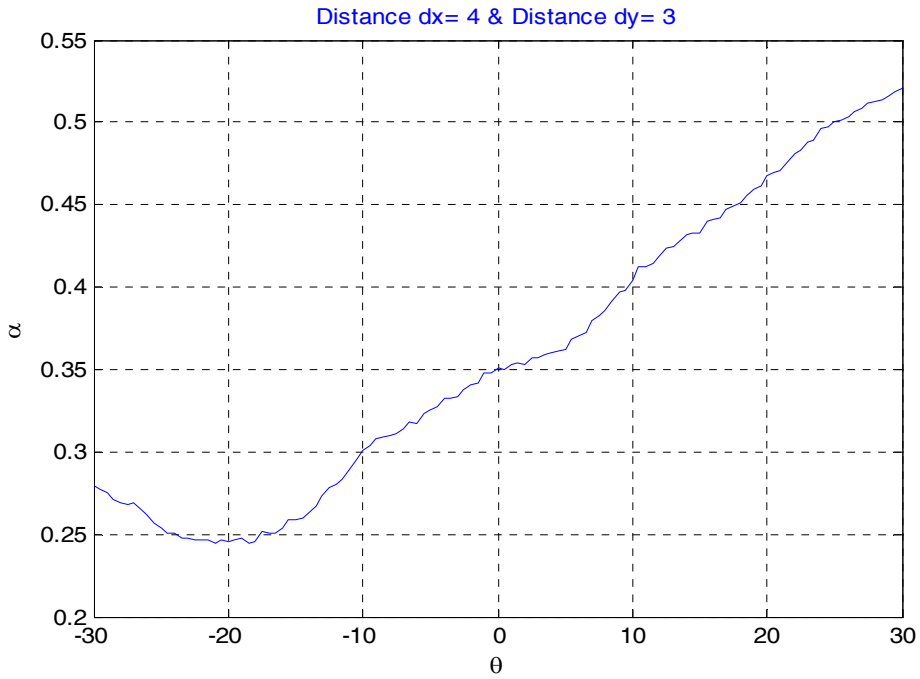


Figure 6.11 α versus θ plot for the optimum distance values of small size image.

For the middle size (180x150) image the optimum distances are found as dx=9 and dy=8.

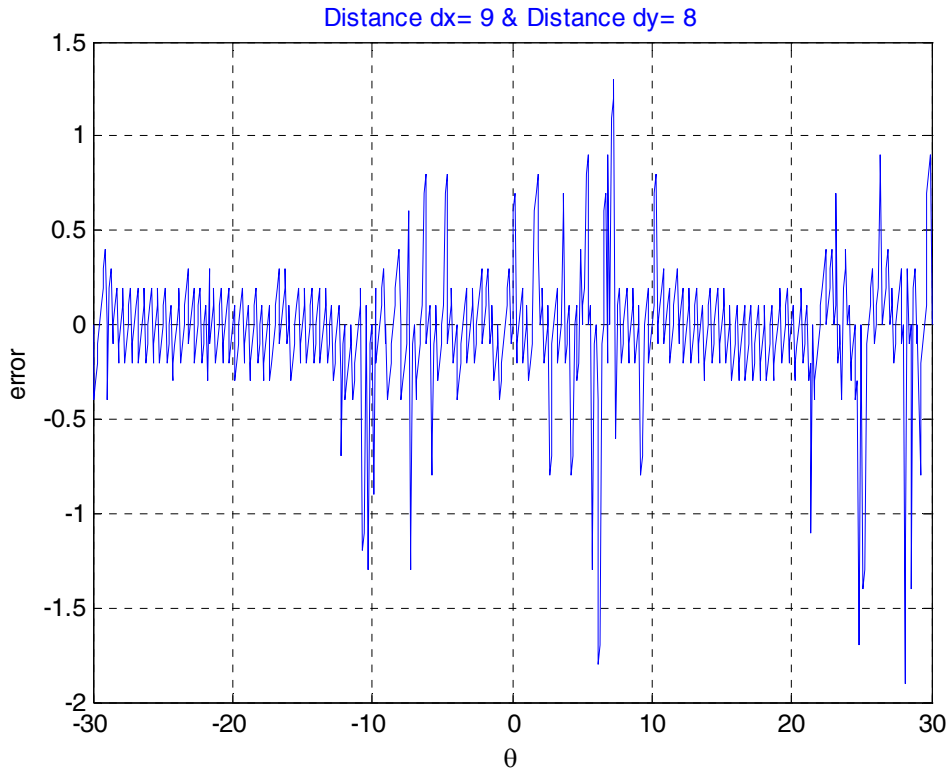


Figure 6.12 *Error versus* θ plot for the optimum distance values of middle size image.

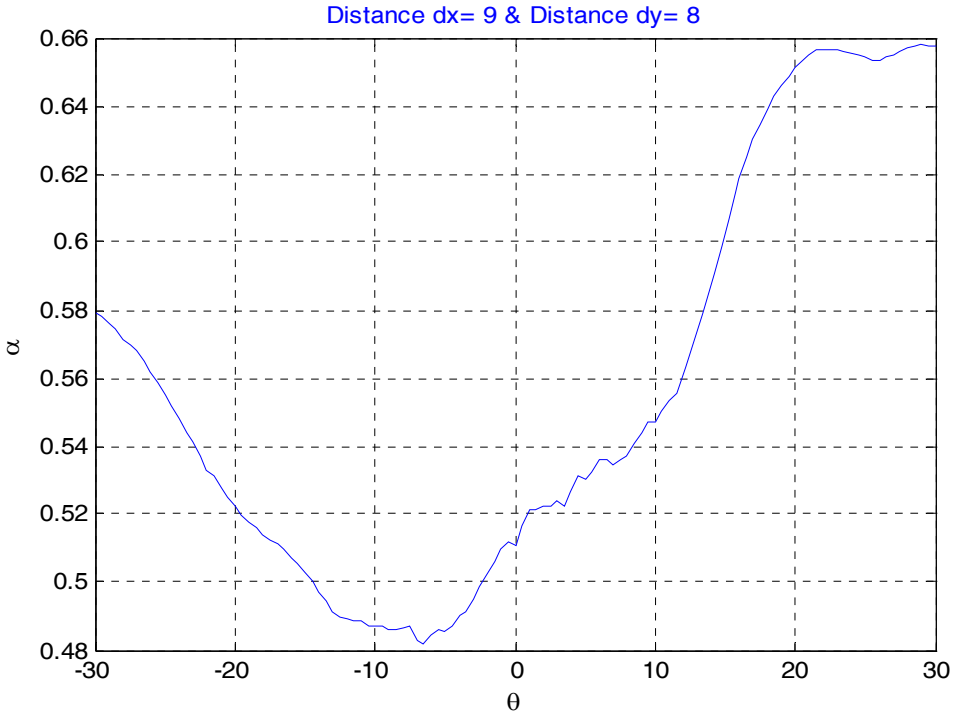


Figure 6.13 α versus θ plot for the optimum distance values of middle size image.

For the large size (440x325) image the optimum distances are found as *dx=1* and *dy=7*.

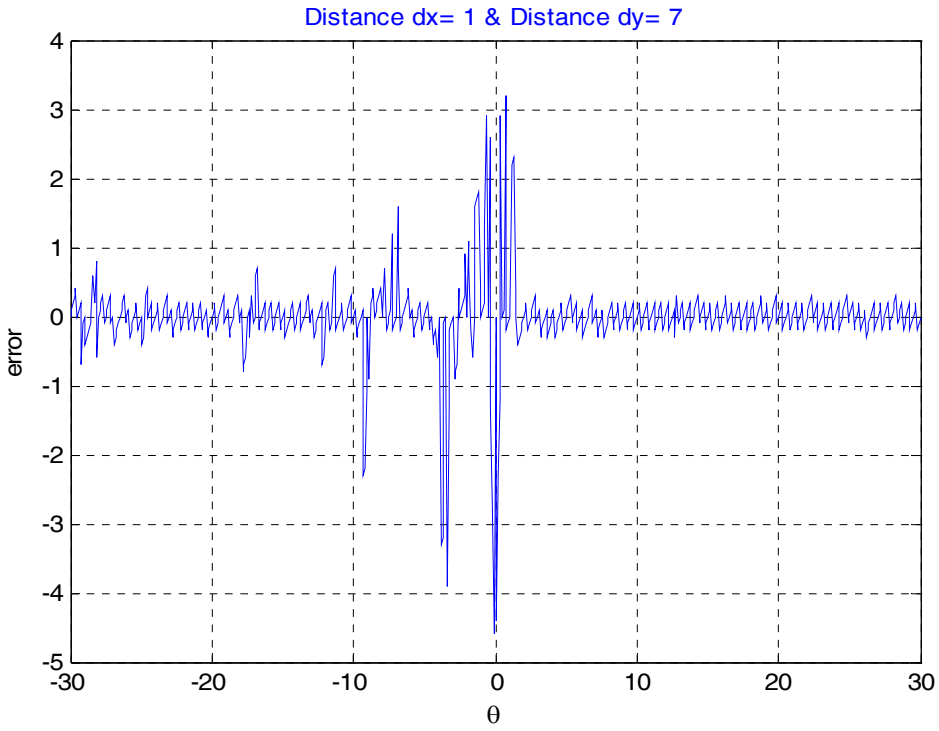


Figure 6.14 *Error versus* θ plot for the optimum distance values of large size image.

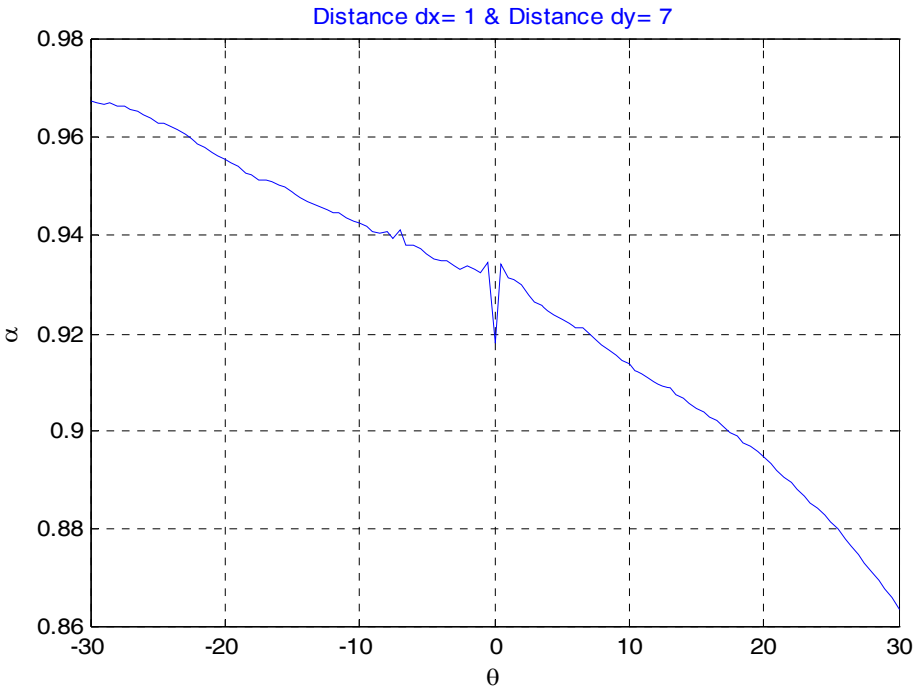


Figure 6.15 α versus θ plot for the optimum distance values of large size image.

From these results, it is shown that, even for the optimum distances the error look high in the range of non-linear region. Another important result is that, for some distance values, the error is very high when the rotation is about zero. Although we could not find the reason of this fact, we consider that this non-linearity occurs due to the numerical computation. Next, we increase the number of statistical parameters to improve the performance of the system.

6.2.2 Six Parameters Model Based Rotation Estimation Algorithm

In addition to two model parameters, we use the following statistical features; mean of the standard deviations along the *x* axis, *y* axis, diagonal axes and one parameter modeling.

6.2.2.1 Mean of Standard Deviation Parallel to the X Axis

The mean of the standart deviations along the *x* axis for small size image is shown in Figure 6.16. As it is seen, there is no linarity between rotation and the mean along *x* axis. Therefore, this result should be considered when estimating the rotation.

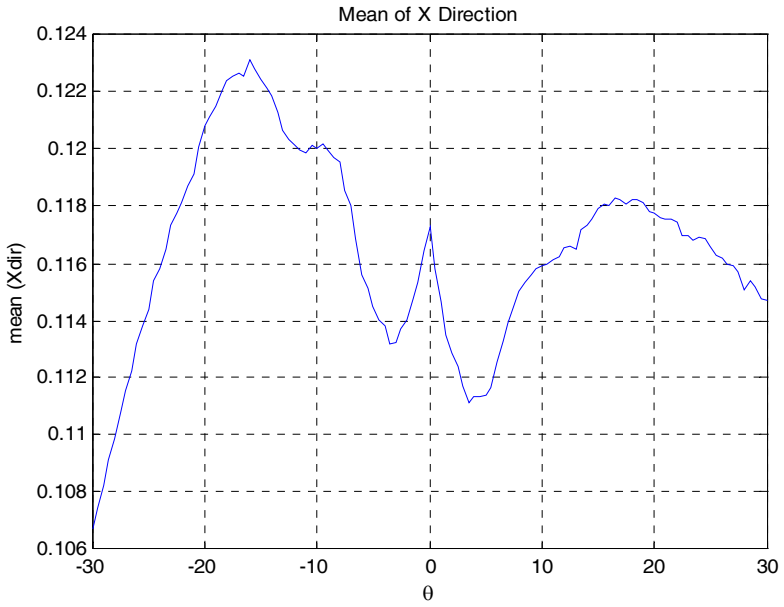


Figure 6.16 Mean of standard deviations along *x*-axis for small size image.

6.2.2.2 Mean of the Standard Deviations Parallel to the Y Axis

We plot the means of the standard deviations on the *y* axis for each rotation angle value as shown in Figure 6.17. Similarly, there is no linearity between rotation and mean along *y* axis.

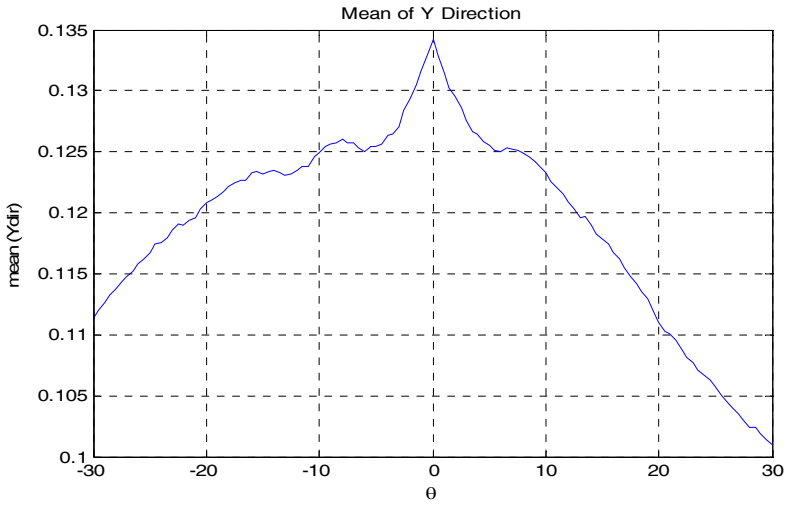


Figure 6.17 Mean of standard deviations along *y*-axis for small size image.

6.2.2.3 Mean of Standard Deviations Along the Diagonal Axes

Again, we plot the means of the standard deviations along the diaganal axes for each rotation angle value is shown in Figure 6.18. The non-linear behavior between rotation and mean along diagonal axes is valid.

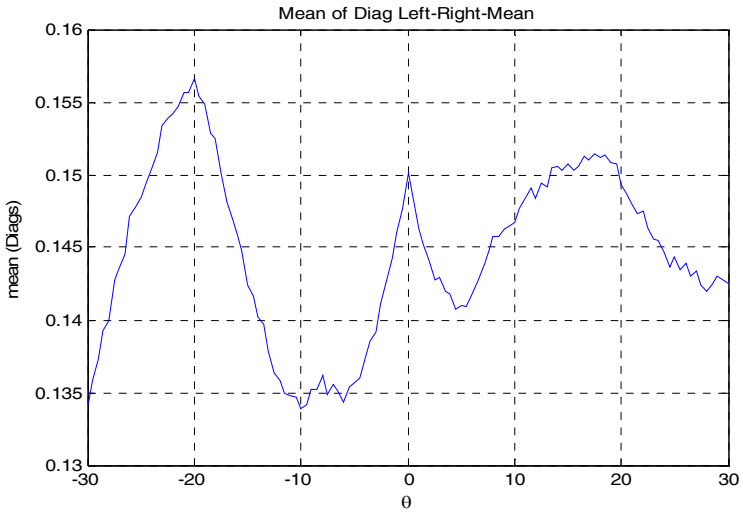


Figure 6.18 Mean of standard deviations along diagonal axes for small size image.

6.2.2.4 1-D Model Based Parameter

The distances *dx* and *dy* are chosen as one. However, in order to improve the performance the optimum disrance values must be found. As seen from the Figure 6.19, the *a versus* θ plot is not linear. By using the optimum distance values (*dx* and *dy*), it can be obtained better performance (see Figure 6.20 through 6.24).

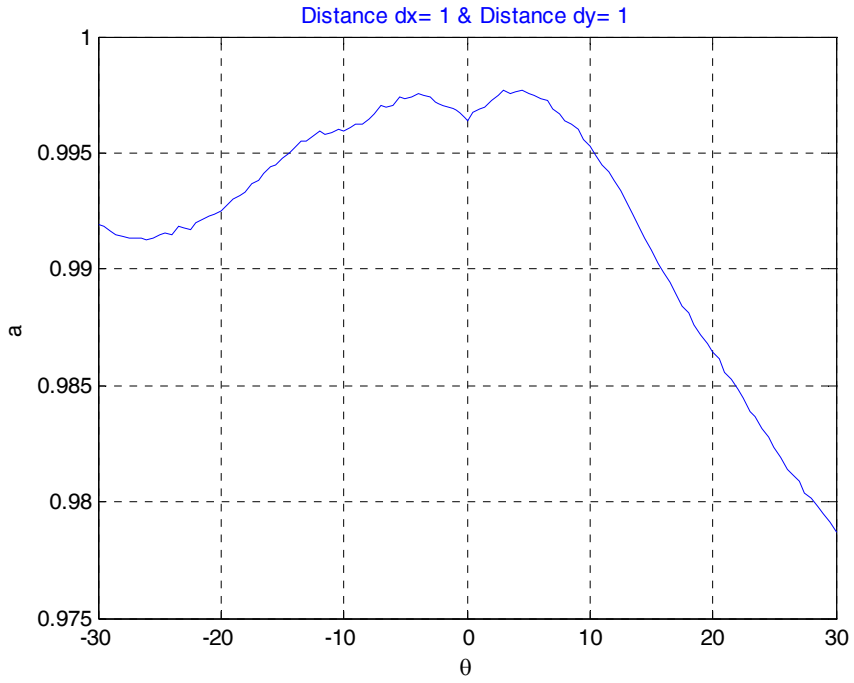


Figure 6.19 One parameter modeling for small size image.

6.2.2.5 The Performance of the Algorithm by Using Six Statistical Parameters

The similar idea is valid for the six parameter modeling. Firstly, we need to find the best candidate distance (*dx* and *dy*) variables. To do this, we follow the similar technique that we apply in the two parameter modeling. We draw the theta-error figure for the combinations of the *dx* and *dy* values. Then we applied the least square method to find the optimum distance values.

For the experimental results, we generate the look-up table in the region of -30 to 30 with the 0.5 degree step interval and we test the system in the interval of 0.1 degree. This region is mostly used in weft-straightening mechanism. However, the system accepts rest of regions too. The performance of the system in the region of -90 to 90 degree rotation is shown in Figure 6.26. The only thing is the computational

difference. Large regions take more time to generate the look-up table; on the other hand small regions take less time.

The results of *dx and dy* values can be seen for different size of images in Figure 6.20 through Figure 6.22, respectively.

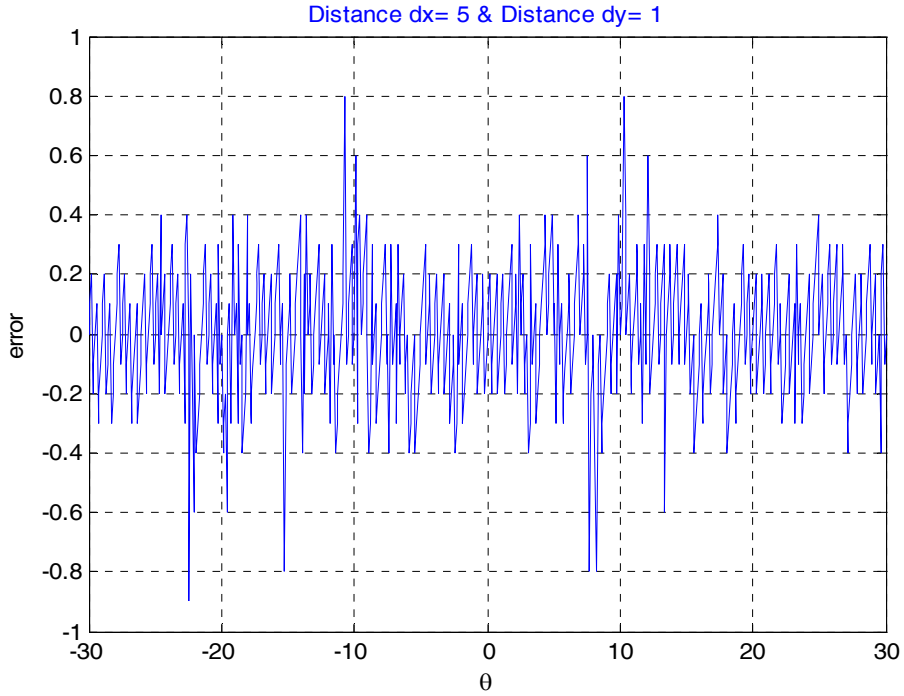


Figure 6.20 *Error versus θ* plot for small size image (*dx=5* and *dy=1*).

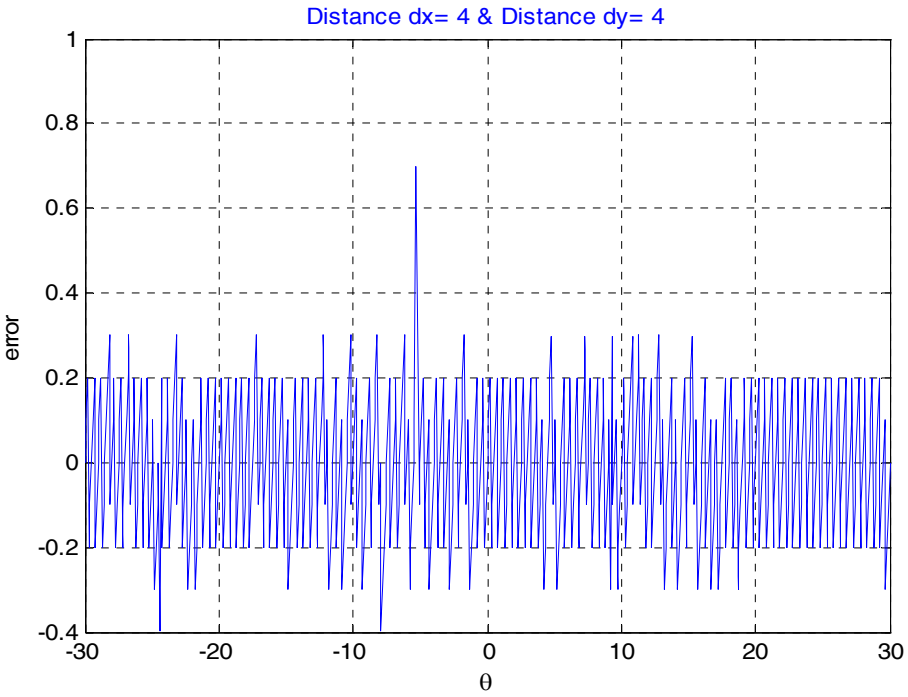


Figure 6.21 *Error versus θ* plot for middle size image (*dx=4* and *dy=4*).

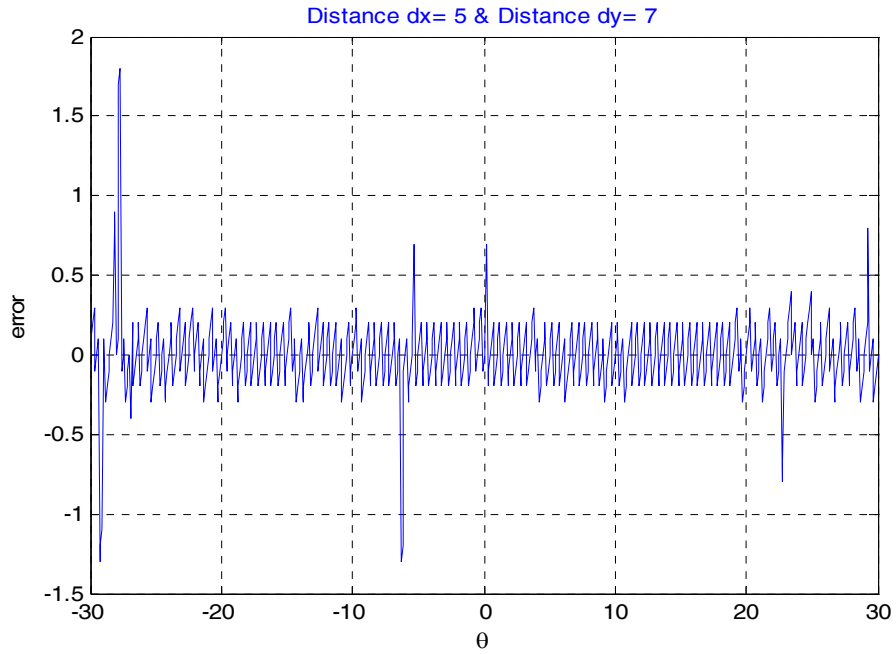


Figure 6.22 *Error versus* θ plot for large size image (*dx=5* and *dy=7*)

Figure 6.22 through in Figure 6.24 for the small size (140x100), middle size (180x150) and large size (440x325) images show the theta-error plot for the optimum dx an dy values.

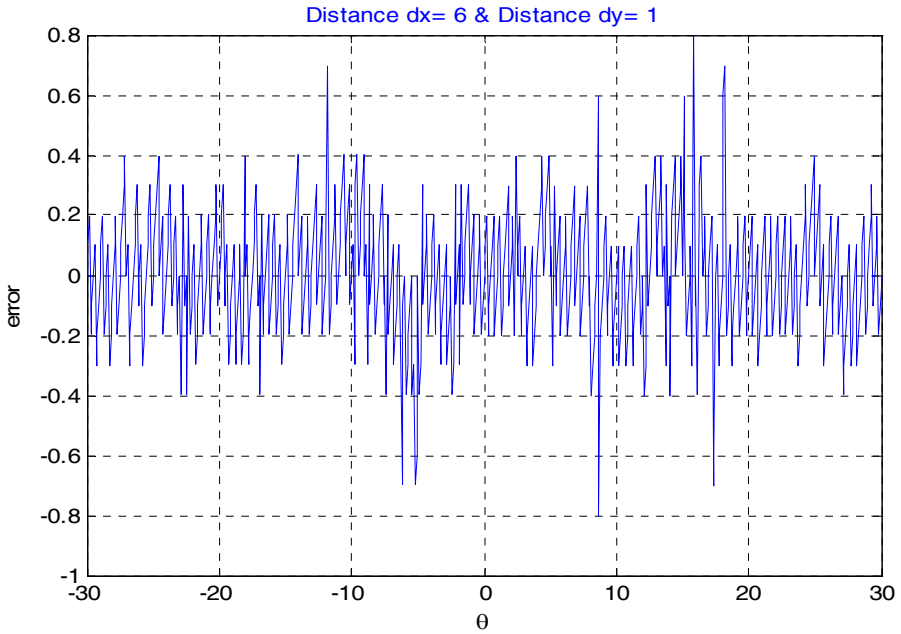


Figure 6.23 *Error versus* θ plot for optimum values of distances for small size image.

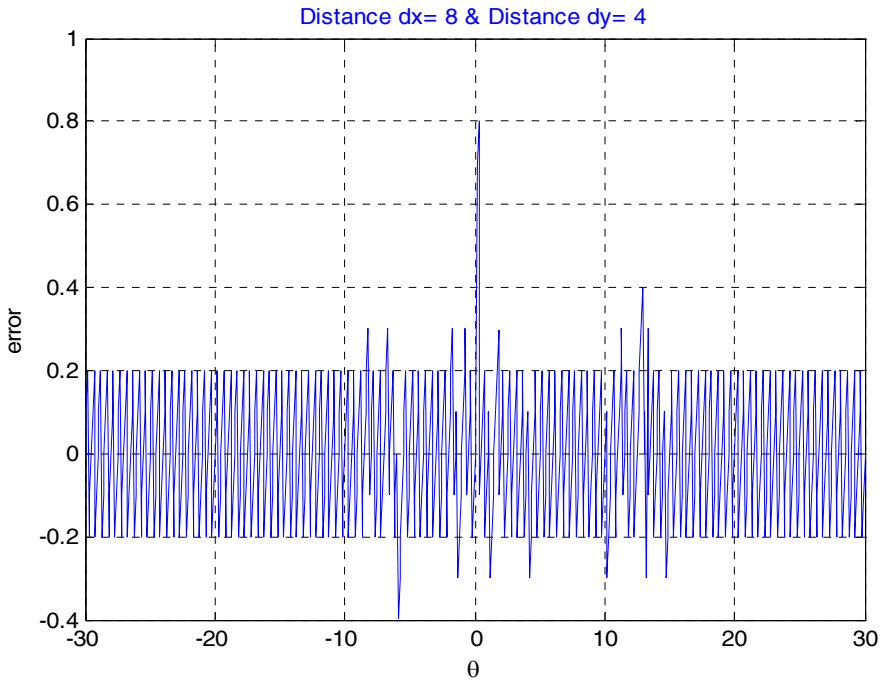


Figure 6.24 *Error versus* θ plot for the optimum distances for middle size image.

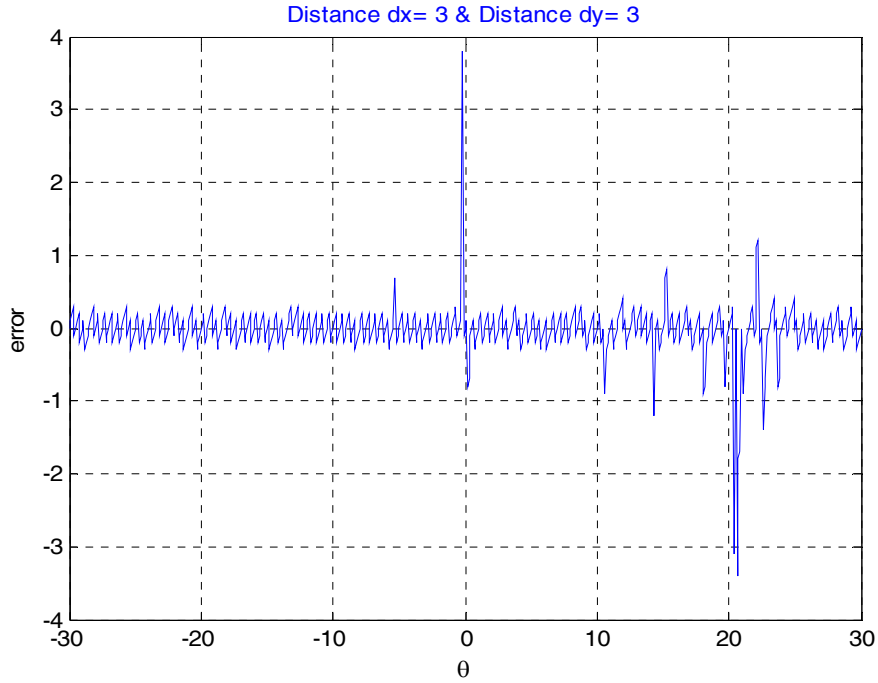


Figure 6.25 *Error versus* θ plot for the optimum distances for large size image.

Finally, to show the similarity between wide range (-90:90) and normal range (-30:30), we apply the same procedure to the middle image The results for this case are

shown in Figure 6.25. In this plot, we used the step interval as 0.3 for testing to get a clear response on the output figure.

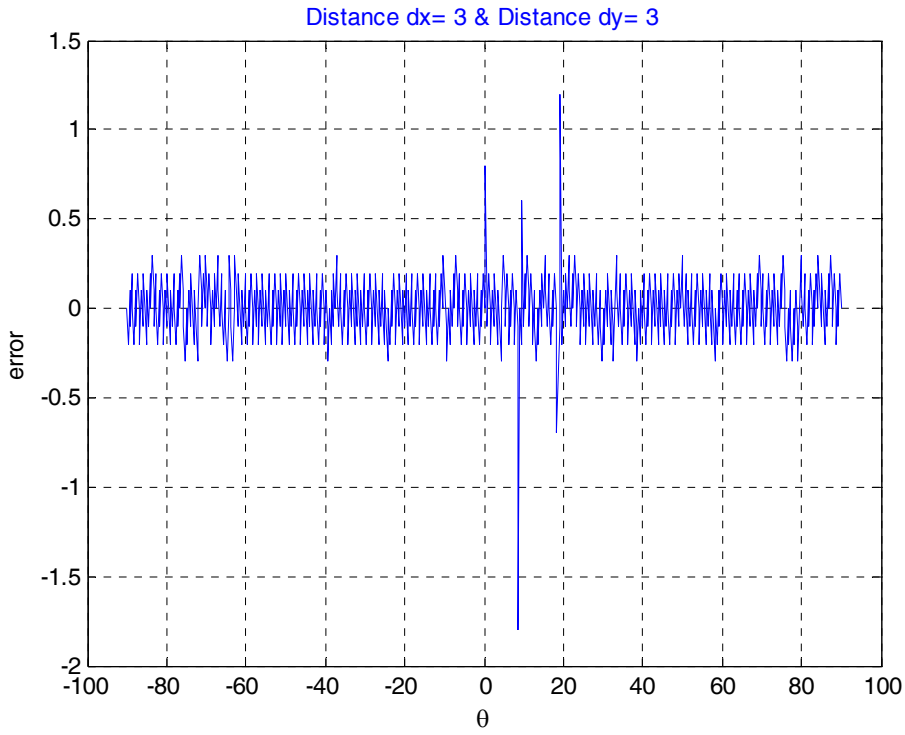


Figure 6.26 Wide range *error versus* θ plot.

CHAPTER 7

CONCLUSIONS

In this thesis, we have presented new smart camera architecture for various purposes. Specifically, a textile industrial system, weft-straightening system, was analyzed and two different rotation estimation algorithms was proposed.

The first method was based on FFT method. It was proved in Chapter 3 that rotation of an image corresponds to rotation of its FT. Based on this fact, FFT based rotation estimation algorithm procedure was explained. The brightest point tracking, searched region, and rotation estimation Matlab code was presented in Chapter 3.

Statistical parameters based rotation estimation algorithm was proposed in Chapter 4. The extraction of the statistical parameters was presented. Statistical parameters which vary significantly with rotation were best candidates for this approach. To illustrate the importance of the number of used statistical parameter this method was divided into two subsections. The first section used only two model based statistical parameters and the second section used six statistical parameters which are one parameter modeling, two parameter modeling, and means of standard deviations on the x, y, and diagonal axes.

The experimental setup of the algorithms was explained in chapter five. The developments of the algorithms and their implementations on the embedded system had been given. The Matlab codes of the algorithms had been explained in detail. The setup process for embedded Linux operating system was explained. Finally, the TCP/IP networking of the smart cameras was discussed and the related flow chart was given for connection oriented TCP/IP networking. The system calls for the networking was expressed in the related chapter.

The system performance was discussed in chapter six. The FFT based rotation estimation and statistical parameters based rotation estimation algorithm results was observed as simulations. The maximum error for FFT based method was around two degree. FFT based method was independent of the texture pattern, so it was very suitable for the industrial application. However, *"FFT based rotation estimation algorithm"* had computational difficulties (Honec et al., 2001). Therefore, it was observed that the system worked considerably sluggish with respect to statistical based method. In order to design a much faster system for the real time applications, another method called "*statistical parameters based rotation estimation algorithm*" was proposed. The maximum error for statistical parameters based method was less than one degree for three different size images. The number of the used statistical parameters improved the system performance significantly. The results showed that this method was very fast and suitable for the real time applications. On the other hand, the method was depended upon the texture pattern, so the system had to be trained for every texture pattern.

In conclusion, the result showed that each algorithm had advantages and disadvantages. The FFT based and statistical parameters based algorithms work in the region of 90 degree due to the texture pattern periodicity. However, implementation of FFT based method was easier than statistical method since it did not depend on the texture pattern. Although statistical method was required a look up table for each type of texture pattern, FFT based method did not need this kind of pre-work.

APPENDIX A

MATLAB CODE

1.1 MATLAB CODE OF THE FFT BASED METHOD

```
% Version : 1.0
% Author  : Cihan Ulas
% Date      : January, 2007
clf
clear all
close all
I=imread('k1z.bmp');n=round(size(I,1)/5)
%n=164;
threshold=0.5;
gamma=0.3;
theta =10 ; %For demonstration
subplot(1,2,1)
imshow(I);
title('Texture Image')
R = imrotate(I,theta,'bilinear','crop');
subplot(1,2,2)
imshow(R)
Im=modifyimage(I,n);
Rm=modifyimage(R,n);
title('Rotated Image (10 degree)')
figure
Ixd=fft2new(Im,threshold,gamma);
[xI yI]=findbrghtpnt(Ixd) ;
```

```
subplot(1,2,1)
imshow(Ixd);
Rxd=fft2new(Rm,threshold,gamma);
%[xR yR]=findbrghtpnt(Rxd) ;
subplot(1,2,2)
imshow(Rxd);
pause;
err=[];
figure(2)
for t=-35:2.5:35
        R = imrotate(I,t,'bilinear','crop');
        Rm=modifyimage(R,n);
        Rxd=fft2new(Rm,threshold,gamma);
        [xR yR]=findbrghtpnt(Rxd) ;
        figure(2)
        imshow(1-Rxd); hold on; plot(yR+n,-xR+n,'ro')
        title(['Threshold:' num2str(threshold) ' ' 'Gamma:' num2str(gamma) ' '…
        'Rotation:' num2str(t)]);
        theta1=180/pi*atan(xR/yR);
        theta2=180/pi*atan(xI/yI);
        theta=theta1-theta2;
        err=[err (t-theta)];
        disp(theta)
end
hold off,
figure
t=-35:2.5:35;
plot(t,err)
```

1.1.1 *Functions of The System*

```
function R=modifyimage(R,n)
        [nx,ny,nc]=size(R);
        nx2=round(nx/2);
        ny2=round(ny/2);
```

```
        R=R(nx2-n+1:nx2+n,ny2-n+1:ny2+n,2);

function Ixd=fft2new(image,tl,gamma)
        %input image=I
        %output Ixd shifted and fft taken image
        Ix=real(ifft2(abs(fft2(double(image))).^2));
        Ixmin=min(min(Ix));
        Ixmax=max(max(Ix));
        Ixd=(Ix-Ixmin)/(Ixmax-Ixmin);
        %tl=0.6;
        Ixd=Ixd-tl;
        Ixd=(Ixd+abs(Ixd))/2;
        %gamma=0.4;
        Ixd=Ixd.^gamma;
        Ixd=Ixd/max(max(Ixd));
        Ixd=fftshift(Ixd);
        %imshow(Ixd);

function [x,y]=findbrghtpnt(I)
        threshold=0.6;
        Imax=max(max(I));
        row=size(I,1);
        col=size(I,2);
        boxsize=20;
        ic=round(row/2);
        jc=round(col/2);
        ii=[];jj=[];
        x=0;y=0;
        iii=0;jjj=0;
        bp=0.1; % brightest point
        % figure
        % hold on
        quit=0;
        rmin=1000;
```

```
for i=0:boxsize
        for j=0:boxsize
                pixelval= I(-i+ic,j+jc);
                if abs(i)>3 && abs(j)>3 && pixelval>0
                        radius=sqrt(x^2+y^2);
                        if bp<pixelval %&& rmin>radius
                                bp=pixelval;
                                x=i;
                                y=j;
                                rmin=radius;
                        end
                end
        end
end

% figure
% plot(ii,jj,'*')
% hold on
% plot(x,y,'ro')
% axis([-30, 30,-30, 30])
```

1.1 MATLAB CODE OF THE STATISTICAL BASED METHOD

1.2.1 Two Parameter Model Examining

```
% Version : 1.0
% Author  : Cihan Ulas
% Date    : January 2007
clear all;
clc;
close all;
while (1)
```

```
        choice=menu('Main Menu',...
        'Get Image',...
        'Inspect for the best statistical distance',...
        'Rotate Image',....
        'Calculate Rotation',...
        'Theta-Error Plotter',...
        'Theta-Error Plotter for differet distances',...
        'Exit');
if (choice ==1)
        [file_name file_path] = uigetfile ('*.bmp');
        I = imread ([file_path,file_name]);
        %I=imread('k1.bmp');
        disp('Enter the table range for training. Example:0:1:180');
        range=-30:0.5:30;%input('Range: ');
        dx=input('Distance dx: ');
        dy=input('Distance dy: ');
        table=gettable(I,range,dx,dy)
        a1=table(1,:);
        a2=table(2,:);
        theta=range;
        figure('Name','alpha','NumberTitle','off')
        plot(theta,a1,'b')
        xlabel ('theta')
        ylabel ('alpha')
        title(['Distance dx= ',int2str(dx),' & Distance dy= ',int2str(dy)],'Color','b')
        figure('Name','beta','NumberTitle','off')
        plot(theta,a2,'b')
        xlabel ('theta')
        ylabel ('beta')
        title(['Distance dx= ',int2str(dx),' & Distance dy= ',int2str(dy)],'Color','b')
end

if (choice == 2)
        drangex=input('Distance Range for dx: ');
```

```
        drangey=input('Distance Range for dy: ');

for d1=drangex
        for d2=drangey
                table=gettable(I,range,d1,d2);
                a1=table(1,:);
                a2=table(2,:);
                theta=table(3,:);
                %%%%%%%%%%%%%%%%%%%%
                figure('Name','a1 and a2','NumberTitle','off')
                plot(theta,a1,'b',theta,a2,'r')
                xlabel ('theta')
                ylabel ('2 Parameters')
                title(['Distance=#',int2str(d1),' Distance=#',int2str(d2)],'Color','b')
                disp('Press any key for the next distance...')
                pause;
                close all;
                clc;
        end
end

end

if (choice == 3)
        disp(' Image will be rotated and a bilinear-crop method will be applied.')
        disp('--------------------------------')
        t=input('Enter the rotation angle: ');
        xr=rotandgetpar(I,t,dx,dy)
end
if (choice == 4)
        disp('--------------------------------')
        disp('Finding the rotation based on 2 parameters');
        disp('--------------------------------')
        theta=gettheta(table,xr)
```

```
end
if (choice==5)
        rot=input('Enter the table range(Example:0:0.5:180):');
        err=[] ;
        for t=rot
                xr=rotandgetpar(I,t,dx,dy);
                thetac=gettheta(table,xr);
                err=[err (t-thetac)];
        end
        disp(mean(err));
        figure
        plot(rot,err)
        axis auto
        xlabel('theta')
        ylabel('error')
        title(['Distance dx= ',int2str(dx),' & Distance dy= ',int2str(dy)],'Color','b')
end

if (choice == 6)
        rot=input('Enter the rotation range(Example:0:1:180):');
        drangex=input('Distance Range dx: ');
        drangey=input('Distance Range dy: ');
        disp('This process may take a few minutes...')
        minerr=10000;
        c=clock;
        disp(['Started at : ' int2str(c(1,4)) ':' int2str(c(1,5)) ':' int2str(c(1,6))])
        for dx=drangex
                for dy=drangey
                        table=gettable(I,range,dx,dy);    %  build new table
                        err=[] ;
                        for t=rot
                                xr=rotandgetpar(I,t,dx,dy);
                                thetac=gettheta(table,xr);
                                 err=[err (t-thetac)];
```

```
                    end
                    pwrerr=sum(err.^2);
                    if pwrerr<minerr
                        minerr=pwrerr;
                        bestdx=dx;
                        bestdy=dy;
                        besterr=err;
                    end
                    disp(dx*dy)
                    c=clock;
                    disp([int2str(c(1,4)) ':' int2str(c(1,5)) ':' int2str(c(1,6))])
        %           figure
        %           plot(rot,err)
        %           xlabel ('rot')
        %           ylabel ('err')
        %           title(['Distances  dx= ',int2str(dx),' & Distance dy=
                    ',int2str(dy)],'Color','b')
        %           %axis([min(theta) max(theta) -2 2]);
        %           axis auto
        %           saveas(gcf,[int2str(d) '.jpg']);
                end
        %           disp('Figures are exported to the application folder!');
        end
        disp(['Finished at : ' int2str(c(1,4)) ':' int2str(c(1,5)) ':' int2str(c(1,6))])
        figure
        plot(rot,besterr)
        xlabel ('rot')
        ylabel ('err')
        title(['Best Distances  dx= ',int2str(bestdx),' & Distance dy=
        ',int2str(bestdy)],'Color','b')
      %axis([min(theta) max(theta) -2 2]);
        axis auto
end
if (choice == 7)
```

```
        %clear all;
        clc;
        close all;
        return;
end
end
```

1.2.1.1 Functions for Two Parameter Modeling

```
function table_real=gettable(I,range,dx,dy)
        n=round(size(I,1)/4);
        xr=[];
        theta=[];
        for t=range
                XR = imrotate(I,t,'bilinear','crop');
        %       XR=myrotate(I,t);
                XR=modifyimage(XR,n);XR=double(XR);
        %       XR=XR+0.05*randn(size(XR,1),size(XR,2));
                xr=[xr getparameters(XR,dx,dy)];
                theta=[theta t];
        end
        table_real=[xr(1,:);xr(2,:);theta];

function R=modifyimage(R,n)
                [nx,ny,nc]=size(R);
                nx2=round(nx/2);
                ny2=round(ny/2);
                R=R(nx2-n+1:nx2+n,ny2-n+1:ny2+n,2);

function x=getparameters(X,dx,dy)
        [M,N]=size(X);
        myA11=sum(sum(X([1:M-dx],[dy+1:N]).*X([1:M-dx],[dy+1:N])));
        myA12=sum(sum(X([1:M-dx],[dy+1:N]).*X([dx+1:M],[1:N-dy])));
        myA21=sum(sum(X([dx+1:M],[1:N-dy]).*X([1:M-dx],[dy+1:N])));
```

```
myA22=sum(sum(X([dx+1:M],[1:N-dy]).*X([dx+1:M],[1:N-dy])));
myB11=sum(sum(X([dx+1:M],[dy+1:N]).*X([1:M-dx],[dy+1:N])));
myB21=sum(sum(X([dx+1:M],[dy+1:N]).*X([dx+1:M],[1:N-dy])));
A=[myA11 myA12;myA21 myA22] ;
B=[myB11;myB21];
x=inv(A)*B;

function xr=rotandgetpar(I,t,dx,dy)
    n=round(size(I,1)/4);
    XR = imrotate(I,t,'bilinear','crop');
    XR=modifyimage(XR,n);XR=double(XR);
    xr=getparameters(XR,dx,dy);

function theta=gettheta(table,xr)
    [m,n]=size(table);
     minerror=10;
     for i=1:n
          xR=table(1:2,i);
          error =norm(xR-xr);
          if error < minerror
          minerror=error;
          theta=table(3,i);
          end
    end
```

1.2.2 Five Parameter Model Examining

```
% Version : 1.0
% Author  : Cihan Ulas
clear all;
clc;
close all;
while (1)
    choice=menu('Main Menu',...
```

```
        'Get Image',...
        'Inspect for the best statistical distance',...
        'Rotate Image',....
        'Calculate Rotation',...
        'Theta-Error Plotter',...
        'Theta-Error Plotter for differet distances',...
        'Exit');
if (choice ==1)
        [file_name file_path] = uigetfile ('*.bmp');
        I = imread ([file_path,file_name]);
        %I=imread('k2.bmp');
        disp('Enter the table range for training. Example:0:1:180');
        range=-30:0.5:30;%input('Range: ');
        dx=input('Distance dx: ');
        dy=input('Distance dy: ');
        table=gettable(I,range,dx,dy)
        xdir=table(1,:);
        ydir=table(2,:);
        diag=table(3,:);
        mod1=table(4,:);
        mod21=table(5,:);
         mod22=table(6,:);
        figure('Name','Mean of X Direction','NumberTitle','off')
        plot(range,xdir,'b')
        xlabel ('theta')
        ylabel ('mean(xdir)')
         title('Mean of X Direction')
         %%%%%%%%%%%%%%%%%%%%%%%%
         figure('Name','Mean of Y Direction','NumberTitle','off')
         plot(range,ydir,'b')
         xlabel ('theta')
         ylabel ('mean(ydir)')
        title('Mean of Y Direction')
        %%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
        figure('Name','Mean of Diag Left-Right-Mean','NumberTitle','off')
         plot(range,diag,'b')
        xlabel ('theta')
        ylabel ('mean(diags)')
        title('Mean of Diag Left-Right-Mean')
        %%%%%%%%%%%%%%%%%%%%%%%%
        figure('Name','1 Model Parameter','NumberTitle','off')
        plot(range,mod1,'b')
        xlabel ('theta')
        ylabel ('a0')
        title(['Distance dx= ',int2str(dx),' & Distance dy= ',int2str(dy)],'Color','b')
        %%%%%%%%%%%%%%%%%%%%%%%%
        figure('Name','2 Model Parameters','NumberTitle','off')
        plot(range,mod21,'b')
        xlabel ('theta')
        ylabel ('a1')
        title('Mean of Diag Left-Right-Mean')
        title(['Distance dx= ',int2str(dx),' & Distance dy= ',int2str(dy)],'Color','b')
        %%%%%%%%%%%%%%%%%%%%%%%%
        figure('Name','2 Model Parameters','NumberTitle','off')
        plot(range,mod22,'b')
        xlabel ('theta')
        ylabel ('a2')
        title('Mean of Diag Left-Right-Mean')
        title(['Distance dx= ',int2str(dx),' & Distance dy= ',int2str(dy)],'Color','b')
end
if (choice == 2)
        drangex=input('Distance Range for dx: ');
        drangey=input('Distance Range for dy: ');

        for d1=drangex
                for d2=drangey

                        table=gettable(I,range,d1,d2);
```

```
                    mod1=table(4,:);
                    mod21=table(5,:);
                    mod22=table(6,:);
                    figure('Name','1 Model Parameter','NumberTitle','off')
                    plot(range,mod1,'b')
                    xlabel ('theta')
                    ylabel ('a0')
                    title(['Distance dx= ',int2str(d1),' & Distance dy=…
                   ',int2str(d2)],'Color','b')
                     %%%%%%%%%%%%%%%%%%%%%%%%
                    figure('Name','2 Model Parameters','NumberTitle','off')
                    plot(range,mod21,'b')
                    xlabel ('theta')
                    ylabel ('a1')
                    title('Mean of Diag Left-Right-Mean')
                    title(['Distance dx= ',int2str(d1),' & Distance dy= …
                   ',int2str(d2)],'Color','b')
                    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    figure('Name','2 Model Parameters','NumberTitle','off')
                    plot(range,mod22,'b')
                    xlabel ('theta')
                    ylabel ('a2')
                    title('Mean of Diag Left-Right-Mean')
                    title(['Distance dx= ',int2str(d1),' & Distance dy=…
                   ',int2str(d2)],'Color','b')
                    disp('Press any key for the next distance...')
                    pause;
                    close all;
                    clc;
                end
            end

        end
```

```
if (choice == 3)
        disp(' Image will be rotated and a bilinear-crop method will be applied.')
        disp('--------------------------------')
        t=input('Enter the rotation angle: ');
        xr=rotandgetpar(I,t,dx,dy)
end
if (choice == 4)
        disp('--------------------------------')
        disp('Finding the rotation based on 2 parameters');
        disp('--------------------------------')
        theta=gettheta(table,xr)
end
if (choice==5)
        rot=input('Enter the table range(Example:0:0.5:180):');
        err=[] ;
        for t=rot
                xr=rotandgetpar(I,t,dx,dy);
                thetac=gettheta(table,xr);
                err=[err (t-thetac)];
        end
        disp(mean(err));
        figure
        plot(rot,err)
        axis auto
        xlabel('theta')
        ylabel('error')
        title(['Distance dx= ',int2str(dx),' & Distance dy= ',int2str(dy)],'Color','b')
end

if (choice == 6)
        rot=input('Enter the rotation range(Example:0:1:180):');
        drangex=input('Distance Range dx: ');
        drangey=input('Distance Range dy: ');
```

```
disp('This process may take a few minutes...')
minerr=10000;
c=clock;
for dx=drangex
        for dy=drangey
                table=gettable(I,range,dx,dy);    %  build new table
                err=[] ;
                for t=rot
                        xr=rotandgetpar(I,t,dx,dy);
                       thetac=gettheta(table,xr);
                        err=[err (t-thetac)];
                end
                pwrerr=sum(err.^2);
                if pwrerr<minerr
                        minerr=pwrerr;
                        bestdx=dx;
                        bestdy=dy;
                        besterr=err;
                end
                disp(dx*dy)
                c=clock;
                disp([int2str(c(1,4)) ':' int2str(c(1,5)) ':' int2str(c(1,6))])
                %saveas(gcf,[int2str(d) '.jpg']);
        end
end
        disp(['Finished at : ' int2str(c(1,4)) ':' int2str(c(1,5)) ':' …
        int2str(c(1,6))])
        figure
plot(rot,besterr)
xlabel ('rot')
ylabel ('err')
title(['Best Distances  dx= ',int2str(bestdx),' & Distance dy=
',int2str(bestdy)],'Color','b')
%axis([min(theta) max(theta) -2 2]);
```

```
        axis auto
end

if (choice == 7)
        %clear all;
        clc;
        close all;
        return;
end
end
```

1.2.2.1 Functions for Five Parameter Modeling

```
function table_real=gettable(I,range,dx,dy)
        n=round(size(I,1)/4);
        table=[];
        theta=[];
        for t=range
                XR = imrotate(I,t,'bilinear','crop');
                XR=modifyimage(XR,n);XR=double(XR);
        %       XR=XR+0.05*randn(size(XR,1),size(XR,2));
                xdir1=xdir(XR);
                ydir1=ydir(XR);
                diagdir1=diagdir(XR);
                mod1=getmodpar(XR,dx,dy);
                mod2=getparameters(XR,dx,dy);
                xr=[xdir1 ydir1 diagdir1 mod1 mod2];
                table=[table;xr] ;
        end
        table_real=[table range']';

function xr=rotandgetpar(I,t,dx,dy)
        XR = imrotate(I,t,'bilinear','crop');
        xr=getpars(XR,dx,dy);
```

```
function xr=getpars(XR,dx,dy)
        n=round(size(XR,1)/4);
        XR=modifyimage(XR,n);XR=double(XR);
        xdir1=xdir(XR);
        ydir1=ydir(XR);
        diagdir1=diagdir(XR);
        mod1=getmodpar(XR,dx,dy);
        mod2=getparameters(XR,dx,dy);
        xr=[xdir1;ydir1;diagdir1;mod1;mod2'];

function meanstd= xdir(I)
        [M,N]=size(I);
        stds=[];
        for i=1:M
                stds= [stds std(I(i,:))];
        end
        meanstd=mean(stds)/mean(mean(I));

function meanstd= ydir(I)
        [M,N]=size(I);
        stds=[];
        for i=1:N
                stds= [stds std(I(:,i))];
        end
        meanstd=mean(stds)/mean(mean(I));

function meanstd= diagdir(I)
        [M,N]=size(I) ;
        Id1=[];
        %%Get Diag Forward Elements
        for i=1:M
                Id1= [Id1 I(i,i)];
        end
```

```
        stdF=std(Id1);
        %%Get Diag Backward Elements
        Id2=[];
        for i=1:M
                Id2= [Id2 I(i,M-i+1)];
        end
        stdB=std(Id2);
        %%Their mean
        meanstd=(stdF+stdB)/2;meanstd=meanstd/mean(mean(I));
        %all=[meanstd;stdF;stdB]/mean(mean(I));

function x=getmodpar(X,dx,dy)
[M,N]=size(X);
        A=sum(sum(X([1:M-dx],[1:N-dy]).*X([dx+1:M],[dy+1:N])));
B=sum(sum(X([1:M-dx],[1:N-dy]).*X([1:M-dx],[1:N-dy])));

function x=getparameters(X,dx,dy)
        [M,N]=size(X);

        myA11=sum(sum(X([1:M-dx],[dy+1:N]).*X([1:M-dx],[dy+1:N])));
        myA12=sum(sum(X([1:M-dx],[dy+1:N]).*X([dx+1:M],[1:N-dy])));
        myA21=sum(sum(X([dx+1:M],[1:N-dy]).*X([1:M-dx],[dy+1:N])));
        myA22=sum(sum(X([dx+1:M],[1:N-dy]).*X([dx+1:M],[1:N-dy])));
        myB11=sum(sum(X([dx+1:M],[dy+1:N]).*X([1:M-dx],[dy+1:N])));
        myB21=sum(sum(X([dx+1:M],[dy+1:N]).*X([dx+1:M],[1:N-dy])));
        A=[myA11 myA12;myA21 myA22] ;
        B=[myB11;myB21];
        x=inv(A)*B;
        x=x';

function theta=gettheta(table,xr)
        [m,n]=size(table);
        minerror=10;
        Wi=[1 1 1 1 1 1];
```

```
for i=1:n
        xR=table(1:6,i);
        error =Wi.*norm(xR-xr);
        if error < minerror
                minerror=error;
                theta=table(7,i);
                 %theta=table(3,i);
        end
end
```

1.2.3 Statistical Method Examining by Using a Camera

```
% Version : 1.0
% Author  : Cihan Ulas
clear all;
clc;
close all;
while (1)
        choice=menu('Main Menu',...
        'Start Capturing',...
        'Inspect for the best distance',....
        'Rotate Image',....
        'Calculate Rotation',...
        'Theta-Error Plotter',...
        'Theta-Error Plotter for differet distances',...
        'Work in Real Time',...
        'Exit');
if (choice ==1)
        vidobj = videoinput('winvideo', 1);
        preview(vidobj)
        disp('Enter the range for training. Example:0:1:180');
        range=-20:.5:20;%input('Range: ');
        dx=input('Distance dx: ');
        dy=input('Distance dy: ');
```

```
I=getsnapshot(vidobj);
table=gettable(I,range,dx,dy);
disp('Table has been created')
xdir=table(1,:);
ydir=table(2,:);
diag=table(3,:);
mod1=table(4,:);
mod21=table(5,:);
mod22=table(6,:);
figure('Name','Mean of X Direction','NumberTitle','off')
plot(range,xdir,'b')
xlabel ('theta')
ylabel ('mean(xdir)')
title('Mean of X Direction')
%%%%%%%%%%%%%%%%%%%%%%%%
figure('Name','Mean of Y Direction','NumberTitle','off')
plot(range,ydir,'b')
xlabel ('theta')
ylabel ('mean(ydir)')
title('Mean of Y Direction')
%%%%%%%%%%%%%%%%%%%%%%%%
figure('Name','Mean of Diag Left-Right-Mean','NumberTitle','off')
plot(range,diag,'b')
xlabel ('theta')
ylabel ('mean(diags)')
title('Mean of Diag Left-Right-Mean')
%%%%%%%%%%%%%%%%%%%%%%%%
figure('Name','1 Model Parameter','NumberTitle','off')
plot(range,mod1,'b')
xlabel ('theta')
ylabel ('a0')
title(['Distance dx= ',int2str(dx),' & Distance dy= ',int2str(dy)],'Color','b')
%%%%%%%%%%%%%%%%%%%%%%%%
figure('Name','2 Model Parameters','NumberTitle','off')
```

```
        plot(range,mod21,'b')
        xlabel ('theta')
        ylabel ('a1')
        title('Mean of Diag Left-Right-Mean')
        title(['Distance dx= ',int2str(dx),' & Distance dy= ',int2str(dy)],'Color','b')
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%
        figure('Name','2 Model Parameters','NumberTitle','off')
        plot(range,mod22,'b')
        xlabel ('theta')
        ylabel ('a2')
        title('Mean of Diag Left-Right-Mean')
        title(['Distance dx= ',int2str(dx),' & Distance dy= ',int2str(dy)],'Color','b')
end
if (choice == 2)
        drangex=input('Distance Range for dx: ');
        drangey=input('Distance Range for dy: ');
        for d1=drangex
                for d2=drangey
                        table=gettable(I,range,d1,d2);
                        mod1=table(4,:);
                        mod21=table(5,:);
                        mod22=table(6,:);
                        figure('Name','a0','NumberTitle','off')
                        plot(range,mod1,'b')
                        xlabel ('theta')
                        ylabel ('2 Parameters')
                        title(['Distance=#',int2str(d1),'…
                        Distance=#',int2str(d2)],'Color','b')
                         %%%%%%%%%%%%%%%%%%%%%%%%
                        figure('Name','a1 and a2','NumberTitle','off')
                        plot(theta,mod21,'b',theta,mod22,'r')
                        xlabel ('theta')
                        ylabel ('2 Parameters')
                        title(['Distance=#',int2str(d1),'…
```

```
                    Distance=#',int2str(d2)],'Color','b')

                    disp('Press any key for the next distance...')
                    pause;
                    close all;
                    clc;
                end
            end
end

if (choice == 3)
            disp(' Image will be rotated and a bilinear-crop method will be applied.')
            disp('---------------------------------')
            t=input('Enter the rotation angle: ');
            %       dist1=input('Enter dx:');
            %       dist2=input('Enter dy:');
            xr=rotandgetpar(I,t,dx,dy)
end

if (choice == 4)
            disp('---------------------------------')
            disp('Finding the rotation based on 2 parameters');
            disp('---------------------------------')
            theta=gettheta(table,xr)
end

if (choice==5)
            rot=input('Enter the table range(Example:0:0.5:180):');
            err=[] ;
            for t=rot
                        xr=rotandgetpar(I,t,dx,dy);
                        thetac=gettheta(table,xr);
                        err=[err (t-thetac)];
            end
```

```
        disp(mean(err));
        figure
        plot(rot,err)
        axis auto
        xlabel('theta')
        ylabel('error')
        title(['Distance dx= ',int2str(dx),' & Distance dy= ',int2str(dy)],'Color','b')
end

if (choice == 6)
        rot=input('Enter the rotation range(Example:0:1:180):');
        drangex=input('Distance Range dx: ');
        drangey=input('Distance Range dy: ');
        disp('This process may take a few minutes...')
        for dx=drangex
                for dy=drangey
                        table=gettable(I,range,dx,dy);    %  build new table
                        err=[] ;
                        for t=rot
                                xr=rotandgetpar(I,t,dx,dy);
                                thetac=gettheta(table,xr);
                                 err=[err (t-thetac)];
                        end
                        figure
                        plot(rot,err)
                        xlabel ('rot')
                        ylabel ('err')
                        title(['Distance dx= ',int2str(dx),' & Distance dy=…
                        ',int2str(dy)],'Color','b')
                        %axis([min(theta) max(theta) -2 2]);
                        axis auto
                        disp('.')
                        %saveas(gcf,[int2str(d) '.jpg']);
                end
```

```
                %disp('Figures are exported to the application folder!');
        end
end

if (choice == 7)
%           range=input('Range: ');
%           dx=input('Distance: ');
%           dy=input('Distance: ');
        while(1)
                xpars=[];
                for j=1:10 %%get avarage.
                        I = getsnapshot(vidobj);
                        xr=getpars(I,dx,dy);
                        xpars=[xpars xr];
                end
                theta=gettheta(table,mean(xpars')') ;
                disp(theta);
                %pause(0.5)
        end
end
if (choice == 8)
        %clear all;
        clc;
        close all;
        delete(vidobj);
        clear vidobj
        return;
end
end
```

REFERENCES

Bramberger, M., Doblander A., Maier A., Rinner, B., and Schwach, H., "Distributed Embedded Smart Cameras for Surveillance Applications", *IEEE Computer Society*, Vol. 39, No. 2, pp. 68-75, February 2006.

Brayer, J. M., *"Introduction to Fourier Transforms for image processing"*, 2006, http://www.cs.unm.edu/~brayer/vision/fourier.html

Broers, H., "Face Detection and Recognition on a Smart Camera", *Proceedings of Acivs 2004,* September 2004.

Caelli, T.M., "An Adaptive Computational Model for Texture Segmentation", *Transactions and Cybernetics*, Vol.18, No. 1, 1988.

Chellappa, R. and Chatterjee, S., "Classification of Textures Using Gaussian Markov Random Fields", *IEEE Transaction on Acoustics, Speech, and Signal Processing*, Vol. 33, No. 4, August 1985.

Cheng Y.C. and Chen S.Y., "Image Classification Using Color, Texture and Regions", *Image and Vision Computing*, Vol.21, No.9, pp. 259-276, September 2003.

Chetverikov, D., "Pattern Regularity as a Visual Key", *Image and Vision Computing*, Vol. 18., pp. 975-985, March 2000.

Chetverikov, D. and Gede, K., "Textures and Structural Defects", *Lecture Notes in Computer Science*, Vol.1296, pp. 167-174, 1997.

Daugman, J.G., "Two-dimensional spectral analysis of cortical receptive field profiles", *Vision Research*, Vol. 20, pp. 847-856, 1980.

Dold, J. "The Role of a Digital Intelligent Camera in Automating Industrial Photogrammetry", *Photogrammetric Record*, Vol. 16, pp. 199-212, October 1998.

Epifanio, I. and Ayala, G., "A Random Set View of Texture Classification", *IEEE Transactions on Image Processing*, Vol. 11, No. 8, 2002.

Fu, K.S., *Syntactic Pattern Recognition and Applications*, Prentice-Hall, New Jersey, 1982.

Foote, J. and Kimber, D., “FlyCam: Practical Panoramic Video and Automatic Camera Control”, 2000 IEEE International Conference, New York, 30 July – 2 August 2000, Vol.3, pp 1419-1422, ICME’ 2000., 2000.

Frahm, J.M., Koser, K., and Koch, R., “Pose Estimation for Multi-camera Systems”, *Lecture Notes in Computer Science*, Vol. 3175, pp. 286-293, 2004.

Gustafson, J. L., “Reevaluating Amdahl's law”, *Communications of the ACM*, Vol. 31, pp. 532-533, 1988.

Hammerstrom, D.W, Henry, W., and Kuhn, M., "The CNAPS Architecture for Neural Network Emulation", in Przytula, K.W. and Prasanna Kumar, V.K. (Eds.), Prentice-Hall, *Englewood Cliffs*, pp. 107-138., 1993.

Hammerstrom, D.W. and Lulich, D.P, “Image Processing Using One-Dimensional Processor Arrays”, *Proceedings of the IEEE*, Vol. 84, pp. 1005-1018, July 1996.

Haralick, R. M., K. Shanmugam, and I. Dinstein, “Textural Features for Image Classification”, *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-3, pp. 610-621, 1973.

Harris, F.J., Member, IEEE, “One of the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform”, Proceeding of the IEEE, Vol. 66, No. 1, pp. 51-83, 1978.

Hjelmas, E. and Low B. “Face Detection: A Survey.” *Computer Vision and Image Understanding*, Vol. 83, pp. 236-274, September 2001.

Hoffman, T., Puzicha, J., IEEE, Buhmann, J. M., “Unsupervised Texture Segmentation in a Deterministic Annealing Framework”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol.20, No.8, 1998.

Honec, J., Honec, P., Petrovsky, P., Valach, S., Brambor, J., “Parallel 2-D FFT Implementation With DSPs”, 13th Int. Conference on Process Control, 2001.

Illingworth, J., and Kittler, J., “A Survey of the Hough Transform”, CVGIP, Vol. 44, pp. 87-116, 1988.

Jain, A.K. and F. Farrokhnia. “Unsupervised Texture Segmentation Using Gabor Filters”, *Pattern Recognition*, Vol. 24, No.12, pp.1167-1186, December 1991.

Kleihorst, R.P., Abbo, A.A., van Der Avoird, A., Op de Beeck, M.J.R., Sevat, L., Wielage, P., van Veen, R., and van Herten, H., “Xetal: a Low-Power High-Performance Smart Camera Processor”, 2001 IEEE International Symposium, Sydney-NSW, 6 June-9 June 2001, Vol. 5, pp. 215-218, 2001.

Leinhart, R. and Maydt J., “An Extended Set of Haar-Like Features for Rapid Object Detection”, *IEEE ICIP*, Vol.1, pp. 900-9003, 2002.

Lu, F. and Milios, E.E., "Robot Pose Estimation in Unknown Environments by Matching 2D Range Scans", Vol.18, pp. 249-275, 1997.

Lu, T., Design and Analysis of a Real-Time Video Human Gesture Recognition System, Ph.D. Thesis, Princeton University, 2004.

Maclean, W.J., "An Evaluation of the Suitability of FPGAs for Embedded Vision Systems", Computer Vision and Pattern Recognition, 2005 IEEE Computer Society, Vol. 3, pp. 131-131, 2005.

Majoor, T., Face Detection Using Color Based Region of Interest Selection, Ms. Thesis, University of Amsterdam, 2000.

Malik, J. and P. Perona, "Preattentive Texture Discrimination with Early Vision Mechanisms", *Journal of the Optical Society of America, Series A*, Vol. 7, pp. 923-932, 1990.

Markandey, V., and Rao D. "TMS320DM642 Technical Overview", Application Report SPRU615, September 2002.

Marques J. S. and Abrantes, A.J., "Shape Alignment - Optimal Initial Point and Pose Estimation", Pattern Recognition Letters, Vol. 18, pp. 49-53, February 1997

Ohanian, P. P. and Dubes, R. C., "Performance Evaluation for Four Classes of Textural Features", *Pattern Recognition*, Vol. 25, pp. 819-833, 1992.

Panda, R. and Chatterji, B.N, **"**Unsupervised Texture Segmentation Using Tuned Filters in Gaborian Space", *Pattern Recognition Letters*, Vol.18, pp. 445 – 453, 1997.

Porter, R., and Canagarah, N., "Robust Rotation-Invariant Texture Classification: Wavelet, Gaborfilter and GMRF Based Schemes", *Vision, Image and Signal Processing, IEE Proceedings*, Vol. 144, pp. 180-188., 1997.

Reed, T. R. and Wechsler H., "Segmentation of Textured Images and Gestalt Organization Using Spatial/Spatial-Frequency Representations", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, pp. 1-12, 1990.

Sato, K., Evans, B.L., Aggarwal, J. K., "Designing an Embedded Video Processing Camera Using a 16-bit Microprocessor for a Surveillance System", *Journal of VLSI Signal Processing Systems*, Vol. 42, pp. 57-68, January 2006.

Sezer, O.G., Ercil, A., and Ertuzun, A., "Using Perceptual Relation of Regularity and Anisotropy in the Texture With Independent Component Model for Defect Detection", *Pattern Recognition*, Vol. 40, pp. 121-133, 2007)

Shakhnarovich, G., Viola, P., and Darrell, T., "Fast Pose Estimation With Parameter-Sensitive Hashing", *Proceedings of the Ninth IEEE International Conference on Computer Vision*, Vol. 2, pp. 750- 757, 2003.

Simon, D. A., Hebert, M., and Kanade, T., "Real-Time 3-D Pose Estimation Using a High-Speed Range Sensor", IEEE International Conference on Robotics and Automation, Vol.3, pp. 2235-2241.

Smith, G. and Burns, I., "Measuring texture classification algorithms", *Pattern Recognition Letters*, Vol.18, pp. 1495-1501, 1997.

Sonka, M., Hlavac, V., and Boyle, R., *Analysis and Machine Vision*, Chapman & Hall, London, 1993.

Souza, A., "Smart Cameras as Embedded Systems", ICCA 03, Vol.4, pp. 105-112, 2003.

Stevens, W., *Unix Network Programming First Edition*, Prentice Hall, 1990

Sun, J., Gu, D., Zhang, S., and Chen, Y., "Hidden Markov Bayesian Texture Segmentation Using Complex Wavelet Transform", *IEE Proc.-Vis. Image Signal Process*, Vol. 151, No. 3, June 2004.

Tasdizen T., Tarel, J. P., and Cooper D.B., "Improving the Stability of Algebraic Curves for Applications", *IEEE Transactions on Image Processing*, Vol. 9, pp. 405-416, March 2000.

Texas Instruments, "TMS320DM642 Technical Overview," Application Report SPRU615, Sep. 2002.

TriMedia Technologies, 2003, http://www.trimedia.com

Tuceryan, M. and Jain, A. K., "Texture Segmentation Using Voronoi Polygons", *IEEE* Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-12, pp. 211-216, 1990.

Tuceryan, M. and Jain A. K, "Texture Analysis", *Handbook of Pattern Recognition & Computer Vision*, pp. 207-248, 1998.

Turner, M.R., "Texture Discrimination by Gabor Functions", *Biological Cybernetics*, Vol. 55, pp. 71-82, 1986.

Unsalan C., "A Model Based Approach for Pose Estimation and Rotation Invariant Object Matching", *Pattern Recognition Letters*, Vol. 28, pp. 49-57, January 2007.

Unser, M. and M. Eden, "Nonlinear Operators for Improving Texture Segmentation Based on Features Extracted by Spatial Filtering," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-20, No.4, pp. 804-815, Jul/August 1990.

Zucker, S. W., “Toward a model of Texture”, *Computer Graphics and Image Processing*, Vol. 5, pp. 190-202, 1976.

Viola, P. and Jones, M., "Rapid Object Detection Using a Boosted Cascade of Simple Features", *IEEE Computer Conference on Computer Vision and Pattern Recognition*, Vol.1, pp. 511-518, December 2001

Whitney, A., “A Direct Method of Nonparametric Measurement Selection”, IEEE Trans. on Computers, Vol. 20, pp. 1100-1103, 1971.

Wolf, W., Ozer, B. and Lv, T. “Smart Cameras as Embedded Systems”, *IEEE Computer*, Vol. 35, pp. 48-53, September 2002.

Yaghmour, K., Building Embedded Linux Systems, O’Reilly Media, California, 2003.