A Novel Global Pattern Recognition Algorithm

Joseph M. Stoffa

West Virginia University

Follow this and additional works at: https://researchrepository.wvu.edu/etd

Recommended Citation
https://researchrepository.wvu.edu/etd/3016

This Dissertation is protected by copyright and/or related rights. It has been brought to you by the The Research Repository @ WVU with permission from the rights-holder(s). You are free to use this Dissertation in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you must obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/or on the work itself. This Dissertation has been accepted for inclusion in WVU Graduate Theses, Dissertations, and Problem Reports collection by an authorized administrator of The Research Repository @ WVU. For more information, please contact research.repository@mail.wvu.edu.
A Novel Global Pattern Recognition Algorithm

By
Joseph M. Stoffa

Dissertation submitted to the
College of Engineering and Mineral Resources
At West Virginia University
In partial fulfillment of the requirements
For the degree of

Doctor of Philosophy
In
Chemical Engineering

Dr. Alfred H. Stiller, Chair
Dr. Charles Jaffe
Dr. Joseph Shaeiwitz
Dr. Charter Stinespring
Dr. Richard Turton

Department of Chemical Engineering
Morgantown, West Virginia
2010

Keywords: Pattern Recognition, Markov Chain, Monte Carlo Integration, Fingerprint Identification, Random Walk, FVC2000
Copyright 2010 Joseph M. Stoffa
Abstract

A Novel Global Pattern Recognition Algorithm

Joseph M. Stoffa

The background, development, performance assessment, and analysis of a novel pattern recognition algorithm that is applicable to any set of binary images are discussed. The efficacy of the algorithm when applied to the problem of fingerprint recognition is quantified. The conclusion was that the algorithm is relatively poor as a fingerprint identification algorithm, averaging an equal error rate of approximately 19% as calculated by the rules specified in the Year 2000 Fingerprint Verification Competition. The positive attributes of the algorithm were its ultra-fast matching times, orientation independence, lack of rejection events, relative insensitivity to resolution difference, and one-way transformations. The mechanism of algorithm operation as applied to fingerprints was investigated using integral geometry. This investigation showed that the algorithm was an indirect measure of ridge width, which explained the algorithm’s relatively poor performance. Another set of experiments suggests that the algorithm may be well-suited to other pattern recognition problems, specifically cloud and precipitation particle recognition and camouflage recognition. In summary, the research extends the field of pattern recognition by developing, assessing the performance, and determining the mechanism of operation of a novel pattern recognition algorithm that is applicable to any set of binary images.
Dedications

I dedicate this work to everyone who has helped and supported me through the innumerable hours, inevitable setbacks, and irritable moods that accompanied my research. My friends’ and family’s support over the past four years were as vital to this effort as the hours of research or the guidance provided by my committee members. It’s a cliché but it’s true, I couldn’t have done it without all of you.
Acknowledgements

First and foremost I want to acknowledge my research advisor, Dr. Alfred Stiller. Our discussions were always enjoyable and productive. Dr. Stiller’s tutelage has ensured that research yielded a complete and compelling story. I also want to thank Dr. Charles Jaffe, his experience and knowledge were vital during the early stages when the research plan was still fluid. Thanks also to Dr. Joseph Shaeiwitz, Dr. Charter Stinespring, and Dr. Richard Turton, their comments and perspective produced a stronger research effort.

I would also like to thank my fellow graduate and undergraduate students who provided fruitful interactions during research; John Deal, Nicholas Hansford, and Jeremy Hardinger.

Finally, I would like to acknowledge the National Institute of Justice and the West Virginia University C. Eugene Bennet Department of Chemistry for supporting this work.
# Contents

Title Page .......................................................................................................................................... i  
A61.3H6 ............................................................................................................................................ ii  
M6661.031 ........................................................................................................................................ iii  
A6636 412 631 ................................................................................................................................ iv  
Contents ........................................................................................................................................... v  
List of Figures ...................................................................................................................................... viii  

Chapter 1: Introduction ................................................................................................................... 1  
1.1: Problem Statement .................................................................................................................. 1  
1.2: Research Goals ...................................................................................................................... 2  
1.3: Document Structure .............................................................................................................. 3  

Chapter 2: Background .................................................................................................................... 6  
2.1: Pattern Recognition ................................................................................................................ 7  
2.2: Pattern Recognition Tools ................................................................................................... 8  
2.2.1: Signals ............................................................................................................................... 9  
2.2.2: Markov Chains ................................................................................................................ 9  
2.2.3: Monte Carlo Integration ................................................................................................. 11  
2.3: Fingerprints ............................................................................................................................ 13  
2.3.1: Early Observations on Fingerprints ............................................................................... 16  
2.3.2: Fingerprints as Biometrics ............................................................................................. 17  
2.3.3: Early Fingerprint Research ............................................................................................ 18  
2.3.4: Impetus for a Better Biometric ....................................................................................... 19  
2.3.5: The Henry System .......................................................................................................... 22  
2.3.6: Refining the Henry System ............................................................................................ 23  
2.3.7: Fingerprints Enter the Digital Age ............................................................................... 25  
2.4: Fingerprint Recognition Algorithms ..................................................................................... 26  
2.4.1: Minutiae Based Methods ............................................................................................... 26  
2.4.2: Correlation Based Methods .......................................................................................... 28  
2.4.3: Ridge Feature Based Methods ....................................................................................... 29  
2.4.4: MCMC Methods as a Means of Fingerprint Identification ......................................... 29  
2.5: The MCMC algorithm Discussed Herein ............................................................................. 31  
2.5.1: Random Walks and Random Steps .............................................................................. 31  
2.5.2: The Novel MCMC Algorithm Described by Deal ......................................................... 32  
2.5.3: Comparing Scale-Spectra ............................................................................................. 35  
2.6: Cloud and Precipitation Particles ......................................................................................... 37  
2.7: Camouflage ............................................................................................................................. 39
4.5.4: Optimizing the Number of Random Steps ............................................................ 115
4.6: Recreating FVC2000 .............................................................................................. 117
4.7: Comparison with Other FVC2000 Entrants ............................................................ 124
4.8: Application of Integral Geometry .......................................................................... 126
4.9: Other Areas of Applicability .................................................................................. 129
  4.9.1: Cloud and Precipitation Particle Recognition ..................................................... 129
  4.9.2: Camouflage Recognition .................................................................................... 137
4.10: Summary of Results ............................................................................................. 140
Chapter 5: Summary ..................................................................................................... 142
Chapter 6: Recommendations ......................................................................................... 145
Bibliography .................................................................................................................... 147
Appendix A: Fingerprint Preprocessing Function ......................................................... 151
Appendix B: Scale-Spectra Generating Function ............................................................ 157
Appendix C: Template Generating Function .................................................................. 168
Appendix D: Matching Score Calculation Function ...................................................... 170
List of Figures

Figure 2-1: A fingerprint image [18] ........................................................................................... 14
Figure 2-2: Global fingerprint patterns ........................................................................................ 14
Figure 2-3: Fingerprint ridge characteristics [15] ................................................................. 15
Figure 2-4: Anthropometric measurements [23] .................................................................. 21
Figure 2-5: Random step diagram ............................................................................................ 33
Figure 2-6: The scale-spectra, a matrix of matrices .................................................................. 34
Figure 2-7: Cloud and precipitation particle images from a 2D-S probe [45] ..................... 38
Figure 2-8: Camouflaged flounder ......................................................................................... 39
Figure 3-1: Six sub-modules designed and written during research ........................................ 47
Figure 3-2: Raw fingerprint image (each square is one pixel) ................................................... 50
Figure 3-3: Preprocessing function flowchart ......................................................................... 54
Figure 3-4: Scale-spectra generating function flowchart .......................................................... 57
Figure 3-5: Two needles on a plank floor ............................................................................... 79
Figure 4-1: Six sub-modules designed and written during research ........................................ 86
Figure 4-2: Time required for scale-spectra generation .............................................................. 88
Figure 4-3: Fingerprint preprocessing function output when N was too small ...................... 91
Figure 4-4: Fingerprint preprocessing function output when N was too large ..................... 92
Figure 4-5: Fingerprint preprocessing function output if \( \sigma_{\text{GV}} \) were too large .................. 93
Figure 4-6: Fingerprint preprocessing function output if \( \sigma_{\text{GV}} \) were too small .................. 93
Figure 4-7: Fingerprint preprocessing function output if thresholds were not used .............. 94
Figure 4-8: Optimized fingerprint preprocessing function output ......................................... 95
Figure 4-9: Estimated transition probabilities for \( F_{61,5} \) from FVC 2002 DB1A ..................... 96
Figure 4-10: Invariant scale-spectra for \( F_{61,5} \) of FVC2002 DB1A ........................................ 97
Figure 4-11: Determinant scale-spectra of \( F_{61,5} \) and \( F_{85,2} \) from FVC 2002 DB1A .......... 98
Figure 4-12: CDF of estimated transition probabilities ......................................................... 100
Figure 4-13: Transition probability estimate error as a function of random steps .............. 101
Figure 4-14: Determinant and trace scale-spectra ................................................................. 102
Figure 4-15: First and Second Eigen Value scale-spectra ..................................................... 103
Figure 4-16: Impostor and genuine matching scores for DB1A of FVC2002 ....................... 106
Figure 4-17: FNMR vs FMR of four difference metrics ......................................................... 108
Figure 4-18: EER as a function of the largest step-size .......................................................... 110
Figure 4-19: EER as a function of any single step-size .......................................................... 111
Figure 4-20: EER as a function of the weighting factor ......................................................... 113
Figure 4-21: EER as a function of step-size increment ......................................................... 115
Figure 4-22: EER as a function of random steps ................................................................. 116
Figure 4-23: FNMR vs FMR for FVC2000 DB1A................................................................. 118
Figure 4-24: Genuine and impostor matching scores for determinant metric .................. 119
Figure 4-25: Genuine and impostor matching scores for hybrid metric ......................... 120
Figure 4-26: Genuine and impostor matching scores for first Eigenvalue metric ............. 121
Figure 4-27: F*_{66_2} from DB1A (left-side) compared to F*_{54_2} from DB3A (right-side) ........ 123
Figure 4-28: F*_{14_5} from DB1A of FVC 2002 ................................................................. 127
Figure 4-29: Actual and predicted scale-spectra invariants ........................................... 128
Figure 4-30: From left to right, Particle 1, Particle 2, and Particle 3 .................................. 130
Figure 4-31: Determinant scale-spectra of three different particles .................................. 131
Figure 4-32: Particle 1 (left) and Particle 1a (right) ......................................................... 132
Figure 4-33: Determinant scale-spectra Particle 1 and Particle 1a .................................... 132
Figure 4-34: From left to right, Particle 1, Particle 1b, and Particle 1c [54] ......................... 134
Figure 4-35: Determinant scale-spectra of same particle at various resolutions ............. 134
Figure 4-36: From left to right, Particle 2, Particle 2a, and Particle 2b [54] ......................... 135
Figure 4-37: Determinant scale-spectra for rotated images ............................................ 136
Figure 4-38: A photograph of a flounder over a bed of gravel [55] .................................... 137
Figure 4-39: Four regions extracted from flounder photograph ....................................... 138
Figure 4-40: Determinant scale-spectra of four regions from the flounder photograph ....... 139
Chapter 1: Introduction

This chapter will state the problem addressed by this research, enumerate the research goals, and provide an overview of this document. The chapter will open by defining the problem statement and describing the philosophical approach to solving this problem. Next the research goals will be enumerated. This chapter will conclude by highlighting the important concepts, experiments, and results that were employed, developed, and obtained during research.

1.1: Problem Statement

The goal of research was to develop, optimize, assess, and understand a novel pattern recognition algorithm. Earlier research by Deal had shown that this algorithm could transform binary fingerprints into transition probabilities via a Monte Carlo Integration employing a random walk process [1]. Regarding this algorithm, Deal’s research did not encompass assessing its performance, understanding its mechanism of operation, or evaluating its application to other pattern recognition problems [1]. The research discussed herein formulates an experimental plan of attack to address the above issues and yields a more complete examination of the pattern recognition algorithm. A more in-depth discussion of the goals of this research follows.
1.2: Research Goals

The goals of research are to quantify the fingerprint recognition performance of the algorithm, increase the understanding of its operation, and explore its applicability to other pattern recognition problems. These goals are listed with more specificity below:

1) Quantify the performance of the pattern recognition algorithm as applied to fingerprints
   a. Optimize algorithm parameters for fingerprint identification

2) Increase understanding of the algorithm’s operation and abilities
   a. Justify tacit assumptions underlying algorithm operation
   b. Identify mechanism by which the algorithm operates in certain instances
      i. Apply concepts from integral geometry

3) Qualify the applicability of the algorithm to other pattern recognition problems
   a. Cloud and precipitation particle recognition and camouflage recognition

The first goal of research is to quantify the performance of the algorithm as a fingerprint recognition algorithm. This was accomplished by optimizing the algorithm for fingerprint identification, and using parameters and metrics that have already been developed for quantifying the performance of fingerprint recognition algorithms [2]. The result of this work is a comparison between the algorithm described herein and other fingerprint recognition algorithms.

The second goal of research was to increase understanding of the algorithm’s operation and abilities. There were a few assumptions underlying the operation of the algorithm
discussed herein, and these assumptions needed to be justified. For instance, it was assumed that an N-step random walk was equivalent to N instances of a random step (a random step is simply a random walk that is one step in length). It was also assumed that the Monte Carlo Integration would estimate transition probabilities with errors that were normally distributed. It was vital that these assumptions were examined and justified to ensure that the approach described herein was valid and that the transition probability measurements possessed a quantifiable error. Also included in the second goal of research was the application of integral geometry to analytically explain the behavior exhibited by the pattern recognition algorithm, specifically as it applied to fingerprint identification. Integral geometry was used to describe the character of the data produced by the algorithm.

The third and final goal of research was to qualify the applicability of the algorithm to other classes of pattern recognition problems. Therefore, it was decided to qualify the algorithm's performance on problems where minutiae recognition may not be critical. Two areas identified were cloud and precipitation particle recognition and camouflage recognition. The algorithm was applied to these problems in a preliminary manner to see if further research was warranted.

1.3: Document Structure

The next chapter provides the background information necessary to appreciate the research. This will include background information on the fields of study referenced during research, such as pattern recognition in general and fingerprint identification in particular. The background section will also define and provide descriptions of the mathematical tools
employed during research, including Markov Chains and Monte Carlo Integration. After the background, the next chapter in this document will discuss the experiments that were designed to accomplish the previously enumerated research goals.

Chapter three provides a detailed description of the data, equipment, computational tools, mathematics, and experiments that were utilized to research, develop, and quantify the performance of the algorithm described herein. The computational tools (hardware and software) employed during research and their relationship to the research will be explained. Following the discussion of hardware and software, the chapter discusses experiments that were designed to determine whether assumptions underlying the algorithm’s performance were justified. Experiments designed to quantify the efficacy of the algorithm as a fingerprint recognition technique will also be developed and detailed. Additionally, experiments that employ integral geometry to elucidate the mechanism of the algorithm’s operation will be defined. Chapter three will conclude by developing the sets of experiments that will test the algorithm’s applicability to other classes of pattern recognition problems.

Chapter four reports, analyzes, and interprets results from experiments introduced in the previous chapter. Chapter four opens with a discussion of the design, development, and implementation of the computer code that was written during research. Following this discussion, the chapter discusses the results of experiments described in the previous chapter. The experiments that demonstrated that earlier assumptions were justified will be discussed, as well as the experiments to optimize and quantify the performance of the algorithm as a fingerprint identification technique. The chapter will conclude with a discussion of the results
obtained via integral geometry and the applicability of the pattern recognition algorithm to other problems.

Chapters five and six present the results of the entire research effort and make recommendations for future research, respectively. Chapter five will cover the most important results and conclusions of the research effort. There are; the quantification or fingerprint recognition performance, the justification of underlying assumptions, the ability of integral geometry to explain algorithm behavior, and the applicability of the algorithm to other classes of pattern recognition problems. The sixth and final chapter presents recommendations for future research. These recommendations include suggestions for improving algorithm performance and suggestions for additional research into other pattern recognition problems where this algorithm may find applicability. Following the final chapter are several appendices that present computer code and other details that can be used to reproduce the results presented herein.
Chapter 2: Background

This chapter opens with a discussion of pattern recognition, defining the field’s history and applications. The chapter will then discuss some of the tools common to pattern recognition in general and this research in particular; Monte Carlo Integration, Markov Chains, and random walks. Discussion will then turn to a brief history of fingerprints, including a brief discussion of their purpose and some early observations on their uniqueness. This discussion will lead into a description of the early fingerprint identification systems. The environment in which fingerprint identification was developed will be described – which provide some insight into the impetus for a better biometric and the increased interest in fingerprint identification. This discussion will continue to the Henry system, which was the first widely adopted fingerprint identification system and served as the basis for future fingerprint identification systems, including many modern digital systems.

The advent of computers and computerized fingerprint recognition algorithms forever changed the field of fingerprint identification. Computers made new fingerprint recognition techniques possible, which has yielded many new fingerprint recognition algorithms over the past decades. These new computer-specific techniques will be discussed and their benefits and drawbacks relative to existing techniques will be discussed. This will lead into a discussion of Markov Chain Monte Carlo (MCMC) methods as fingerprint recognition algorithms. An entire subsection will summarize the MCMC algorithm described by Deal, which is the algorithm examined in this research effort [1]. Following this, the review of fingerprint recognition techniques will be complete.
This chapter will conclude with some background on other pattern recognition applications that were explored during research. A brief history of atmospheric science and the driving force behind cloud and precipitation particle recognition will be presented. Also, a brief section on camouflage is included.

2.1: Pattern Recognition

Pattern recognition has been defined as an attempt to solve the following problem: “Given some examples of complex signals and the correct decisions for them, make decisions automatically for a stream of future examples.” [3] Pattern recognition systems have been used to assist humans who search for patterns, replace human labor, or solve problems that humans are ill-suited to address. Pattern recognition emerged as a field of study in the 1950s. It included disciplines such as engineering, statistics, artificial intelligence, psychology, physiology, and computer science [4]. The field of pattern recognition facilitated communication between psychologists and physiologists (who appreciated human brain function) and engineers (who understood how to build systems) [3]. This intermingling of ideas led to concepts such as the perceptron (in the 1950s) and the neural network (in the 1980s) [3]. The concepts of the perceptron and neural network quickly departed from their biological roots and fell under the domain of statisticians, mathematicians, and engineers who were interested in developing practicable pattern recognition systems [3].

Pattern recognition systems usually consist of a sensor to capture a signal, a feature extraction mechanism to segregate the useful part of the signal, and a system that describes or classifies the signal. Early researchers attempted to apply this knowledge to the “classical”
pattern recognition problems, such as medical diagnosis and character recognition [4]. In these applications, pattern recognition systems were designed to mimic human behavior. In some later applications, such as credit scoring, pattern recognition systems were used because they were faster, less expensive, or superior to human beings performing the same task [3]. Eventually, communication systems such as bar codes that were difficult for humans to read were developed and designed specifically because they were easily read by electronic systems [3]. Eventually, scientists and engineers would develop pattern recognition systems for the applications discussed in this document.

Fingerprint recognition algorithms and cloud and precipitation particle recognition algorithms are two examples of pattern recognition systems that displace human effort. Certainly humans are capable of comparing fingerprints or classifying cloud and precipitation particles. However, pattern recognition algorithms are needed because the number of fingerprint comparisons or cloud particle classifications far exceeds the available human capital. Like other pattern recognition systems, these systems describe or classify the patterns of interest. Before discussing the details of these systems, it will be necessary to define some terms that are associated with these pattern recognition systems.

2.2: Pattern Recognition Tools

The following sub-sections will describe signals, Markov Chains and Monte Carlo Integration, three concepts that were invoked during research. Signals are the input to which Markov Chains and Monte Carlo Integration are applied. Markov Chains have existed for more than a century, and have found application to pattern recognition problems such as fingerprint
classification, fingerprint matching, voice recognition, face recognition, and more [5,6,7,8,9].

Markov Chains are often combined with Monte Carlo Integration, and the resulting combination is known as Markov Chain Monte Carlo (abbreviated MCMC) methods. The next three subsections will provide background information on signals, Markov Chains, and Monte Carlo Integration, respectively.

2.2.1: Signals

A signal is some quantity that varies in one or more dimension. The value of the quantity at any point in the dimension is known as its state. Extending the definition of a signal to binary images, the dimensions were the x-dimension length (width) and y-dimension length (height), and the quantity was the color (black or white). Specifying an x-coordinate and y-coordinate in an image identified a unique point in the two-dimensional image, this point was called a pixel. This pixel had a value of 0 or 255, black or white, respectively. As will be discussed in a later subsections, it was possible to transform these signals (binary images) into Markov Chains.

2.2.2: Markov Chains

Markov Chains are processes that possess the Markov property. The Markov property stipulates that the future state of a system depends solely on the present state. An everyday example of Markov Chains is a game of baseball. Envision a runner on second base. The runner has four possible future states; still on second base, advanced to third base, advanced to home, or out. The probabilities that the base runner will transition to one of these four states are known as the transition probabilities. The base runner’s future state is independent of his or
her past state; it does not matter if the base runner advanced to second base with a double, a single and a stolen base, or a single and a walk. Likewise, when traversing along a dimension through some signal, if the future state depended solely on the present state, the signal had the Markov Property.

Markov chains were developed by and were named after Andrei Andreevich Markov [10]. Markov first wrote about chain dependence in 1906 [11]. As previously stated, the Markov property dictated that the future state depended solely on the present state. Thus, the future state was completely independent of past states. This is stated formally as Equation 2-1.

\[
P(X_{n+1} = x|X_n = x_n, \ldots, X_1 = x_1) = P(X_{n+1} = x|X_n = x_n)
\]

Equation 2-1

Where:

- \(X_{n+1}\) is the future state of the variable \(X\)
- \(x\) is the value of the variable

Transition from one state to the next was probabilistic, and completely independent of prior states. The probability of one state proceeding to another state was known as a transition probability. Equation 2-2 represents the transition probability of a Markov Process in state \(i\) proceeding to state \(j\).

\[
p_{ij} = P(X_{n+1} = x_j|X_n = x_i)
\]

Equation 2-2
Transition probabilities were often quantified by examining sequences of data from a signal and noting the frequency with which transitions occurred. For this technique to work, the process under consideration must have been ergodic. Ergodic processes were “a poll-taker’s dream, because any reasonably large sample tends to be representative of the sequence as a whole.”[12]

Markov chains have found use in the fields of physics, electrical engineering, computer science, mathematics, statistics, game theory, and more. Particularly applicable to the current work was the application of Markov chains to communication signals [12]. All digital communication and information can be considered a signal, so the insight that signals can be modeled as Markov chains had profound consequences. For fingerprint identification, this raised the possibility of transforming fingerprints into their Markov chain representations, and comparing these representations to compare fingerprints. The estimation of transition probabilities from the examination of an arbitrarily long sequence of data was a form of Monte Carlo Integration. Markov Chains have often been combined with Monte Carlo Integration, which is discussed next.

2.2.3: Monte Carlo Integration

Monte Carlo methods have been described as “the rebirth of statistical sampling” [13]. Monte Carlo refers to the concept that meaningful observations can be generated by repeated random sampling of a specified distribution [14]. Monte Carlo methods are often utilized when integrating a distribution is unrealistic, due to either mathematical difficulties or time constraints. The error associated with a Monte Carlo generated observation decreases as the
reciprocal of the square root of the sample size. Thus, if it is desired to reduce the error associated with an observation by a factor of ten, the sample size must be increased one hundred fold. Because the sample size required for confidence in an observation may be quite large, Monte Carlo methods are almost exclusively associated with computers.

With the invention of the electrical computer in 1945, it became possible for mathematicians to solve problems experimentally that were difficult to solve analytically [13]. Monte Carlo methods achieved prominence around the middle of the twentieth century. The year 1945 saw the invention of both the atomic bomb and the electronic computer. Physicists who were interested in fusion weapons realized they could use the electronic computer’s immense calculating power to track the lives of neutrons, and thus understand the conditions necessary for fusion [13]. Physicists were able to use existing differential equation methods to achieve an exact solution for the life of neutrons. However, the neutron’s velocity spectrum was very difficult to handle mathematically, making traditional differential equation methods non-ideal [13]. Instead, physicists used computer programs to simulate the life of a single neutron. Random numbers from a special distribution determined the life of this single neutron. The results of the neutron’s life were noted. This result was combined with the results from thousands or millions of other simulated neutron lives. By observing many single neutron life spans (samples), an observation about the average behavior of a neutron was made, and the problem was solved.

Monte Carlo Integration has often been used to estimate the transition probabilities of Markov Chains. The combination between the two has been referred to as Markov Chain Monte Carlo Methods, or MCMC Methods. These methods have been applied to multiple
pattern recognition problems, including fingerprint recognition, as will be discussed later. This concludes the discussion of pattern recognition tools that were employed during this research. The next section will summarize the history of fingerprints and fingerprint identification.

2.3: Fingerprints

It has been hypothesized that humankind’s sub-primate ancestors evolved friction ridges for three basic purposes [15]:

1.) Provision of a gripping surface
2.) Tactile Facility
3.) Exudation of perspiration

Regarding the second purpose, researchers have suggested that friction ridges amplify vibrations, transmitting fine-texture data to nerve cells [16]. Regarding the third purpose, the exudation of perspiration is responsible for fingerprints, which are produced when the perspiration released from sweat pores leaves an impression of the friction ridges [15].

The term fingerprint has a formal and informal definition. According to the Scientific Working Group on Friction Ridge Analysis, Study and Technology (SWGFAST), a fingerprint was defined as “an impression of the friction ridges of all or any part of the finger.”[17] Informally, the term fingerprint may refer to either the impression produced by a finger’s surface, or, the finger’s surface itself. During this dissertation, the formal definition was used. Note the distinction between a finger’s surface and a fingerprint – one person’s finger may produce many fingerprints. An example fingerprint is presented as Figure 2-1 [18].
The human eye can quickly recognize several features in a fingerprint. The black and white colors represent where a finger left (or failed to leave) an impression. The long black and white lines are known as ridges and valleys, respectively. Often, these ridges and valleys form large chiral spirals – global patterns known as whorls. The common global patterns have been classified as; arch, tent, loop, and whorl, as presented below in Figure 2-2 [15].
In addition to global patterns, fingerprints contain a wealth of information in the form or smaller scale features dispersed throughout the fingerprint; these are known as minutiae. For example, sometimes a ridge (or valley) may split into two, abruptly end, or exist alone. These minutiae are known as bifurcations, terminations, and islands, respectively. A figure illustrating several types of minutiae is presented below as Figure 2-3 [15].

![Fingerprint ridge characteristics](image)

**Figure 2-3: Fingerprint ridge characteristics [15]**

As will be discussed in more detail later, most fingerprint identification algorithms group fingerprints into broad categories based on their global patterns and then match fingerprints based on minutiae. Minutiae can include other features such as sweat pores and scars. Fingerprint features are often categorized as belonging to one of three levels, with higher levels denoting finer features. Level one detail encompasses general morphological information including ridges and valleys, level two details includes minutiae such as bifurcations and islands,
and level three detail includes finer features such as sweat pores [17]. Now that the term fingerprint has been defined, discussion will turn to early observations on fingerprints.

2.3.1: Early Observations on Fingerprints

Neolithic stone carvings from as early as 2,000 BC suggest that ancient people were aware and intrigued by the intricate patterns present on their fingertips [15]. The earliest recorded description of local ridge features on the surfaces of hands and feet was given by plant morphologist Nehemiah Grew [19]. In his Philosophical transactions (1684), Grew writes:

“For if any one will but take the pains, with an indifferent Glass, to survey the Palm of his Hand very well washed with a Ball; he may perceive (besides those great Lines to which some men have given Names, and those of middle size call’d the Grain of the skin) innumerable little Ridges, of equal bigness and distance, and everywhere running parallel with one another.” [19]

Grew described fingerprints in great detail, but did not propose their use as a unique identifier. The first documented evidence of people using fingerprints as a type of signature is from 1690 [15].

The residents of Lodonberry (Northern Ireland) were under siege in 1690, until finally they were relieved by William of Orange. Several citizens drafted a document requesting compensation for damages incurred during the siege. Each citizen who signed the document included an ink fingerprint next to their signature – obviously these citizens considered their
fingerprints unique biometric identifiers. Although the uniqueness of fingerprints was assumed by Londonberry residents, there would not be serious research into the subject until 200 years after the siege of Londonberry.

2.3.2: Fingerprints as Biometrics

Fingerprints have two characteristics that have made them well suited to serve as biometric identifiers. Firstly, it has been demonstrated that fingerprints are nearly immutable with the passage of time. Henry Faulds, the first person in the modern era whom suggested that fingerprints could be used to identify people, demonstrated that if fingerprints were removed by shaving off the upper layers of the epidermis, they will grow back as they were before [20]. Additionally, fingerprints develop in the womb and persist until death. Thus, fingerprints have been demonstrated constant throughout a person’s life and nearly immutable; this made them useful as biometrics provided they were unique to every individual. This lead to the second surprising attribute of fingerprints: “no one has ever been found who has a sequence of ridge detail on the hands and feet that is identical to the ridge detail of any other person.”[15] Note that the uniqueness of fingerprints has been empirical; there is no proof that two people will never have identical fingerprints.

Before proceeding, it is important to note that fingerprint identification by humans is a skill and not a science. This was highlighted in 2002, when a judge in Pennsylvania ruled that expert witnesses could not testify whether fingerprints from a crime scene belonged to the defendant because fingerprint identification failed to meet three of the four criteria that the Supreme Court used to define science (Daubert v. Merrell Dow Pharmaceuticals) [21]. The
judge found fingerprint identification “hadn’t been scientifically tested, wasn’t subject to scientific peer review, and didn’t possess a known rate of error.”[21] Furthermore, “Fingerprinting was ‘generally accepted’ among forensic scientist, he found, but that did not establish its reliability.”[21] The judge was asked to reconsider his decision, and eventually reversed his decision, concluding that “fingerprinting is a form of technical expertise akin to accident reconstruction or art appraisal, he said, so it need not meet the scientific peer-review requirement.”[21] Even though the judge reversed his decision, it highlighted the point that despite over a hundred years of research and development, fingerprints are not a perfect biometric.

However, due to their nearly immutable character and individuality, fingerprints have gained wide acceptance as biometrics. This section will open with a discussion of early observations on fingerprints, and the development of fingerprints as a means of identification (including fingerprint identification’s role in replacing more antiquated identification techniques).

2.3.3: Early Fingerprint Research

The pioneers of modern fingerprint research were William Herschel, Henry Faulds, Edward Henry, and Francis Galton [22]. In 1858, William Herschel began experimenting with fingerprint identification in India [22]. Herschel, like the citizens of Londonberry, intended to use fingerprints as a signature that couldn’t be forged [22]. Henry Faulds is credited as the first person to suggest that fingerprints could be used to identify criminals, in a paper he submitted to Nature in 1880 [20]. Note the difference between what Herschel and Faulds suggested.
Herschel suggested that fingerprints could be used as a signature; if there were two fingerprints an examiner could determine if they were produced by the same finger. Faulds suggested that fingerprints could be used as identification; if an examiner were given a fingerprint, he or she could determine if that person’s fingerprint already existed in a database of fingerprints. Using fingerprints for identification was a much more difficult task than using fingerprints for identity verification, the former requiring a reliable method of sorting fingerprints (or a cold search, which was unrealistic for large databases prior to computers). Because Faulds never developed a practical classification system, his work was largely ignored, which cause him a great deal of grief during his lifetime [22]. At one point Faulds wrote a letter to Charles Darwin, hoping that Darwin would be able to assist in the development of fingerprint recognition. Darwin found the letter interesting enough to forward to his cousin, Francis Galton [22]. Galton possessed more prestige and influence than Faulds, which allowed him to reach a larger audience when publishing the results of his research [22]. Galton's work interested the Inspector General of Police in Bengal India, Edward Henry [22]. Edward Henry was looking for an identification system that could overcome the shortcomings of anthropometry, an identification technique that was previously developed to identify recidivists [22]. Before proceeding into a discussion of these biometric techniques, the impetus behind the identification of recidivists will be discussed in the following subsection.

2.3.4: Impetus for a Better Biometric

The development of modern biometric methods was spurred by England’s Habitual Criminals Bill, passed in 1867 [23]. The Habitual Criminal Bill specified longer jail sentences for
repeat offenders. Historically, identifying repeat offenders was straightforward; people were well known in their local communities. However, with the industrial revolution fueling mass migration and urban growth, citizens were more anonymous to their neighbors and to the authorities. Criminals could avoid being labeled repeat offenders simply by using aliases when arrested. It became obvious that reliable methods to identify recidivists were necessary.

Law enforcement experimented with various methods to identify recidivists, including branding. The most popular of the more benevolent systems was anthropometry; the identification of individuals using measurements of the human body. This system was designed by Alphonse Bertillon, and relied on the measurement of features that did not change with time (assuming you were measuring a fully grown adult). Bertillon’s system relied on eleven measurements:

1. Height
2. Stretch: Length of body from left shoulder to right middle finger when arm is raised
3. Bust: Length of torso from head to seat, taken when seated
4. Length of head: Crown to forehead
5. Width of head: Temple to temple
6. Length of right ear
7. Length of left foot
8. Length of left middle finger
9. Length of left cubit: Elbow to tip of middle finger
10. Width of cheeks
11. Length of left little finger

These measurements were written on cards and sorted in a filing cabinet. Criminals were then identified by searching through the collection of cards until a small set of cards are obtained, with the final identification occurring by a photograph that was attached to each anthropometric card. The process of anthropometry is illustrated in Figure 2-4.

Figure 2-4: Anthropometric measurements [23]

France adopted Bertillon’s system in 1883, and initially experienced great success [23]. However, there were drawbacks to Bertillon’s system. Firstly, the anthropometric measurements required fine (expensive) tools [23]. Secondly, the system required highly
trained specialists to record the measurements [23]. Finally, measurements were extremely
time consuming, and errors that crept in were hard to correct [23]. It was clear that a better
system was necessary, and fingerprints were recognized as having potential for use as
biometrics.

In 1893, a British committee, chaired by Charles Troup, was formed to determine
whether Britain should adopt anthropometry or fingerprinting (using Galton’s system) as a
means to identify recidivists [24]. This committee concluded that fingerprints were promising,
but the lack of an effective fingerprint classification scheme would make the system
unworkable as a means of identification [24]. Britain therefore decided to adopt anthropometry
[24]. It was obvious that an improved fingerprint classification system was necessary before
fingerprints could receive widespread application as a biometric.

2.3.5: The Henry System

In the British colony of India, the shortcomings of anthropometry were more
pronounced. The British colony lacked the skilled labor and fine instruments necessary to
record meaningful anthropometry measurements [23]. Fingerprints were, therefore, more
attractive as a means of identification in India because they could be collected by unskilled
laborers and examined at a centralized location using skilled examiners. It was in this context
that Edward Henry, and his two assistants Azizul Haque and Chandra Bose, attacked the
problem of developing a workable fingerprint classification system. By noting the general ridge
patterns and number of ridges which occur on all ten fingerprints, Henry and his assistants were
able to classify a large set of fingerprint cards into much smaller sub-sets [23]. Henry estimated
that the classification scheme could reliably handle up to 100,000 sets of fingerprints [23].

News of Henry’s system spread, and in 1897 the colonial government in India formed a committee to judge the merits of Henry’s new fingerprint identification system [25].

The committee determined that the Henry System had addressed the shortcomings of Galton’s system, and that fingerprinting should be adopted as a means of identification in India [25]. The Henry system of fingerprint identification had successful outcomes as an identification tool in India. In one famous case in 1898, a fingerprint was used to place a suspect at the scene of a murder, netting a conviction for burglary [26]. In 1899, the Indian Evidence Act was passed by the colonial legislature – the earliest recorded endorsement of fingerprints as evidence [25]. These successes led to the formation of another British committee.

The Belper committee was formed to determine if Britain should accept fingerprint identification on a trial basis [27]. The Belper committee eventually concluded that Henry’s system had addressed the shortcomings of Galton’s system, and that fingerprinting should be used on a trial basis in Britain [27]. The Henry system was trialed in Scotland Yard with great success [23]. During 1902 (the first full year using the Henry system) Scotland Yard identified 1722 recidivists, whereas in 1900 (the last full year of anthropometry) they identified only 467 recidivists [23]. After the unquestionable success at Scotland Yard, the Henry System would soon spread across the World [28].

2.3.6: Refining the Henry System

As the mobility of criminals increased, the need for information sharing among law enforcement became obvious. However, larger law enforcement agencies were reluctant to
share fingerprints because the fingerprints represented a source of power. Therefore, national governments who attempted to centralize all sets of fingerprints met resistance, especially in America given the philosophy that the Federal Government could not be trusted in matters of law enforcement [23]. In 1922, Hakon Jorgensen took the lead and established an international bureau for identification in Copenhagen. This pan-European identification bureau would eventually become Interpol [29]. Hakon Jorgensen was also responsible for developing a system that represented all ten fingerprints as a code (up to 50 characters in length) that could be transmitted via telegraph [29]. While these systems had some successes, they were still limited by the drawbacks inherent to the Henry system.

There were two major drawbacks associated with the Henry System. Firstly, the Henry System was not well suited to identify an individual given a single print [23]. Thus, all ten fingerprints were needed (rare at a crime scene), a suspect was needed, or a large portion of the database had to be searched. Secondly, as fingerprint cards began to number in the hundreds of thousands, the Henry System became cumbersome [23]. To alleviate these problems, modifications to the Henry system and alternative systems were proposed [23]. By 1945 there were well over 30 fingerprint identification schemes worldwide, all of them incompatible [23]. Many of these modified systems attempted to develop a classification scheme for single prints, allowing for a single latent print at a crime scene to be matched to a suspect already on file.

The next forty years were marked by the development of new systems, attempts to ease the burden of searching through fingerprints, or new systems to classify single prints. Eventually fingerprints were represented as punch cards, facilitating a means to search quickly
through the more crowded sections of the Henry System [23]. The next leap in fingerprint technology would occur in 1979, when the FBI began using an Automated Fingerprint Identification System (AFIS). [23]. Electronic fingerprint recognition represented a paradigm shift in fingerprint identification technology, as will be discussed in the following subsection.

2.3.7: Fingerprints Enter the Digital Age

Modern Automated Fingerprint Identification Systems (AFIS) were developed during the 1970s [23]. By 1980, the FBI had scanned 14.3 million fingerprint cards – all of their cards associated with people born after 1928 [23]. AFIS provided, for the first time, a realistic means to “cold search” a fingerprint database. A cold search occurs when a single fingerprint is compared to all of the fingerprints in a database. With human examiners, this would have been a prohibitively time consuming task. With AFIS, cold searches were routine. Additionally, AFIS provided a method for quickly determining if someone in custody was already on file. Thus, it was easily determined if someone was using an alias to avoid their criminal past, which was the reason fingerprint technology was originally adopted in England.

In 1999, the FBI’s fingerprint database was combined with criminal histories, physical characteristics (height, weight, eye color, hair color, etc...), mug shots, aliases, and photos of tattoos [30]. This combined database is known as the Integrated Automated Fingerprint Identification System (IAFIS). The IAFIS is the largest biometric database in the world, housing over 66 million fingerprints and criminal histories, and over 25 million civil prints [30]. The IAFIS system processes approximately 162,000 ten-print submissions per day, and submitted fingerprints are processed in an average of 10 minutes [30]. It is important to note that an IAFIS
match is not admissible as evidence of guilt in a United States court of law; the final
determination as to whether two fingerprints match is made by a trained human examiner.

IAFIS is the present day state of the art in fingerprint identification systems. IAFIS is only
one of many fingerprint recognition systems, albeit the most highly developed one. These
systems all have an algorithm underlying their operation, whether it is the century old Henry
system or a newer computer-specific algorithm. The next few subsections will discuss these
different types of fingerprint recognition algorithms and their pros and cons.

2.4: Fingerprint Recognition Algorithms

There are many fingerprint matching algorithms, which fall into three major categories:
minutiae based methods, correlation based methods, and ridge feature based methods [18].
Each class of methods possesses its own advantages and disadvantages. Most methods do not
use raw fingerprint images. The raw image (called the input) is usually fed through some pre-
processing algorithm, producing a modified image (called the template) that can then be fed to
the matching algorithm [18]. The class of methods that most closely resemble how a human
fingerprint examiner matches fingerprints are the minutiae based methods. These will be
discussed first.

2.4.1: Minutiae Based Methods

Early fingerprint identification algorithms were minutiae based methods developed for
the FBI’s AFIS [31]. Early algorithms divided fingerprints according to a grid, and noted which
grid sections contained minutiae [32]. Each grid location was given a value based on its
minutiae type [32]. Grid values from different fingerprints could then be compared for matching minutiae. [32]. The benefit of this method was that it closely resembles the process followed by a human fingerprint examiner. The disadvantage to this algorithm was that fingerprints must be correctly aligned (offset and rotation) according to a strict set of rules [32].

The inherent variability and elasticity of fingerprints makes alignment a difficult task [18]. Not all fingerprints share the same global patterns or common minutiae, so it is difficult to develop a set of rules that will always align a fingerprint in the desired way. Additionally, fingers are elastic and can deform (stretch and twist) when leaving fingerprints on surfaces [18]. This problem is exacerbated by the poor quality of latent fingerprints often found at crime scenes. The ultimate goal is absolute pre-alignment. Absolute pre-alignment would reduce fingerprint identification to a simple task of comparing coordinates [18]. However, no absolute pre-alignment algorithm has been demonstrated, so all fingerprint identification algorithms must tolerate some mis-alignment [18].

Some researchers have avoided alignment issues by cataloging minutiae with respect to their local friction ridges [33]. The disadvantage to this “intrinsic coordinate system” approach is difficulty in application to low-quality latent prints. The shortcomings of minutiae-based systems can be avoided entirely by developing algorithms that do not directly utilize minutiae, These are known as correlation based methods, which will be discussed in the following subsection.
2.4.2: Correlation Based Methods

Correlation based methods do not rely on minutiae. Correlation based methods operate by overlaying two fingerprint images and noting the differences between them. Small differences suggest that the fingerprints may have originated from the same finger. The largest disadvantages associated with correlation based techniques involve fingerprint quality and computational costs [18]. As with other fingerprint matching algorithms, the variable pressure and skin condition with which a finger applies fingerprints can cause distortion and noise, which can suggest relatively large differences between fingerprints produced by the same finger [18]. Furthermore, by their nature correlation based methods must examine several permutations of two fingerprints overlying one another. This can lead to immense computational costs [18].

Researchers have made progress in addressing some of the shortcomings of correlation based methods. The distortion problem has been partially addressed by noting the correlation between regions of a fingerprint rather than the entire fingerprint, which decreases the contribution of distortion [34]. Some researchers have restricted the operating space (and decreased the computational load) of correlation based algorithms by examining only the regions surrounding various types of minutiae [35]. Researchers have also reduced the computational load of correlation-based techniques by applying transformations (usually Fourier or Fourier-Mellin) to produce orientation and rotation independent fingerprint transitions [36,37]. While there are drawbacks to correlation based methods, it should be noted that they are relatively immature compared to minutiae based methods.
2.4.3: Ridge Feature Based Methods

The last major category of fingerprint identification methods discussed is ridge feature based methods. These methods were developed as an alternative to minutiae based methods because minutiae extraction can be difficult and time consuming [18]. Some of these methods use alternative fingerprint feature such as the silhouette of a fingerprint, geometrical features of the friction ridge, sweat pores, and textures and fractal features [18, 38, 39, 40]. Eventually, ridge feature methods may be used to complement more traditional methods or may be used in embedded systems. The limited processing power of ultra-portable devices may be more able to cope with ridge based fingerprint identification algorithms than minutiae based fingerprint identification algorithms [18]. This concludes the discussion of the different classes of fingerprint recognition algorithms. The following section will discuss fingerprint identification methods that employ MCMC algorithms.

2.4.4: MCMC Methods as a Means of Fingerprint Identification

There are various techniques to utilize Markov chains for the purpose of fingerprint identification [6, 9, 1]. Guo and others showed that fingerprint identification using a Hidden Markov Model (HMM) is possible [6]. A HMM is a system where the transition states are hidden from the observer, but the process output is modeled through parameters that assume an underlying Markov process. Guo segmented a fingerprint into a series of discrete squares, noted the ridge orientation in each square, and recorded the transition in ridge angles from one discrete square to another [6]. Using a training set of fingerprints, Guo fitted the HMM to the fingerprints [6]. Guo applied his algorithm to the first database from the 2002 Fingerprint
Verification Competition (FVC2002-DB1). When applied to FVC2002-DB1, Guo’s algorithm demonstrated an EER of 7.1% [6]. Because Guo measured ridge angles relative to some point, pre-alignment of fingerprints was necessary [6].

Bouchaffra and others demonstrated a technique to model fingerprint features (and face features) using a structural hidden Markov model (SHMM) [9]. The modeled features were extracted from fingerprints using discrete wavelet transformations or ridgelet transformations [9]. The goal of their Bouchaffra’s work was stated as the fusion of face and fingerprint recognition. Unfortunately, Bouchaffra did not quantify the performance of their algorithm using a standard database such as the FVC databases, so it was impossible to judge the effectiveness of their proposed approach [9].

Deal showed that a random step could be applied to a fingerprint many times (a form of Monte Carlo Integration) to model the transition between two points separated by a fixed distance as a Markov Chain [1]. Deal was examining binary fingerprints, so in the context of Deal’s research there were only two states under examination, black states and white states [1]. These black and white states represent the black and white pixels that compose binary fingerprint images [1]. Deal hypothesized that the probability of transitioning between these two states will vary for different fingerprints, which raised the possibility that the transition state probabilities can be used to recognize fingerprints [1]. However, Deal failed to quantify the performance of the algorithm he described [1]. Because this research focuses on the MCMC method described by Deal, the following section will provide a more detailed description of this algorithm.
2.5: The MCMC algorithm Discussed Herein

As was mentioned previously, this research represented a further development to the novel MCMC algorithm described by Deal [1]. Previous research by Deal has shown that a novel MCMC method based on a random step was capable of transforming binary fingerprint images into a set of transition probabilities. This advancement suggested that the novel MCMC method may be amenable to fingerprint recognition. The following sub-sections will detail the principles of the algorithm’s operation. The section will open by defining a random step and describing its benefits. The following subsection will present a description of the algorithm’s operation and describe how it can represent binary images as a set of transition probabilities. The final subsection will discuss comparisons – the process of comparing one set of transition probabilities to another (and thus comparing the parent fingerprints). Although Deal did not address comparisons, they are included in this section as they are a prerequisite to one of the goals of this research; quantifying the fingerprint recognition performance of the algorithm.

2.5.1: Random Walks and Random Steps

The term random walk was introduced by Karl Pearson in 1905, and referred to the familiar model for random processes [41]. Random walks can occur in spaces of any dimension. This research was restricted to two-dimensional space, because the binary images used during research were represented as two-dimensional signals. Note that random walks are by definition serial processes, the coordinates of point ‘n’ must be known before point ‘n+1’ can be calculated. This was a drawback from a computing standpoint, because the power of parallel
computing could not be leveraged on an inherently serial process. A solution to this drawback was developed in the form of a random step.

A random step is defined as a random walk that was only one step in length. It is hypothesized that transition probabilities estimated via \( N \) iterations of a random step are equivalent to transition probabilities estimated via an \( n \)-step random walk. The benefit of this approach is that a serial process is transformed into a parallel process, which is amenable to parallel processing.

### 2.5.2: The Novel MCMC Algorithm Described by Deal

The digital images examined by Deal were binary (black and white) fingerprints overlaid with a green mask (representing areas that contained extraneous information) \([1]\). The heart of the pattern recognition algorithm described by Deal was a method to transform binary images into a series of transition probabilities via a Monte Carlo Integration consisting of many random steps \([1]\). The random step began with the selection of two random numbers. These two random numbers were picked from a uniform distribution. These two random numbers represented an \( x \)-coordinate and a \( y \)-coordinate, which defined a point within the image, \( P_1 \). Because the two random numbers were picked from a uniform distribution, all of the points within the rectangular image have an equal probability of being selected. There were three possibilities for \( P_1 \); it was black, white, or green (the area which outlined the binary images, if necessary). If \( P_1 \) was green it was not part of the binary image, and two new random numbers were picked from a uniform distribution until a valid point was selected. Once a valid point was selected, another random number between 0 and \( 2\pi \), called theta, was picked from a uniform
distribution. Given an initial point $P_1$, an angle theta, and a step-size; a new point was defined. This new point was designated $P_2$. As with $P_1$, $P_2$ could be black, white, or green. Unlike $P_1$, $P_2$ could fall outside of the bounds of the binary image as well. If $P_2$ was within the green mask or out of bounds, it was not valid, and the steps above were repeated. Once two valid points were chosen, the random step was complete. A graphical illustration of this process (as applied to a fingerprint) is presented as Figure 2-5.

![Figure 2-5: Random step diagram](image-url)
For binary images, every point in the x-y plane was either black or white. Thus, when the transition from $P_1$ to $P_2$ was examined, it was seen that there were only four possibilities. The transition from $P_1$ to $P_2$ can go from: black to black, black to white, white to black, or white to white. The transition was noted, and thus one random step was complete. The above procedure was repeated many times to accurately estimate these four transition probabilities. These four transition probabilities were denoted as $p_{BB}$, $p_{BW}$, $p_{WB}$, and $p_{WW}$, as illustrated by the following equations.

\[
\begin{align*}
p_{BB} &= P(X_{n+1} = B|X_n = B) \\
p_{BW} &= P(X_{n+1} = W|X_n = B) \\
p_{WB} &= P(X_{n+1} = B|X_n = W) \\
p_{WW} &= P(X_{n+1} = W|X_n = W)
\end{align*}
\]

The above Markov Chain transition probabilities were estimated at a given step size, $S$. If the step size was changed, the transition probabilities may change as well. The evolution of the transition probabilities as a function of step-size was called the scale-spectrum. The scale-spectrum can be visualized as a matrix of matrices, as illustrated in Figure 2-6.

**Figure 2-6: The scale-spectra, a matrix of matrices**

Each sub-matrix contains four transition probabilities at a given step-size
The subscripts on the sub-matrices represent the step-sizes utilized by the random steps when the transition probabilities were estimated. After the transition probabilities were obtained for all scales of interest, the binary image was been transformed into a scale-spectrum – a matrix of matrices that represented the image. It follows that the parent images could be compared by comparing their scale-spectra, although this was never implemented by Deal [1].

The pattern recognition algorithm described above possessed some distinct advantages. Because images were represented as a small set of numbers, matching times are likely to be very fast. Additionally, the random step approach was amenable to parallel processing, allowing increased computational power to be harnessed. Orientation independence (due to the selection of a random angle) was also attractive, as it can decrease preprocessing requirements. Finally, it was straightforward to transform an image into its corresponding scale-spectra, but it was not apparent that the inverse operation was possible. This is an advantage where there are security or privacy concerns with the dissemination of the original image. The concludes the discussion of the algorithm described by Deal, the next sub-section will cover how scale-spectra that results from fingerprints may be compared.

2.5.3: Comparing Scale Spectra

When the difference between transition probabilities from two different images is calculated, the result is a unitless scalar. It was hypothesized that relatively small differences between transition probabilities reflected relatively small differences between their parent images. Thus, by comparing the transition probability differences to some arbitrarily set threshold, the significance of the difference was judged. If the differences between transition
probabilities were less than or equal to the threshold, then the images were indistinguishable and the two images were assumed to be the same pattern. If the differences between transition probabilities were greater than the threshold, then the images were significantly different and the two images were assumed to have originated from different patterns. This is a common approach to matching biometric patterns, and there are four possible outcomes when comparing two patterns in this manner [18]:

1. True Positive. Two images from the same pattern were judged indistinguishable. In biometric parlance, the probability of a true positive was called the match rate.
2. True Negative. Two images from different patterns were judged unique. In biometric parlance, the probability of a true negative was called the non-match rate.
3. False positive. Two images from different patterns were judged indistinguishable. In biometric parlance, the probability of a false positive was called the false match rate (FMR).
4. False negative. Two images from the same pattern were judged unique. In biometric parlance, the probability of a false negative was called the false non-match rate (FNMR).

The FMR and FNMR depended heavily upon the threshold. If the threshold was extremely small, then all images were judged unique. In this case, the FMR was zero and the FNMR was near one. Alternatively, if the threshold was extremely large, then all images were judged indistinguishable. In this case, the FMR was approximately one, and the FNMR was zero. As the threshold was varied from a small number to a large number, the FMR and FNMR cross paths.
The value where the FMR equaled the FNMR was known as the equal error rate (EER) [18]. The EER was one of the most important metrics for quantifying the efficacy of a pattern recognition algorithm. The EER of a perfect pattern recognition algorithm would be zero. A pattern recognition algorithm that randomly guesses if two images were unique or indistinguishable would have an EER near 50%. This concludes the discussion of the fingerprints and fingerprint recognition algorithms. The remaining two sections in this chapter will discuss cloud and precipitation particles and camouflage.

2.6: Cloud and Precipitation Particles

The World Meteorological Organization defined a cloud as “an aggregate of minute particles of water or ice, or of both, in free air.” [42] Precipitation particles are generally larger (and fall faster) than cloud particles, although there is no specific scientific demarcation between cloud particles and precipitation particles [43]. Cloud and precipitation particle recognition is important because the properties of these particles affect Earth’s weather. Specifically, there are several properties of cloud and precipitation particles that affect precipitation and radiation, two of which are phase (liquid or ice), and droplet and crystal size [44]. The purpose of cloud and precipitation particle recognition algorithms is to catalogue these properties for later analysis. Images of cloud and precipitation particles are captured when airplanes with special probes fly through clouds. Some representative images of cloud and precipitation particles are presented as Figure 2-7 [45].
Cloud and precipitation particle images are captured by Optical Array Probes (OAPs). All OAPs operate according to the same basic principle, a cloud or precipitation particle travels (at high speed) across an elliptical beam of radiation, casting a shadow on an array of photodiodes [46]. The first OAPs appeared in the early 1970s [47]. OAP development since then has produced probes that can capture vastly more images at higher resolutions [45]. These advancements have resulted in the need for new algorithms that are capable of processing much larger sets of data [48]. As with fingerprints, in the context of this research, cloud and precipitation particle images were simply another signal. The final set of signals that will be examined during this research effort were images of camouflaged objects.
2.7: Camouflage

Camouflage is a method for an object to avoid detection or relay misinformation via deception. Camouflage is common in the natural world has been utilized by mankind as well. The three most common types of camouflage are [49]:

1.) Figure ground blending – An object blending with its surroundings
2.) Figure disruption – Breaking up the surface with highly contrasting shapes
3.) Coincident disruption – Some parts blend with surroundings, while others do not

An example of figure ground blending camouflage is illustrated below in Figure 2-8.

Figure 2-8: Camouflaged flounder

In this instance the flounder is attempting to avoid detection by resting on a bed of similarly colored gravel. The gravel and flounder in the above image can be thought of as two patterns. This raises the possibility that pattern recognition algorithms may be able to discern between the two patterns and thus recognize camouflaged objects.
2.8: Summary

This chapter opened with a discussion of pattern recognition. This discussion provided a brief history of the field, highlighted the roles of pattern recognition examples, and provided some examples of everyday pattern recognition systems. Following this, some detail was provided regarding concepts relevant to the pattern recognition research contained in this document; signals, Markov Chains, and Monte Carlo Integration. These concepts were defined and the history of Markov Chains and Monte Carlo were summarized, which was vital given their role in the research described herein.

The discussion then turned to fingerprints, defining the term fingerprint and providing some early history and observations on fingerprints. The pioneers of fingerprint research and their motivations for developing a better biometric were noted. The development of the Henry System for fingerprint identification and its successes were discussed in some detail. This led into the discussion of refinements to the Henry System, which remained more or less the status quo until the advent of computers. The launch of the FBI’s AFIS system represented a major advancement in the history of fingerprint identification. The speed of modern computers allows local law-enforcement to quickly determine if detained suspects are using aliases, which was one of the major reasons that fingerprint identification was developed in the first place. Following this, the different classes of fingerprint recognition algorithms were developed. A discussion of MCMC algorithms was presented. This included a detailed discussion of Deal’s research, describing how the algorithm functioned and how the resulting Markov Chain transition probabilities could be compared for the purpose of fingerprint identification.
It had been hypothesized that the algorithm discussed herein may be applicable to pattern recognition problems outside of fingerprint recognition. Specifically, it was hypothesized that the pattern recognition algorithm may be applicable to cloud and precipitation particle recognition and camouflage recognition. Therefore, the final sections of this chapter briefly introduced these two subjects. This concludes the background chapter of this document. The next chapter will describe the experiments that were designed to accomplish the research goals enumerated in the Introduction.
Chapter 3: Experimental

This chapter provides a detailed description of the data, equipment, computational tools, mathematics, and experiments that were utilized to research, develop, and quantify the performance of the algorithm described herein. This chapter opens with a discussion of the fingerprint databases utilized during research. This discussion will detail the origins of the fingerprint databases and their utility for this research. Next, the computational tools (hardware and software) employed during research and their relationship to the research will be explained. This discussion will cover software developed during research, including the fingerprint preprocessing function and scale-spectra generating function. Following the discussion of hardware and software, the chapter discusses experiments that were designed to determine whether assumptions underlying the algorithm’s performance were justified. Experiments to determine whether transition probability estimates were normally distributed will be formulated, as well as experiments to determine whether N iterations of a random step were equivalent to a traditional N-step random walk. Following this, metrics to quantify the difference between scale-spectra will be developed. This will lead into a discussion of transition probability error and how to quantify the efficacy of the algorithm. Following this, experiments will be described that will optimize algorithm variables to maximize the fingerprint recognition performance of the algorithm. All of these results culminated in an experiment that recreates the Year 2000 Fingerprint Verification Competition (FVC 2000) [2]. FVC 2000 provided a common set of fingerprints and a standardized testing protocol to benchmark the performance of fingerprint identification algorithms, allowing the algorithm described herein to be compared to previously demonstrated algorithms. Following this, experiments that employ integral
geometry in an attempt to explain the algorithm’s performance will be developed. Finally, the chapter will conclude by detailing the experiments that were designed to qualify the algorithm’s performance for the applications of cloud and precipitation particle recognition and camouflage recognition.

3.1: Fingerprint Databases

A stated goal of this research was to quantify the performance of the fingerprint identification algorithm discussed herein. This allowed the algorithm’s performance to be compared to that of other fingerprint identification algorithms. Historically, comparing algorithms has been difficult because an algorithm’s performance can vary significantly based on the properties of the fingerprints under examination [18]. Maltoni and colleagues partially addressed this problem by organizing the Year 2000 Fingerprint Verification Competition (FVC 2000) [2]. The competition provided fingerprint identification algorithm developers a common set of fingerprints and a standardized testing protocol to quantify the performance of their algorithms. The competition was a success, and spawned subsequent fingerprint verification competitions every two years thereafter. Results from these competitions have been widely cited in the literature, and thus provided a useful metric of algorithm performance.

The fingerprints included in the FVC 2000 databases were captured electronically. Electronically captured fingerprints varied significantly from their ink and paper counterparts [2]. Electronically captured fingerprints are normally much smaller than their paper and ink equivalents, which are typically “rolled” fingerprints that include the impressions of the sides of
the fingers [2]. The four fingerprint databases from FVC2000 were electronically captured or created using the following equipment:

1. A lower-cost optical sensor
2. A lower-cost capacitive sensor
3. A higher-cost optical sensor
4. A synthetic fingerprint generating algorithm

Each database contained eight different impressions (fingerprints) from the same finger of an individual. A total of 110 individuals volunteered for each database, yielding a total of 880 fingerprints per database. For each database, the 880 fingerprints were divided into a set of eight hundred fingerprints (the A set) and a set of eighty fingerprints (the B set). Fingerprints were numbered by the individual (1-110) and the impression (1-8). Fingerprints were represented by the symbol $F_{i,j}$, where the subscript $i$ represented the number of the individual and the subscript $j$ represented the number of the impression. Fingerprints numbered 1-100 were contained in the A set, while fingerprints numbered 101-110 were contained in the B set. The B set was provided to researchers ahead of the competition for training purposes. More information regarding the acquisition of fingerprints and additional details regarding equipment were provided in the report *FVC2000: Fingerprint Verification Competition Report* [2]. The same procedures, equipment, and number of volunteers were used for the year 2002 Fingerprint Verification Competition (FVC 2002) [50]. Fingerprints from FVC 2002 were utilized in this research as well, as will be discussed later.
Fingerprints were referred to by Fingerprint Verification Competition year, database number, individual, and impression. For example, FVC 2000 DB3 F102_5 referred to the fifth fingerprint impression of the hundred and second individual in the third database from the Year 2000 Fingerprint Verification Competition. Notation developed during research purposely mirrored much of the notation used during FVC 2000. Preprocessed fingerprints and scale-spectra (also called templates) were represented by the notation \( F^*_i,j \) and \( T_{i,j} \), respectively. As with raw fingerprints, the subscripts \( i \) and \( j \) referred to the individual and impression, respectively. There was one preprocessed fingerprint and one template for each raw fingerprint. With the details of the fingerprint databases and nomenclature explained, the conversation will move onto the computational tools utilized during research.

### 3.2: Computational Tools

Experimentation was conducted using a 478 MHz AMD K6 Toshiba laptop with 192 MB of RAM running the Windows XP Professional operating system. This computer was utilized for research because it was similar to the computer used during FVC 2000; a 450 MHz Pentium III running Windows NT 4.0 [2]. Thus, algorithm performance metrics of a temporal nature quantified during this research (average time to enroll a fingerprint, etc...) should be roughly comparable to results obtained during FVC 2000.

The main piece of software utilized during research was Version 2.4 of Python. Python is a high-level, interpreted language. Python was deemed an attractive programming language because it allowed for rapid code development, was easy to debug, had a wealth of extensions freely available. The main drawback of Python was its slow processing speed. Because Python
was an interpreted language, it was slow compared to compiled languages (C, FORTRAN, Java, etc...). Therefore, the processing speed of Python was insufficient for the calculation-intensive procedure used to transform fingerprints into their scale-spectra. This problem was addressed by incorporating Python extensions such as Psyco and Pyrex. Psyco is a just-in-time interpreter that speeds execution of Python code by simultaneously running several instances of Python [51]. Pyrex is a tool set designed to ease integration of C code and Python code [52]. Psyco was easier to use than Pyrex, but Pyrex should have provided a larger performance increase. Both Psyco and Pyrex were trialed during research, and their performance benefit was assessed. For integration with Pyrex, the C compiler used during research was Microsoft Visual Studio 2003 .NET.

There were several program modules needed to transform raw fingerprints into their scale-spectra, compare scale-spectra, and quantify the efficacy of these comparisons. A flowchart illustrating these program modules is presented as Figure 3-1.
Regarding the first two blocks in the flowchart, these were developed (but not optimized) under Deal’s research. These program modules were redeveloped under this research. Native Python does not contain a robust set of image handling functions, which would have made
developing the preprocessing module difficult. This issue was resolved by extending Python with the Python Imaging Library (PIL), version 1.1.6. PIL added several functions that made reading, editing, and writing a variety of image file formats straightforward. Fingerprints in the FVC databases were in Tagged Image File Format (.TIFF) and were captured at a resolution of 500 dpi. After fingerprints were preprocessed they were saved in the Portable Network graphics (PNG) format. The PIL was able to read, edit, and write both of these file formats.

The next process block, also developed under this research, transformed the transition probabilities into scale-spectra and saved them as templates. These templates were used by the next two processes to calculate the genuine and impostor matching scores. The genuine and impostor matching scores were calculated according to an established procedure [2]. The final program module calculated and summarized the most important metrics of algorithm performance.

The assemblage of program modules, as illustrated in Figure 3-1, was a powerful tool for developing, optimizing, and quantifying the performance of the pattern recognition algorithm as applied to fingerprints. The assemblage of program modules made it possible to optimize the algorithm. It was possible to determine how the EER was affected algorithm parameters; the beginning and ending step-sizes, the number of random steps in a Monte Carlo Integration at each step size, and the number of step-sizes examined. Thus, the assemblage of program modules described above enabled the completion of the second goal of research, the further development, optimization, and quantification of the algorithm’s performance. While these program modules were developed for fingerprint recognition, they were applicable to other pattern recognition problems as well.
3.3: Fingerprint Preprocessing Function

Preprocessing encompassed all of the methods that were necessary to prepare a fingerprint prior to the creation of the scale-spectra. During preprocessing, the grayscale fingerprint images from the FVC databases were transformed into black and white (binary) fingerprint images and extraneous information was masked. Extraneous information in fingerprint images included the regions outside of the fingerprint itself, and other smudges or smears. The raw grayscale fingerprint images presented in the FVC databases were in the tagged image file format (.TIFF). The preprocessing function read these existing grayscale fingerprint images pixel by pixel, and determined if the pixel would best be represented as a white pixel, a black pixel, or a green pixel (extraneous information). The masked binary images produced by the preprocessing algorithm were saved in the portable network graphics (.PNG) format. The details by which this transformation occurred are discussed in more detail below.

A pixel’s color was determined by examining the pixel itself and the pixel neighborhood. The pixel neighborhood was a NxN square, where N is an odd integer. The pixel neighborhood was always centered on the pixel under examination. This is illustrated graphically as Figure 3-2.
Figure 3-2: Raw fingerprint image (each square is one pixel)
Pixel under examination and the surrounding pixel neighborhood (N = 5)

Note that the pixel neighborhood was undefined near the image border, because the pixel neighborhood would have looked for pixels outside of the image. From a practical standpoint, this meant that the perimeters of all fingerprint images were designated as extraneous information. This was judged acceptable because in practice the perimeter width was small compared to the overall fingerprint dimensions. The width of this perimeter was (N-1)/2 pixels, or about half of the neighborhood width.

When examining the pixel neighborhood, the preprocessing function noted or calculated the following three properties:
1. The standard deviation of grayscale values within the pixel neighborhood

2. The grayscale value of the pixel under examination (a number between 0-255)

3. The average grayscale value within the pixel neighborhood

The first property examined by the preprocessing algorithm was the standard deviation of grayscale values within the pixel-neighborhood. This was calculated according to Equation 3-1.

\[ \sigma_{GV} = \frac{\sum_{i=1}^{i=N} \sum_{j=1}^{j=N} (I_{ij} - I_{avg})^2 / N^2}{3-1} \]

Where:

- \( I_{ij} \) represents the value (0-255) of the pixel at row \( i \) and column \( j \) of the pixel neighborhood

- \( I_{avg} \) represents the average value (0-255) of all pixels within the neighborhood

The average grayscale value in the pixel neighborhood, \( I_{avg} \), is calculated according to Equation 3-2:

\[ I_{avg} = \frac{1}{N^2} \sum_{i=1}^{i=N} \sum_{j=1}^{j=N} I_{ij} \]
Homogenous areas exhibited relatively low standard deviations in their grayscale values, while inhomogeneous areas yielded relatively high standard deviations in their grayscale values. Homogenous areas were typically not part of the fingerprint, they were either outside of the fingerprint area, or were smudges or smears. Inhomogeneous areas were typically part of the fingerprint, these areas were usually composed of alternating black and white areas (ridges and valleys). If the standard deviation of grayscale values within the pixel neighborhood was below a certain threshold (denoted as \( \sigma_{GVt} \)), the pixel under examination was designated as extraneous information. If the standard deviation of grayscale values within the neighborhood were equal to or greater than \( \sigma_{GVt} \), then the pixel under examination was designated as being part of the fingerprint. In this case, the preprocessing function required more information to determine if the pixel under examination should be white or black.

The next property examined by the preprocessing function was the grayscale value of the pixel under examination. This grayscale value ranged from 0 (pure black) to 255 (pure white). This value was compared to two thresholds, a lower limit (black threshold) and an upper limit (white threshold), denoted as \( I_{Bi} \) and \( I_{Wi} \), respectively. If the grayscale value of the pixel under examination was below the lower threshold or above the upper threshold, the pixel was automatically classified as black or white, respectively. This is described by the following two equations:

\[
\begin{align*}
\text{If } I_{ij} < I_{Bi}, & \text{ Then } I_{ij} = 0 \\
\text{If } I_{ij} > I_{Wi}, & \text{ Then } I_{ij} = 255
\end{align*}
\]

Equation 3-3

Equation 3-4
If the pixel under examination was between the two thresholds, the preprocessing function needed to consider additional information to determine if the pixel under examination should be designated as black or white.

The final property examined by the preprocessing function was the grayscale value of the pixel under examination relative to the average grayscale value of the pixel neighborhood. If the pixel under examination had a grayscale value lower than the neighborhood average, it was designated as black. If the pixel under examination had a value equal to or greater than the neighborhood average, it was designated as white. This is illustrated by the following equations:

\[
\begin{align*}
\text{If } I_{ij} < I_{\text{Avg}}, \text{ Then } I_{ij} &= 0 \\
\text{If } I_{ij} \geq I_{\text{Avg}} \text{ Then } I_{ij} &= 255
\end{align*}
\]

Following this step, the preprocessing function had designated every pixel in the fingerprint image as black, white, or green (extraneous information). A flowchart describing the operation of the preprocessing function is presented as Figure 3-3.
Figure 3-3: Preprocessing function flowchart
In summary, the preprocessing function walked through grayscale fingerprint images one pixel at a time, determining if each pixel should be designated as black, white, or green (extraneous information). The preprocessing function examined a pixel and its neighbors. The preprocessing algorithm required a neighborhood size (N), neighborhood grayscale standard deviation threshold ($\sigma_{GVt}$), and a black threshold and white threshold ($l_{bt}$ and $l_{wt}$, respectively). These arguments were developed using fingerprint database FVC2002 DB1A until a “good” set of black and white masked fingerprints were produced. Developing a method to quantify the quality ("good" vs "bad") of preprocessed images was outside of the scope of this work. Instead, the arguments were manually adjusted until the output (processed fingerprint images $F^*_{ij}$) looked "good" to an observer's eye. For this work "good" was defined as an image that masked areas outside of the fingerprint and masked smudges and smears; while simultaneously preserving the areas that contained ridges, valleys, and minutiae. The preprocessing algorithm was then applied to the other fingerprint databases to confirm that one set of parameters can yield effective pre-processing for all fingerprint databases. The next function wrote during research transformed the preprocessed fingerprints into their associated scale-spectra.

3.4: Scale-Spectra Generating Function

Following the preprocessing function described in the previous section, a raw fingerprint image $F_{ij}$ has been transformed into a preprocessed fingerprint image $F^*_{ij}$. Next, the preprocessed fingerprint image $F^*_{ij}$ was transformed into its corresponding scale-spectra (also known as a template, denoted $T_{ij}$). As explained in the introduction, this transformation
occurred via many iterations of a random step at various step-sizes. A flowchart describing the scale-spectra generation via Monte Carlo Integration is presented as Figure 3-4. After the transition probabilities were estimated using Monte Carlo Integration, the matrix invariants were calculated at each step-size examined, yielding the various scale-spectra. To maintain consistent nomenclature with the FVC, the scale-spectra of fingerprints were also referred to as templates, denoted as $T_{ij}$. For a more detailed account of this process that transforms preprocessed fingerprint images into scale-spectra, the reader is referred to the research of Deal [1]. The computer code to transform a preprocessed fingerprint into its corresponding scale-spectra was a custom Python script integrated with C code via Pyrex. As previously mentioned, the majority of the computer code used to transform fingerprints into scale-spectra was written in Pyrex, due to the poor numeric processing speed of Python.
Figure 3-4: Scale-spectra generating function flowchart
The time required to transform a fingerprint into its corresponding scale-spectra is a function of the number of step-sizes examined and the number of random steps occurring at each step-size. During early experiments (detailed in Section 3.5), there were 160 step-sizes examined, from 0.0 \( \mu \text{m} \) to 2032 \( \mu \text{m} \), by increments of 12.7 \( \mu \text{m} \). During these early experiments, the transition probabilities at a given scale were estimated via 500,000 random steps. In hindsight, such a large number of iterations and large range of step-sizes were not necessary. For experiments designed to optimize and quantify the performance of the fingerprint identification algorithm, the choice of step-sizes under consideration and the number of random steps at each step-size were carefully adjusted to achieve satisfactory performance while minimizing computational load.

If, for any reason, the scale-spectra generating function failed to transform a fingerprint into its corresponding template, this event would be noted as a Failure To Enroll (FTE). The FTE rate was a widely cited metric of fingerprint identification algorithm performance. A failure to enroll occurs when a fingerprint identification algorithm cannot classify or quantify the features of a fingerprint according to its rules. In practice, the algorithm described herein does not experience failures to enroll. This is due to the algorithm’s mode of operation, which simply tabulated whatever it observed during the random steps. This contrasted with other algorithms, which may search for features (ridges, minutiae, etc...) and may fail to enroll a fingerprint if these features could not be identified. Early in research, the scale-spectra were used to justify assumptions regarding the algorithm’s operation. These assumptions and the experiments designed to justify them are detailed in the next section.
3.5: Justifying Assumptions

A stated goal of this research was to justify assumptions that were made throughout the course of research. With respect to earlier work, Deal assumed that the transition probabilities calculated via many iterations of a random step were equivalent to the transition probabilities calculated via a random walk. This assumption was utilized to decrease processing time. A random walk was an inherently serial process, the previous step must have been known before the next position could have been calculated. Using the random steps, the inherently serial process was transformed into a process amenable to parallel processing, which can provide significant performance benefits when coupled with modern multiple core processors. Another assumption made early in this research was that the error associated with the estimated transition probabilities were easily quantified according to a familiar relationship. This relationship (discussed in more detail later) assumed that the errors in transition probabilities estimated via Monte Carlo Integration were normally distributed. These assumptions needed to be justified however. The experiments that were designed to test these assumptions are discussed in the following subsections.

3.5.1: Assumption that Estimations Were Normally Distributed

In later sections, equations describing the variability inherent in the transition probabilities estimated via Monte Carlo Integration will be presented. These equations are valid only if the errors associated with the estimated transition probabilities are normally distributed. Thus, it was necessary to determine if the error associated with transition probabilities estimated via Monte Carlo Integration were indeed normally distributed. For this experiment,
fingerprint F\textsubscript{102_1} from FVC2000 DB1 was arbitrarily chosen. The transition probabilities associated with this fingerprint were estimated 10,000 times. For each estimation of the transition probabilities, there were one million random steps, at a step-size of 1000 \( \mu \text{m} \). A histogram of these transition probabilities will be compared to a normal distribution. If the histogram of the estimated transition probabilities appeared normally distributed, it will be straightforward to calculate the error inherent in the estimated transition probabilities.

### 3.5.2: Assumption of Equivalence Between a Random Walk and Random Steps

Recall that many random steps were preferred over a random walk. A random walk was a serial processes and random steps could be run in parallel – the latter was much more amenable to parallel processing. Thus far it has been assumed that transition probabilities calculated via many iterations of a random step were equivalent to the transition probabilities calculated via a random walk. This assumption was tested by comparing transition probabilities calculated via many iterations of a random step to transition probabilities calculated via a random walk.

Fingerprint F\textsubscript{103_1} from FVC2000 DB3 was arbitrarily chosen for this experiment. There were 161 random walks performed on this fingerprint. Each random walk utilized a different step-size, from 0 \( \mu \text{m} \) to 2032 \( \mu \text{m} \) (by increments of 12.7 \( \mu \text{m} \)). Each random walk consisted of one million steps. The transition probabilities calculated at each step-size were noted and the template was saved. These transition probabilities were compared to the transition probabilities yielded via one million iterations of a random step. The random step iterations occurred at 161 different step-sizes, from 0 \( \mu \text{m} \) to 2032 \( \mu \text{m} \) (by increments of 12.7 \( \mu \text{m} \)). If the
transition probabilities calculated via a random walk are within the error of the transition
probabilities calculated via many iterations of a random step, then equivalence between a
random walk and random steps will be assumed. After earlier assumptions are justified,
research will focus on metrics to compare and quantify the difference between fingerprint
templates.

3.6: Metrics of Fingerprint Comparison

After the fingerprints were transformed into templates, it was necessary to quantify
how similar or different two templates were. Comparing templates required the development
of a metric to quantify the difference between templates. The obvious solution was to compare
two templates by calculating the difference between one or more of the respective transition
probabilities. Alternatively, a metric that incorporated the four transition probabilities could be
developed. Thus, templates could be compared by calculating the difference between two
metrics that utilized all four transition probabilities. One possibility was to utilize the invariants
of the two by two sub-matrix that contained the four transitions probabilities estimated at a
given step-size. There were four invariants associated with the 2x2 transition probability sub-
matrix; the determinant, the trace, the first Eigenvalue, and the second Eigenvalue. These
invariants are represented by the following equations:

\[
P = \begin{bmatrix} p_{BB} & p_{BW} \\ p_{WB} & p_{WW} \end{bmatrix}
\]

\[
det(P) = p_{BB}p_{WW} - p_{BW}p_{WB}
\]
Where:

\[ \text{det}(P) \] is the determinant of the probability matrix

\[ \text{tr}(P) \] is the trace of the probability matrix

\[ \lambda_1(P) \] is the first Eigenvalue of the probability matrix

\[ \lambda_2(P) \] is the second Eigenvalue of the probability matrix

As an example, the determinant of the sub-matrices could have been calculated, thereby representing a fingerprint as a one-dimensional matrix of scalars. This was appealing because it used all four transition probabilities and yielded straightforward comparisons. The one dimensional matrix composed of the determinant of the transition probabilities was referred to as the determinant scale-spectrum. The difference between two fingerprints could then be quantified by calculating the difference between their respective determinant scale-spectra. The difference metric to compare two determinant scale-spectra is presented as Equation 3-12. This equation yielded a dimensionless scalar that represented the determinant scale-spectra difference between two fingerprints

\[
D_{det}(x, y) = \sum_{s=0}^{n} \sqrt{\frac{\det(P_x)_s^2 - \det(P_y)_s^2}{2}}
\]  

Where:
\( \text{det} (P_x)_s \) is the determinant of fingerprint x’s scale-spectrum at step-size s.

\( \text{det} (P_y)_s \) is the determinant of fingerprint’s y’s scale-spectrum at step-size s.

\( n \) is the largest step-size in the scale-spectra.

The corresponding difference metrics for the other three probability matrix invariants were developed as illustrated in Equations 3-13 through 3-15.

\[
D_{tr}(x, y) = \sum_{s=1}^{n} \sqrt{\text{tr}(P_x)_s^2 - \text{tr}(P_y)_s^2}
\]

\[
D_{\lambda_1}(x, y) = \sum_{s=1}^{n} \sqrt{\lambda_1(P_x)_s^2 - \lambda_1(P_y)_s^2}
\]

\[
D_{\lambda_2}(x, y) = \sum_{s=1}^{n} \sqrt{\lambda_2(P_x)_s^2 - \lambda_2(P_y)_s^2}
\]

These difference metrics provided a straightforward method to compare templates. It was hypothesized that using multiple metrics may be more effective than a single metric, and that the optimal metric may change as a function of the step-sizes under examination. To address this possibility, a hybrid metric was conceived, which took the following form:

\[
D_{Hy}(x, y) = a \ast D_{\text{det}}(x, y) + b \ast D_{tr}(x, y) \\
+ c \ast D_{\lambda_1}(x, y) + d \ast D_{\lambda_2}(x, y)
\]

Where a, b, c, and d were weighting factors that may be a function of scale.
Finally, it seemed likely that the effectiveness of the above metrics would be a function of the number of random steps, the step-sizes under examination, and other variables. However, before these questions could be tested, it was necessary to determine the amount of error inherent in the above estimated transition probabilities.

### 3.7: Quantifying Transition Probability Estimation Errors

The error associated with the estimated transition probabilities can be represented as follows [14]:

\[
\begin{align*}
\varepsilon_{WW} &= \pm 1.96 \sigma_{WW} \\
\varepsilon_{WB} &= \pm 1.96 \sigma_{WB} \\
\varepsilon_{BW} &= \pm 1.96 \sigma_{BW} \\
\varepsilon_{BB} &= \pm 1.96 \sigma_{BB}
\end{align*}
\]

Where \(\sigma_{WW}, \sigma_{WB}, \sigma_{BW},\) and \(\sigma_{BB}\) represent the standard deviations of the corresponding transition probability estimations.

The above developed error equations assumed a 95% confidence interval. Note that the standard deviations represent the variability in the estimated transition probabilities produced by Monte Carlo integration. This variability is a function of the number of random steps employed during the Monte Carlo Integration. More random steps produced lower standard deviations.
deviations. The transition probability estimate error is inversely proportional to the square root of the number of random steps employed. In practice, the standard deviations of the four estimated transition probabilities were similar, allowing the above equations to be represented as a single equation, illustrated as Equation 3-21.

\[ \varepsilon_{WW} = \varepsilon_{WB} = \varepsilon_{BW} = \varepsilon_{BB} = \pm 1.96\sigma_{Avg} \]

Using the above equation, it was possible to determine the error associated with the estimated transition probability matrix invariants. The value of the trace (including error) was represented as follows:

\[ \text{Tr}(P) + \varepsilon_{Tr} = P_{WW} + \varepsilon_{WW} + P_{BB} + \varepsilon_{BB} \]
\[ = (P_{WW} + P_{BB}) \pm (|\varepsilon_{WW}| + |\varepsilon_{BB}|) \]

Subtracting the trace from each side of the above equation illustrated that the total error associated with the trace was the sum of the error associated with the white-white transition probability and the black-black transition probability. Thus, the error associated with the estimated trace invariant was easily calculated as follows:

\[ \varepsilon_{Tr} = \pm (|\varepsilon_{WW}| + |\varepsilon_{BB}|) = \pm 3.92\sigma_{Avg} \]

Calculating the error associated with the determinant invariant was complicated because of the multiplication of errors. Consider the error inherent in the determinant by examining the following equation:
Expanding Equation 3-24 yielded:

\[
\det(P) + \varepsilon_{\text{det}} = (P_{WW} + \varepsilon_{WW}) \cdot (P_{BB} + \varepsilon_{BB}) - (P_{WB} + \varepsilon_{WB}) \cdot (P_{BW} + \varepsilon_{BW})
\]

Equation 3-26 was rearranged to a more usable form:

\[
\det(P) + \varepsilon_{\text{det}} = (P_{WW}P_{BB} + P_{WW}\varepsilon_{BB} + P_{BB}\varepsilon_{WW} + \varepsilon_{WW}\varepsilon_{BB}) - (P_{WB}P_{BW} + P_{WB}\varepsilon_{BW} + P_{BW}\varepsilon_{WB} + \varepsilon_{WB}\varepsilon_{BW})
\]

Subtracting the determinant from both sides of the Equation 3-26 yielded the error.

\[
\varepsilon_{\text{det}} = P_{WW}\varepsilon_{BB} + P_{BB}\varepsilon_{WW} + \varepsilon_{WW}\varepsilon_{BB} + P_{WB}\varepsilon_{BW} + P_{BW}\varepsilon_{WB} + \varepsilon_{WB}\varepsilon_{BW}
\]

The above equation can be simplified by assuming that the errors were small compared to the transition probabilities. In this case, the terms which are the product of two errors are negligible and the previous equation reduces to the following:
\[ \varepsilon_{Det} = P_{WW}\varepsilon_{BB} + P_{BB}\varepsilon_{WW} + P_{WB}\varepsilon_{BW} + P_{BW}\varepsilon_{WB} \]  \hspace{1cm} 3-28

which was rearranged into the following form:

\[ \varepsilon_{Det} = 1.96\sigma_{Avg}(P_{WW} + P_{BB} + P_{WB} + P_{BW}) \]  \hspace{1cm} 3-29

The four transition probabilities, by definition, sum to one. Therefore, Equation 3-29 reduces to the following:

\[ \varepsilon_{Det} = \sigma_{Avg}1.96\left(\frac{1}{N}\right)^{1/2} = \frac{1}{2}\varepsilon_{Tr} \]  \hspace{1cm} 3-30

The equations for the error associated with the First and Second Eigenvalues include the equations for the error associated with the trace and determinant:

\[ \lambda_1(P) + \varepsilon_{\lambda_1} = \pm \frac{1}{2}\left(Tr \pm \varepsilon_{Tr} + \sqrt{Tr^2 + 2Tr\varepsilon_{Tr} + \varepsilon_{Tr}^2 + 4Det \pm 4\varepsilon_{Det}}\right) \]  \hspace{1cm} 3-31

Rearranging Equation 3-31, and assuming that the squared error term is small, yields the following:
Examining Equation 3-30, it was observed that the error associated with the determinant is half of the error associated with the trace. Thus, the Equation 3-32 can be rewritten in the following form:

\[ E_{\lambda_1}(P) + \varepsilon_{\lambda_1} = \pm \frac{1}{2} \left( Tr \pm \varepsilon_{Tr} + \sqrt{Tr^2 + 4Det} \pm 4\varepsilon_{Det} \pm 2Tr\varepsilon_{Tr} \right) \]

\[ 3-32 \]

Equation 3-33 cannot be simplified. To proceed, it is necessary to make some further assumptions. Typically, the non-error terms (the trace and the determinant) will be larger than the error terms. This suggests that the error terms will not significantly affect the value of the quantity within the square root. In this case, the error associated with the Eigen Values is half the error associated with the trace, which is the same as the error associated with the determinant:

\[ \varepsilon_{\lambda_1} = \frac{1}{2} \varepsilon_{Tr} = \varepsilon_{Det} \]

\[ 3-34 \]

Using Equations 3-23, 3-30, and 3-33 it will be possible to graph the error in the transition probability matrix invariants as a function of the number of random steps utilized at a given step-size. With the errors of the transition probability matrix invariants quantified, research will
focus on the development of experiments to quantify the efficacy of previously developed difference metrics.

3.8: Quantifying the Efficacy of the Difference Metrics

As discussed in the Introduction, the Equal Error Rate (EER) was the most often cited metric describing the performance of a fingerprint identification algorithm. In addition to the EER, it was important to quantify the False Non-Match Rate (FNMR), False Match Rate (FMR), Failure To Enroll (FTE) rate, and the Failure To Match (FTM) rate of an algorithm. Beyond algorithm efficacy, it was also important to quantify the operating speed of the algorithm. Experiments will be designed to quantify the time required to preprocess fingerprints and generate the templates from preprocessed fingerprints (the enroll time). Experiments were also designed to quantify the time required to compare templates (the match time). The subsequent sub-sections will detail the experiments for calculating these performance metrics. For a more detailed explanation of common fingerprint performance metrics the reader is referred to FVC2000: Fingerprint Verification Competition Report [2].

3.8.1: Genuine and Impostor Matching Scores

A unitless scalar was produced when the difference metrics presented in Equation 3-12 through Equation 3-16 were applied to two templates. If the templates of two fingerprints that originated from the same finger were compared, the unitless scalar produced by a difference metric would be designated as a “genuine matching score.” Conversely, if the templates for two
fingerprints that originated from different fingers were compared, the unitless scalar produced by the difference metric would be designated as an “impostor matching score.”

During FVC2000, the A-sets (800 fingerprints) of each database were used to calculate the genuine and impostor matching scores [2]. To calculate the genuine matching scores, each fingerprint template $T_{ij}$ was compared to templates from the same finger, $T_{ik}$, where $j < k <= 8$. This is illustrated below.

<table>
<thead>
<tr>
<th>$T_{i_1}$</th>
<th>$T_{i_2}$</th>
<th>$T_{i_3}$</th>
<th>$T_{i_4}$</th>
<th>$T_{i_5}$</th>
<th>$T_{i_6}$</th>
<th>$T_{i_7}$</th>
<th>$T_{i_8}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{i_2}$</td>
<td>$T_{i_3}$</td>
<td>$T_{i_4}$</td>
<td>$T_{i_5}$</td>
<td>$T_{i_6}$</td>
<td>$T_{i_7}$</td>
<td>$T_{i_8}$</td>
<td></td>
</tr>
<tr>
<td>$T_{i_3}$</td>
<td>$T_{i_4}$</td>
<td>$T_{i_5}$</td>
<td>$T_{i_6}$</td>
<td>$T_{i_7}$</td>
<td>$T_{i_8}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T_{i_4}$</td>
<td>$T_{i_5}$</td>
<td>$T_{i_6}$</td>
<td>$T_{i_7}$</td>
<td>$T_{i_8}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T_{i_5}$</td>
<td>$T_{i_6}$</td>
<td>$T_{i_7}$</td>
<td>$T_{i_8}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T_{i_6}$</td>
<td>$T_{i_7}$</td>
<td>$T_{i_8}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T_{i_7}$</td>
<td>$T_{i_8}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Thus, there were 28 genuine matching scores for each of the 100 individuals, or 2,800 genuine matching scores for each database. To calculate the impostor matching scores, each fingerprint template, $T_i$, was matched against each fingerprint template from different fingers, $T_{ik}$, where $i < k <= 100$. This is illustrated below.

<table>
<thead>
<tr>
<th>$T_{1_1}$</th>
<th>$T_{2_1}$</th>
<th>$T_{3_1}$</th>
<th>$T_{4_1}$</th>
<th>$T_{98_1}$</th>
<th>$T_{99_1}$</th>
<th>$T_{100_1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{2_1}$</td>
<td>$T_{3_1}$</td>
<td>$T_{4_1}$</td>
<td>$T_{5_1}$</td>
<td>$T_{98_1}$</td>
<td>$T_{99_1}$</td>
<td>$T_{100_1}$</td>
</tr>
</tbody>
</table>
$T_{98.1}$ was compared to $T_{99.1}, T_{100.1}$

$T_{99.1}$ was compared to $T_{100.1}$

Thus, the number of impostor matching scores was equal to the sum of numbers from 1 to 99, which yielded 4,950.

Histograms of the genuine and impostor matching scores were created to illustrate how the various difference metrics separate the genuine and impostor matching scores. It was hypothesized that comparing templates originating from the same finger will yield lower differences, so the histogram of genuine matching scores should have exhibited a lower mean than the histogram of impostor matching scores. After these histograms were generated, the genuine and impostor matching scores were utilized to calculate the FMR, FNMR, and EER.

### 3.8.2: FMR, FNMR, and EER

All genuine and impostor scores were normalized by the same factor, such that the highest score (either genuine or impostor) was assigned a value of one. Further, a new variable, called the threshold, which could take any value between zero and one, was created. If the difference metric calculated that the difference between two templates was less than the threshold, then the parent fingerprints were judged to have originated from the same finger. Conversely, if the difference metric calculated that the difference between two templates was greater than or equal to the threshold, then the parent fingerprints were judged to have
originated from different fingers. There were four possible outcomes when comparing two templates to the threshold [18]:

1. True Positive. Two fingerprints from the same finger were judged indistinguishable. In biometric parlance, the probability of a true positive was called the match rate. In this case, the difference metric was less than the threshold.

2. True Negative. Two fingerprints from different fingers were judged unique. In biometric parlance, the probability of a true negative was called the non-match rate. In this case, the difference metric was greater than or equal to the threshold.

3. False positive. Two fingerprints from different fingers were judged indistinguishable. In biometric parlance, the probability of a false positive was called the false match rate (FMR). In this case, the difference metric was less than the threshold.

4. False negative. Two fingerprints from the same finger were judged unique. In biometric parlance, the probability of a false negative was called the false non-match rate (FNMR). In this case, the difference metric was greater than or equal to the threshold.

The FMR and FNMR depend heavily upon the threshold. If the threshold was extremely small, then all fingerprints were judged unique. In this case, the FMR was zero and the FNMR was near one. Alternatively, if the threshold was extremely large, then all fingerprints were judged indistinguishable. In this case, the FMR was near one, and the FNMR was zero. As the threshold was varied from a small number to a large number, the FMR and FNMR crossed paths. The value where the FMR equals the FNMR is known as the Equal Error Rate (EER) [18].
The EER was the single most important metric for quantifying the efficacy of a fingerprint identification algorithm. The EER of a perfect fingerprint identification algorithm would have been zero. A fingerprint identification algorithm that randomly guessed if two fingerprints were unique or indistinguishable would have an EER near 50%.

### 3.8.3: Other Metrics of Interest

Other metrics of interest included the zeroFMR and zero FNMR. The zeroFMR was the lowest value of FNMR at which no false matches occurred [18]. Likewise, the zeroFNMR was the lowest value of FMR at which no false non-matches occurred [18]. Additionally, the FTE rate (discussed earlier) and the Failure To Match (FTM) rate were often cited as metrics of fingerprint identification algorithm performance. A FTM occurred when an algorithm determined that there was insufficient information to match a template under consideration to another template within the database. Note that a failure to match was different from a false-match. A failure to match implied the algorithm could not attempt a match, whereas a false-match implied an algorithm wrongly predicted that two templates belonged to the same finger [18]. The FTM rate was calculated as the number of failed matches divided by the total number of attempts. In practice, the algorithm discussed herein did not experience failure to match events (because the algorithm did not fail to enroll fingerprints).

Other metrics of interest were of a temporal nature and included the average enroll time and the average match time. The average enroll time was the time necessary to transform a raw fingerprint into a template. The enroll time included the preprocessing function and the scale-spectra generating function. The average enroll time was noted for all four databases and
was calculated as the total enroll time divided by the number of enrolled fingerprints. The average match time was defined as the time necessary to calculate a difference given a difference metric and two fingerprints. The average match time was noted for all four databases and was calculated as the total matching time divided by the total number of matches. With the tools to quantify algorithm efficacy and performance developed, experimentation focused on optimizing the algorithm.

3.9: Experiments to Optimize Algorithm Performance

There were many parameters that affected the performance of the algorithm described herein. Optimizing some of these parameters was relatively straightforward, while optimizing other parameters involved tradeoffs between algorithm performance and processing time. The following subsections will detail the experiments designed to optimize the algorithm’s performance.

3.9.1: Optimizing the Difference Metric

As described in Section 3.6, Monte Carlo Integration yielded a scale-spectrum composed of 2x2 sub-matrices, which were utilized to formulate four difference metrics based on the four sub-matrix invariants; the determinant, the trace, and the two Eigenvalues. It was hypothesized that some step-sizes would yield more useful information than others, and it was also hypothesized that examining multiple step-sizes would determine which scales were helpful or deleterious to algorithm performance. To test the first hypothesis, an experiment was planned to plot the EER of the four difference metrics as a function of the step-sizes under examination.
For this experiment, the A-set of DB1 from FVC2002 was utilized. The Monte Carlo Integration occurred at 161 various step-sizes, evenly distributed between 0 μm and 2032 μm. The difference metrics were applied to the appropriate templates, which included all step-sizes from 0.0 μm up to the step-size under examination. The result will be a plot illustrating the EER of the difference metrics as a function of the largest scale examined. If the slope of the EER vs. step-size line increased for a given difference metric, then the inclusion of that step-size was deleterious to algorithm performance. If the slope of the EER vs. step-size line decreased for a given difference metric, then the inclusion of that step-size was beneficial to algorithm performance. Next, it was hypothesized that it would be possible to determine which metric performed best at any given step-size.

For this experiment, the A-set of DB1 from FVC2002 was utilized. The Monte Carlo Integration occurred at 80 various step-sizes, evenly distributed between 0 μm and 1003 μm. The difference metrics were applied to the sub-matrix associated with a single step-size, and the EER was calculated for each difference metric. Note that this experiment considered only one step-size at a time. The result will be a plot illustrating EER as a function of step-size examined. Whichever difference metric demonstrated the lowest EER at a particular step-size was the optimal metric for that step-size.

Thus, through the above two experiments, it could be determined which difference metric performed best at a given step-size and which step-sizes should have been included in the analyses. This result made it possible to produce a hybrid metric, in the form described by Equation 3-16. An experiment was designed to optimize this hybrid metric. The weight factor(s) in Equation 3-16 were optimized by plotting the EER as a function of weighting parameters, and
the optimal weighting factor occurred where the EER was a minimum. The optimal difference metric was developed following the above steps. Next, it was necessary to investigate the tradeoff between estimate error and processing speed.

3.9.2: Step-Size Increments in Given Step-Size Range

Next, the step-size increment was adjusted. As before, adjusting the step-size increment presented a tradeoff between increased information at the expense of computational load. Decreasing the step-size increment meant that more transition probability matrices would be estimated for a given step-size range. Increasing the step-size increment resulted in less information but also less computational load. To determine where the best-demonstrated operating point was, the EER was graphed as a function of step-size increment for a predetermined step-size range, from 0.0 μm to 1003 μm. Step-size increments tested during experimentation included; 5 μm, 10 μm, 25 μm, 50 μm, 200 μm, and 250 μm. For this experiment, there were 5,000 random steps performed at each step-size increment throughout the step-size range. This experiment was performed using the A-set of DB1 from FVC 2002.

3.9.3: Adjusting the Number of Random Steps

The number of random steps employed in the Monte Carlo Integration that occurred at each step-size was variable. More random steps yielded more confidence in transition probability estimations, consistent with the previously developed error equations. However, increased numbers of random steps increased computation time. Therefore, the goal was to reduce the number of random steps as far as possible (to reduce computational load) without
degrading algorithm performance. An experiment was designed to quantify the EER of the best-demonstrated difference metrics as a function of the number of random steps. For this experiment, FVC2002 DB1A was utilized. The Monte Carlo Integration occurred at 41 various step-sizes, evenly distributed between 0 μm and 1000 μm. The number of random steps were varied from 100 to 7,000. The EER was plotted as a function of the number of random steps.

3.10: Recreating the FVC2000 Competition

Following the previously described experiments, the preprocessing function, scale-spectra generating function, and difference metric functions have been finalized. Next, it was necessary to quantify the performance of the algorithm. The algorithm’s performance was quantified by recreating the Year 2000 Fingerprint Verification Competition. The competition involved eleven participants from academia and private industry [2]. Details of the procedures and competition were contained in the FVC 2000 Report [2].

The first step in the competition consisted of enrolling all 800 fingerprints from the A-set databases. Note that for the algorithm described herein, enrolling included the preprocessing function and the scale-spectra generating function. When an algorithm attempted to enroll a fingerprint, it was either successful or failed. There are three ways that an algorithm could have failed to enroll a fingerprint:

1. Fail – the algorithm declares it cannot enroll a fingerprint
2. Timeout – the algorithm takes too long to enroll a fingerprint (more than 15 seconds)
3. Crash – the algorithm ceases to function

The fraction of failures (of any type) are summed and stored as REJENROLL [2].

Following a successful enrollment, Fingerprint $F_{ij}$ was successfully transformed into a preprocessed fingerprint $F^*_{ij}$ and Template $T_{ij}$ (the scale-spectrum). The next step was to calculate differences between templates using the metrics described in Section 3.6. The differences between templates from the same finger were designated as genuine matching scores, while the differences between templates from different fingers were designated as impostor matching scores. Genuine matching scores and impostor matching scores were calculated as described in Section 3.8.1. If the difference metrics were unable to calculate a genuine or impostor score given two templates, the event would be recorded as a failure to match. After these scores were tabulated, the FMR and FNMR were calculated, which yielded the EER, as described in Section 3.8.2. In summary, the following metrics of fingerprint identification algorithm performance were tabulated according to the procedures described in FVC 2000 [2].

1) The time to enroll – The average time required (per fingerprint) to preprocess and transform a fingerprint into its scale-spectrum (template).

2) Matching time – The average time required (per match) to calculate the impostor and genuine matching scores.

3) FTE rate – The fraction of fingerprints that the algorithm is unable to enroll

4) FTM rate – The fraction of fingerprints that the algorithm is unable to match

5) EER – The value where the FMR equals the FNMR.
The metrics described above were recorded and the performance of the algorithm described herein was compared to other algorithms which entered FVC 2000.

### 3.11: Relevance of Integral Geometry

Integral geometry is a field of study involving concepts such as probability, measure, groups, and geometry [53]. Integral geometry originated with the problem of Buffon’s needle, named after a French polymath Georges-Louis Leclerc, the Comte de Buffon [53]. Leclerc sought a solution to a seemingly simple problem: What is the probability that a needle of given length will traverse a crack when dropped onto a floor constructed from planks of equal width. This question can be represented graphically, as is done so below in Figure 3-5.

![Figure 3-5: Two needles on a plank floor](image)

Note that in Figure 3-5, one needle traverses the crack while another does not. Leclerc solved this problem in his Essai d’ Arithme’tique Morale (1777) [53]. For the simpler case, when the
length of the needle is less than the width of the plank, the probability that the needle will cross a line is:

\[ P = \int_{\theta=0}^{\pi/2} \int_{x=0}^{l \sin \theta} \frac{4}{t \pi} \, dx \, d\theta = \frac{2l}{t \pi} \]  \hspace{1cm} 3-35

Where:

- \( t \) is the width between planks (analogous to fingerprint ridge width)
- \( l \) is the length of the needle (analogous to the step size)

Two random variables \( \theta \) and \( x \) represented the acute angle between the needle and the line, and the distance from the center of the needle to the closest line, respectively.

The parallels between the above situation and a fingerprint undergoing a series of random steps are apparent. If a fingerprint is simplified as a series of parallel ridges and valleys of equal width, then the transition probabilities can be calculated exactly using the above equation. If the needle does not cross a line, then a white-white or black-black transition was observed. If the needle does cross a line, then a white-black or black-white transition has occurred. Thus, the transition probabilities for this case can be solved for analytically, as follows:

\[ P_{WW} = P_{BB} = \frac{1}{2} \left( 1 - \frac{2l}{t \pi} \right) \]  \hspace{1cm} 3-36

\[ P_{WB} = P_{BW} = \frac{1}{2} \left( \frac{2l}{t \pi} \right) \]  \hspace{1cm} 3-37
Using these transition probabilities it was possible to analytically solve for the trace, determinant, and two Eigen Values. The validity of the above fingerprint model was tested by comparing the calculated scale-spectra against experimentally estimated scale-spectra. For this experiment, a fingerprint that is nearly 50% black and 50% white (with nearly equal ridge and valley widths) was chosen. Following this, all experiments on the subject of fingerprint identification will have been completed. The following two-subsections will discuss experiments designed to qualify the algorithm’s performance as a cloud and precipitation particle and camouflage recognition algorithm.

3.12: Cloud and Precipitation Particle Recognition Experiments

A series of experiments were designed to qualify the performance of the algorithm as a cloud and precipitation particle recognition algorithm. All of these experiments employed the same algorithm arguments: a beginning step-size of 0 µm, an ending step-size of 231 µm, a step-size increment of 1.76 µm, and 200,000 random steps occurring at each step-size. The first experiment was designed to determine whether different cloud and precipitation particles produce significantly different scale-spectra. For this experiment, three particles were transformed into their corresponding scale-spectra. Following this experiment, another experiment was performed to determine if scale-spectra features can be correlated with particle size. For this experiment, a particle and a scaled version of itself were transformed into their corresponding scale-spectra, and the scale-spectra were compared. If the scale-spectra are qualitatively indistinguishable but qualitatively different, it suggests that the scale where
certain scale-spectra features occur may be used to predict particle size. The next experiment compared a particle with two lower-resolution copies of itself. This experiment determined how resolution differences affect scale-spectra. The original particle had a resolution of 10 \( \mu \text{m} \), while the two copies had resolutions equivalent to 40 \( \mu \text{m} \) and 100 \( \mu \text{m} \), respectively. The final experiment demonstrated that the scale-spectra produced were independent of the orientation of the original image. For this experiment, a particle was compared to two rotated versions of itself, one rotated by 90 degrees and one rotated by 180 degrees.

3.13: Camouflage Recognition Experiments

The final experiment was a gross assessment of the algorithm’s potential as a camouflage recognition algorithm. For this experiment, a picture of flounder hidden on a bed of gravel was utilized. Four smaller binary images were extracted from the larger image; three of the images contained gravel and one image contained a piece of the flounder. Scale-spectra were generated for all four of these images. If the scale-spectra were significantly different then it can be determined that the algorithm has at least some potential as a camouflage recognition algorithm, and that further research may be warranted. Following this, all of the research’s experimentation was completed. The next chapter focuses on the results and discussions arising from the experiments described in this chapter.
Chapter 4: Results and Discussion

This chapter will report, analyze, and interpret results from experiments introduced in the previous chapter. This chapter will open with a discussion of the design, development, and implementation of the computer code that was written during research. This discussion focuses on the preprocessing function and scale-spectra generating function, and explains how they were created and tested. Following this discussion, the chapter details experiments designed to test assumptions made earlier in research. These experiments demonstrated that transition probability estimations were normally distributed and demonstrated equivalence between a random walk and many random steps. With earlier assumptions justified, the discussion will turn to the difference metrics used for fingerprint comparisons and the optimization of these difference metrics. This optimization included the development of a hybrid difference metric and a fine-tuning of scale-spectra generating function parameters. Parameters that were adjusted included the step-sizes examined, the step-size increment, and the number of random steps at each step-size. Following optimization, the fingerprint recognition algorithm was applied to the FVC 2000 databases according to FVC 2000 rules [2]. This experiment effectively recreated FVC 2000 as if the current algorithm were an entrant. Following this, the discussion will turn to integral geometry and its ability to explain the mechanism by which the algorithm operates. The chapter will conclude with experiments designed to qualify the algorithm’s applicability to other pattern recognition problems. Specifically, cloud and precipitation particle recognition and camouflage recognition.
4.1: Designing, Developing, and Implementing the Computer Code

Before any computer code was written, it was necessary to select a software engineering model to guide the design and development of the computer code. The “Waterfall Software Engineering Method” was selected for this research [54]. The Waterfall Software Engineering Method is characterized by its philosophy of defining software requirements in great detail. The advantage of this approach was that the software was thoroughly planned before any code was written. This minimized programming time by identifying logical inconsistencies early in development. The first step in the Waterfall Software Engineering method was to enumerate the software’s requirements. These requirements were defined as follows:

1) Integrated sub-modules to facilitate evaluation of algorithm performance
   a. Minimized operator involvement, decreased experimental turnaround time

2) Operational speed needed to be “fast enough”
   a. Met time requirements set forth in previous FVC competitions
   b. Allowed a sufficient number of experiments to be performed during research

3) Programmed sub-modules required robustness
   a. Easily adjustable program variables eased experimentation

4) Ability to read, process, and write a variety of image files

5) Ability to produce processed data (minimized manual data processing)
With the software requirements defined, the next phase of the Waterfall Software Engineering Method was initiated – the development of software specifications. Specifications are more detailed iterations of the software requirements. Specifications included program specifics such as “Software must be able to read and write .tiff and .bmp image formats.” After the specifications were enumerated, a design document was created. This document presented a detailed description of how the software operated and how the various sub-modules communicated with each other. This level of detail eased sub-module development – assembling the various sub-modules would have been unnecessarily difficult if the details of inter-module communication were not clearly established prior to programming. Following the design document, the next step in the Waterfall Software Engineering Method was to begin programming. Before programming could begin, a programming language had to be selected.

The programming languages considered for this research were C and Python, or some combination thereof. The C language is well-suited for computationally intensive tasks such as scale-spectra generation, while Python is ill-suited for these types of number crunching tasks. In Python’s favor were its vast library of freely available extensions, its robust and easily understood file handling functions, and its relatively succinct and easily debugged code. The decision was made to write the less computationally intensive sub-modules using Python and write the more computationally intensive sub-modules using a more appropriate language. There were six sub-modules created during research; these sub-modules and the order in which they operated are presented in Figure 4-1.
Figure 4-1: Six sub-modules designed and written during research

The first sub-module was the preprocessing function, which was discussed in the previous chapter. The preprocessing function was computationally intensive, but fingerprints only needed to be preprocessed once. After a satisfactory set of preprocessing parameters
were identified (discussed in more detail later), all fingerprints could be preprocessed and the preprocessing function was not needed anymore. Because the pre-processing function would be used sparingly, it was decided that the benefits of using Python (easier sub-module development) outweighed the drawbacks (slower processing time). Therefore, the preprocessing function was written in Python. The pre-processing function code is presented in its entirety as Appendix A. Following the preprocessing function, the preprocessed fingerprints were employed by the scale-spectra generating function.

The scale-spectra generating function would be used extensively during research. Any experiment designed to quantify the EER of the algorithm as a function of program parameters would require the generation of new scale-spectra. Because there would be many scale-spectra generated during experimentation, it was worthwhile to quantify the speed benefit that could be gained by switching to a language well-suited to numeric computing. To quantify this benefit, the scale-spectra generating function was written in three languages; Python, Pyrex, and C. Additionally, the processing benefit of using Psyco (a just-in-time compiler for Python) was quantified as well. The normalized processing time required to transform a preprocessed fingerprint into its corresponding scale-spectra is presented in Figure 4-2.
As can be seen from Figure 4-2, native Python code was ten times slower than native C code. Psyco, the just-in-time compiler for Python, functioned by running several instances of Python at once to improve processing speed. Psyco provided a significant performance benefit, and was almost three times as fast as native Python. Psyco had the advantage of being extremely easy to implement, only two lines of code were necessary. Pyrex provided a greater speed benefit than Psyco, but required more effort to implement. Pyrex was a language designed to write extension modules for Python, Pyrex is basically Python with C data types [52]. Pyrex was five times faster than native Python, and nearly twice as fast as Psyco. The standard against which these codes were judged was native C code. Native C code was twice as fast as Pyrex. After analyzing the above data, it was decided to utilize Pyrex for the scale-spectra generating function.
Pyrex provided a significant performance increase over native Python, but was relatively
difficult to implement. Significant effort was required to facilitate communication between the
Python code and the Pyrex code. The Pyrex code was written as a text file, and compiled using
the Microsoft Visual Studio .NET 2003 C compiler. This compiled Pyrex code was then called
from Python code. The Pyrex code handled scale-spectra generation, which was the most
computationally demanding portion of the software. The scale-spectra generating function is
presented in its entirety as Appendix B.

Following scale-spectra generation, the scale-spectra were stored as text files. These
text files were known as templates. The sub-module that wrote templates created a separate
template for each fingerprint. Templates contained some basic information about the file and a
text table that listed the estimated transition probabilities at the various step-sizes examined.
The sub-module that created and stored templates was written in Python. This code is
presented in its entirety as Appendix C. Following this, the templates were used by functions
designed to calculate the genuine matching scores impostor matching scores.

The genuine and impostor matching scores were calculated as described in Chapter
3.8.1. Note that calculation of the impostor and genuine matching scores required that the
templates had the same beginning step-size, ending step-size, step-size increment, and
preferably same number of random steps occurring at each step-size. The sub-modules to
calculate the genuine and impostor matching scores were written in Python. The code for these
sub-modules is presented in their entirety as Appendix D. The genuine and impostor matching
scores were stored in a text file, which was utilized by a sub-module designed to quantify the
FNMR, FMR, and EER.
The FNMR, FMR, and EER were calculated as described in the FVC 2000 report [2]. The module which calculated these metrics was written in Python as well. After these modules were written, all of the computer programming was complete. The next two sections will provide a detailed description of how the major program sub-modules (preprocessing function an scale-spectra generating function) operated as a function of their operating parameters.

4.1.1: Fingerprint Preprocessing Function

The fingerprint preprocessing function was designed to transform grayscale fingerprint images into binary (black and white) images while simultaneously masking extraneous areas of the fingerprint. As described in Section 3.3, the fingerprint preprocessing function required four parameters for operation; a neighborhood size N, a neighborhood variability threshold \( \sigma_{GVt} \), and upper and lower thresholds for black and white pixel intensity, \( I_{Bt} \) and \( I_{Wt} \), respectively. The fingerprint preprocessing function was applied to FVC 2002 DB1A to qualify how function parameters affected the quality of preprocessed fingerprints. These parameters were adjusted manually, and the “quality” of the preprocessed fingerprint was judged by the eye of the author. A manual approach was utilized because mathematically describing the quality of a preprocessed fingerprint would have been an overwhelming task, and was outside the scope of this research.

The first parameter adjusted was the neighborhood size, represented by the variable N. Recall that the pixel neighborhood was an NxN square centered on the pixel under examination, and that N was an odd integer. If N was too small and the pixel neighborhood was centered on a ridge or valley, then the pixel neighborhood would be filled with mostly black or
white pixels. This had the effect of producing a low neighborhood variability (represented by $\sigma_{GV}$), which would cause the middle of the ridges and valleys to be recognized as extraneous information. This situation is illustrated in Figure 4-3.

Figure 4-3: Fingerprint preprocessing function output when N was too small

Of course, the above problem could be rectified by increasing the neighborhood size, but this approach had its own drawbacks. Firstly, as the neighborhood size increased, the preprocessing function could have included more of the area surrounding the fingerprint, which incorporated extraneous information as part of the fingerprint. Also, when the neighborhood size was increased, the border region that surrounded the fingerprint became thicker, which masked some of the useful area of the fingerprint. This situation is illustrated in Figure 4-4.
The next fingerprint preprocessing function parameter adjusted was the neighborhood variability threshold, represented by the variable $\sigma_{GVt}$. Recall that the variability in the pixel neighborhood was compared to the threshold, and if the variability were smaller than the threshold, then the region encompassed by the pixel neighborhood was considered homogenous. In this case, the pixel under examination was designated as extraneous information. Thus, if $\sigma_{GVt}$ was too large, significant swathes of the fingerprint would be designated as extraneous information. This situation is illustrated in Figure 4-5.
Conversely, if the threshold $\sigma_{GVt}$ were too small, then relatively inhomogeneous areas of the image would have been recognized as part of the fingerprint. This would result in smudges, smears, and areas outside of the fingerprint being recognized as part of the fingerprint. This is illustrated in Figure 4-6.
The final parameters adjusted were the black and white pixel intensity thresholds, $I_{bt}$ and $I_{wt}$, respectively. These thresholds ensured that a low intensity pixel would be classified as black and a high intensity pixel would be classified as white, regardless of a pixel’s intensity relative to its neighborhood average intensity. If these parameters were not used, the pixels in the middle of ridges or valleys would sometimes be classified as the wrong color. If these parameters were poorly chosen, relatively dark or light areas may be incorrectly classified as black or white, respectively. This is illustrated in Figure 4-7.

Figure 4-7: Fingerprint preprocessing function output if thresholds were not used

Following the experiments described above, all of the fingerprint preprocessing function’s parameters were set. A satisfactory preprocessing occurred when the four parameters had the following values; a neighborhood size of 21 pixels (approximately 1050 $\mu$m), a gray-scale variability threshold of 150, and upper and lower pixel intensity limits of 245
and 20, respectively. A fingerprint transformation obtained using these parameters is presented in Figure 4-8.

![Optimized fingerprint preprocessing function output](image)

**Figure 4-8: Optimized fingerprint preprocessing function output**

### 4.1.2: Scale-Spectra Generating Function

This section will present a brief overview of the scale-spectra generating function and its operation. As previously detailed, the scale-spectra generating function estimated the transition probabilities that quantified the pixel to pixel color transitions encountered during a random walk across a binary fingerprint image. The scale-spectra generating function had several parameters that affected its operation: the beginning step-size, the ending step-size, the step-size increment, and the number of random steps occurring at each step-size. Adjusting these parameters affected the performance (and operational speed) of the fingerprint recognition algorithm.
For exploratory experiments designed to debug the scale-spectra generating function, the algorithm parameters were adjusted as follows; a starting step-size of 0\(\mu\)m, an ending step-size of 990.6 \(\mu\)m, a step-size increment of 25.4 \(\mu\)m, and 5,000 random steps at each step-size examined. Recall that the four basic transition probabilities estimated by the scale-spectra generating function were; \(P_{BB}\), \(P_{WW}\), \(P_{BW}\), and \(P_{WB}\). These four basic transition probabilities as a function of the step-size are presented in Figure 4-9. Additionally, these four transition probabilities were transformed into the four invariants as described in the Experimental chapter. The invariant scale-spectra are also presented in Figure 4-10.

**Figure 4-9: Estimated transition probabilities for F*61_5 from FVC 2002 DB1A**

Error bars represent 95% confidence interval
Recall that the fingerprint recognition algorithm operates by tabulating differences between the scale-spectra from different fingerprints. To illustrate the scale-spectra differences between two fingerprints, two determinant scale-spectra for different fingerprints are presented in Figure 4-11.
As can be seen in Figure 4-11, sometimes the scale-spectra of different fingerprints may fall within the error of one another, while sometimes the scale-spectra are significantly different. The efficacy of identifying fingerprints by their scale-spectra will be detailed later in this chapter. After the preprocessing function and scale-spectra generating functions were debugged, it was possible to test assumptions made earlier in research.
4.2:  Justification of Assumptions

Section 3.5 detailed several assumptions made early in research. The following sections will detail results from the experiments designed to test these assumptions. Recall that the two assumptions were:

1) Errors associated with estimated transition probabilities were normally distributed
2) An N-step random walk was equivalent to N iterations of a random step

Results demonstrated that these assumptions were justified. The justification of these assumptions carried significant consequences, notably:

1) The error associated with the estimated transition probabilities could be quantified
2) The algorithm was amenable to parallel processing

Regarding the first consequence, the magnitude of scale-spectra error could be quantified as a function of the number of random steps employed to estimate the transition probabilities.

Regarding the second consequence, transforming an inherently serial process (a random walk) into a parallel process (a random step) allowed research to bring increased computational power to bear on the fingerprint recognition algorithm.

4.2.1: Demonstrating that Estimations Were Normally Distributed

As discussed in Section 3.7, it was assumed that the errors associated with the estimated transition probabilities approximated via random steps were normally distributed. To test this hypothesis, the white-white transition probability, $P_{WW}$, of a single fingerprint ($F_{102_1}^*$ from 2000 DB1B) was estimated 10,000 times. Each estimation utilized 10,000 random steps at
a scale of 280 μm. The probability distribution of $P_{WW}$ was graphed alongside a normal distribution with mean 0.2801 and standard deviation $4.5 \times 10^{-3}$. This graph appears as Figure 4-12.

![Figure 4-12: CDF of estimated transition probabilities $P_{WW}$ estimated via $10^4$ random steps. $P_{WW}$ was estimated $10^4$ times.](image)

The experimentally tabulated Cumulative Distribution Function (CDF) for $P_{WW}$ and the normal distribution were closely matched. Thus, it was experimentally demonstrated that the error associated with the transition probabilities were approximately normally distributed. As a consequence, the error equations developed in Section 3.7 can be utilized to determine the confidence interval on transition probability measurements.

The error associated with the estimated transition probabilities was plotted as a function of the number of random steps. This calculation was based on the standard deviation
of the estimated transition probabilities, in accordance with Equation 3-21. This graph is presented as Figure 4-13.

![Graph showing transition probability estimate error as a function of random steps.](image)

**Figure 4-13: Transition probability estimate error as a function of random steps**

### 4.2.2: Demonstrating Equivalence between Random Walk a Random Steps

In Section 3.5.2, it was assumed that the transition probabilities estimated via N iterations of a random step were equivalent to the transition probabilities resulting from an N-step random walk. This assumption was tested by performing several N-step random walks and several N-iteration random steps on the same fingerprint, and comparing the scale-spectra. The fingerprint used in this analysis was F*$_{103\_1}$ from FVC2002 DB1B. For the real random walk (RRW), the beginning step-size was 0.0 µm, the ending step-size was 2032 µm, the step-size increment was 12.7 µm, and there were $10^6$ steps at each step-size. The parameters for the
random steps were the same as for the RRW – there were $10^6$ iterations of a random step at each step-size. The determinant scale-spectra and trace scale-spectra for the invariants estimated via a random walk and random step are presented in Figure 4-14. The Eigen Value scale-spectra for the invariants estimated via a random walk and random steps are presented in Figure 4-15.

Figure 4-14: Determinant and trace scale-spectra
There were $10^6$ steps at each step-size examined
Note that the four invariant scale-spectra overlaid one another almost exactly at every step-size except zero. The reason for this was straightforward. In a real random walk with a step-size of zero, the random walk never left the point at which it started because the step-size was zero. Thus, if a real random walk started on a black pixel, $P_{BB}$ is one and $P_{WW}$ is zero. Alternatively, a new starting point was chosen for all iterations of the random step. For the case when the step-size was zero, many random steps produced a ratio ($P_{BB}$ to $P_{WW}$) that was simply an estimation of the ratio of black to white pixels present in the preprocessed fingerprint image. Thus, it was experimentally demonstrated that at all scales except 0.0 $\mu$m, a random step estimated transition probabilities that were equivalent to those estimated using a random walk. As a consequence, it would be possible to utilize parallel computing when transforming binary images into their corresponding scale-spectra.
4.3: Metrics of Fingerprint Comparison

Section 3.6 described the metrics of fingerprint comparison. Applying any of these metrics to scale-spectra yielded a unit less scalar. It was hypothesized that fingerprints that were similar would yield smaller differences, and that fingerprints that were dissimilar would yield larger differences. To illustrate, consider two determinant scale-spectra. The two scale-spectra were associated with fingerprints $F_{*61.5}$ and $F_{*85.2}$, both of which were from FVC2002 DB1A. The difference between these two scale-spectra was proportionate to the area between the two curves. This was illustrated previously in Equation 3-12.

The five basic difference metrics were the trace, determinant, two Eigenvalues, and hybrid metric. A goal of this research was to quantify the efficacy of these metrics. Metric efficacy depended on the four parameters of the scale-spectra generating function. Before describing how the scale-spectra generating function parameters affected algorithm performance, it will be necessary to describe how the efficacy of the difference metrics was quantified.

4.4: Quantifying the Efficacy of Difference Metrics

As mentioned in Chapter 3, the single most important metric of fingerprint identification algorithm performance was the Equal Error Rate (EER). The EER was defined as the value where the False Non-Match Rate (FNMR) equaled the False Match Rate (FMR). To quantify the FNMR and FMR required the calculation of genuine matching scores and impostor matching scores, as described in Section 3.8.1. The next two sections will illustrate how the genuine and impostor
matching scores were calculated, and how these scores were utilized to calculate the FNMR, FMR, and EER.

### 4.4.1: Calculating Impostor and Genuine Matching Scores

Recall that the impostor and genuine matching scores were the scalar values (obtained via difference metrics) that resulted from comparing the scale-spectra of two fingerprints. Genuine matching scores resulted from the comparison of fingerprints from the same finger. Accordingly, impostor matching scores resulted from the comparison of fingerprints from different fingers. The calculation of the genuine and impostor matching scores will be illustrated using FVC2000 DB1A. All fingerprints from this database were preprocessed and transformed into their scale-spectra (templates) using the scale-spectra generation function. The parameters used during scale-spectra generation were as follows; starting step-size of 0.0 μm, and ending step-size of 1,000 μm, a step-size increment of 25 μm, and 2,000 random steps at each step-size examined. The genuine and impostor matching scores were calculated as described in Section 3.8.1. There were 2800 genuine matching scores and 4950 impostor matching scores. These scores were separated into bins so they could be graphed in histogram form, as illustrated in Figure 4-16.
Figure 4-16: Impostor and genuine matching scores for DB1A of FVC2002

The histograms for the genuine and impostor matching scores displayed some overlap. The amount of overlap is a measure of difference metric efficacy. Better performing difference metrics displayed less overlap. A difference metric with an EER of zero would produce genuine and impostor matching score histograms with zero overlap. Next, the method by which genuine and impostor matching scores were used to calculate the FNMR, FMR, and EER will be detailed.
4.4.2: Calculating the FMR, FNMR, and EER

The genuine and impostor matching scores were compared to the threshold, a variable that could take any value between zero and the maximum genuine or impostor matching score. If a genuine matching score were greater than the threshold, then the algorithm had incorrectly predicted that two fingerprints were not from the same finger; a false non-match. The number of false non-matches that occurred divided by the total number of genuine matching scores (2800) was the FNMR. Likewise, if an impostor matching score were less than the threshold, then the algorithm has incorrectly predicted that two fingerprints were from the same finger; a false-match. The number of false matches that occurred divided by the total number of impostor matching scores (4950) was the FMR. The FNMR and FMR both depended upon the value of the threshold. By varying the threshold between zero and one, the FNMR vs FMR curve was obtained, as illustrated in Figure 4-17.
Figure 4-17: FNMR vs FMR of four difference metrics

Note that at some point the value of the FNMR equaled the value of the FMR, this was the EER. For this example it was observed that the better difference metrics (based on the second Eigenvalue, trace, and the determinant) had EERs under 20%. However, recall that the EER was also a function of scale-spectra generating function parameters. The experiments designed to optimize the EER of the difference metrics by adjusting scale-spectra generating function parameters will be discussed in the next section.
4.5: Optimizing the Difference Metrics

The single most useful metric of fingerprint identification algorithm performance was the EER. The EER was defined as the point at which the FNMR equals the FMR. The EER is calculated using the genuine matching scores and impostor matching scores, as previously described. The EER will be affected by the parameters that control the scale-spectra generating function; the starting step-size, the ending step-size, the step-size increment, and the number of iterations that occur at each size. Thus, the effectiveness of the difference metrics was optimized by observing how these function parameters affected the EER. The following sections will detail the results of experiments described in Section 3.9, which were designed to optimize the parameters utilized by the scale-spectra generating function and select the optimal difference metric.

4.5.1: Quantifying the EER as a Function of Step-Sizes Examined

The goal of the first experiment was to quantify the performance of the fingerprint identification algorithm as function of the largest step-size employed during the random steps. The 800 fingerprints in FVC 2002 DB1A were used for this experiment. The other scale-spectra generation parameters were held constant for this experiment, a starting step-size of 0.0 μm, a step-size increment of 12.7 μm, and 5,000 random steps at each step-size examined. The results of this experiment are presented in Figure 4-18.
The x-axis represents the largest step-size included in the calculation of the EER. For example, at an x-axis value of 250 μm, the EER was calculated using all transition probabilities associated with step-sizes between 0 μm and 250 μm. Similarly, the EER at an x-axis value of 1500 μm was calculated using the transition probabilities associated with all step-sizes between 0 μm and 1500 μm. Thus, if the slopes of the curves in Figure 4-18 were negative, the EER was decreasing and the inclusion of that step size was beneficial for algorithm performance. Conversely, if the slopes were positive, the EER was increasing and the inclusion of that step-size was deleterious to algorithm performance. Examining Figure 4-18, it was observed that the first Eigenvalue is a poor metric for matching fingerprints, regardless of the step-sizes under consideration. For the determinant and second Eigenvalue, scales up to 1,000 μm have no
effect or improve the EER, while scales above 1000 µm offer no improvement or worsen the EER. For the trace, scales less than 150 µm are useful, as are scales between 500 µm and 600 µm, while other scales offer no benefit.

The next experiment focused on determining what scale-spectra invariant yielded the lowest EER at any one step-size. The EER was calculated using only one step size, the beginning step-size was the same as the ending step-size. This experiment was performed for all step-sizes from 0 µm to 1000 µm, by increments of 25 µm. The number of random steps at each steps-size was held constant at 5,000. The 800 fingerprints in FVC 2002 DB1A were used for this experiment. The results of this experiment are presented in Figure 4-19.
Examining these data, it was observed that matching fingerprints based on the value of the trace yielded the lowest EER at step-sizes less than approximately 185 µm. At step-sizes greater than 185 µm, the determinant and second Eigen-value yielded similar EERs.

The data in Figure 4-19 showed that the EER oscillated as a function of step-size. This oscillatory behavior mirrored the oscillatory behavior exhibited by the transition probabilities themselves (see Figure 4-9). This strange behavior was unexplained for much of the research. However, eventually the application of integral geometry was able to shed some light on this behavior (discussed in more detail in Section 4.8). Moving past the oscillatory behavior of the EER as a function of step-size, the beneficial step-sizes have been identified and the optimal difference metric at any given step-size was determined. The next experiment was designed to develop a hybrid metric that utilized this information to produce a superior metric.

**4.5.2: Developing the Hybrid Difference Metric**

Using information from the previous section, the hybrid metric (abbreviated Met) was created. The hybrid metric used both the trace and determinant as a comparison metric. The formulae for the hybrid metric are:

\[
\text{Met} = \text{Trace} \quad \text{(Step-size} \leq 185 \mu m) \tag{4-1}
\]

\[
\text{Met} = \omega \times \text{Determinant} \quad \text{(Step-size} > 185 \mu m) \tag{4-2}
\]

In equation 4-2, \( \omega \) represented a weighting factor. Recall that the trace was a sum of two numbers between zero and one, while the determinant was the difference of two products of
numbers between zero and one. In practice, this meant that the difference between the traces of two fingerprints was, on average, about six hundred times larger than the difference between the determinants of two fingerprints. Therefore, a weighting factor was necessary to prevent the contribution of the trace from overpowering the determinant. The optimal weighting factor was determined by graphing the EER as a function of the weighting factor, \( \omega \). The 800 fingerprints in FVC 2002 DB1A were used for this experiment. The scale-spectra generation function parameters were adjusted as follows, a beginning step-size of 0.0 \( \mu \text{m} \), an ending step-size of 1000 \( \mu \text{m} \), a step-size increment of 25 \( \mu \text{m} \), and 2,000 random steps at each step-size examined. The results of this experiment are presented below in Figure 4-20.

![Graph showing EER as a function of the weighting factor \( \omega \)](image)

**Figure 4-20: EER as a function of the weighting factor**
It was observed that the EER reached a minimum (29%) when $\omega$ had a value of twenty. Following this experiment, there were a total of five difference metrics that could be used to compare fingerprints, the four matrix invariants and a hybrid metric composed of two invariants. The next sections will describe the results of experiments designed to determine how other scale-spectra generating function parameters affected algorithm performance.

4.5.3: Quantifying the EER as a Function of Step-Size Increment

The next parameter adjusted was the step-size increment. There were six step-size increments employed; 5 $\mu$m, 10 $\mu$m, 25 $\mu$m, 51 $\mu$m, 101 $\mu$m, and 254 $\mu$m. For this experiment, the 800 fingerprints in FVC 2002 DB1A were used. The starting step-size and ending step-size were set to 0 $\mu$m and 1000 $\mu$m, respectively. At each step size there were 2,000 random steps. After calculations were completed the EER was plotted as a function of step-size increment, as shown in Figure 4-21.
As is observed in Figure 4-21, the EER calculated via the hybrid metric was much more sensitive to step-size increment than the four matrix invariants. This was because the hybrid metric was equivalent to the trace for step-sizes up to 185 μm, so if the step-size increment was too large the hybrid metric missed this information. After the above data were considered, the step-size increment was frozen at a value of 25 μm.

4.5.4: Optimizing the Number of Random Steps

Following the previous experiments, it was demonstrated that step-sizes from 0 μm to 1000 μm were beneficial for algorithm performance, and a hybrid metric was developed. Therefore, the only remaining parameter to adjust was the number of iterations that occurred
at each step-size. Therefore, the next experiment determined how the number of iterations employed at each step-size affected the EER. For this experiment, the 800 fingerprints in FVC 2002 DB1A were used. The starting step-size and ending step-size were set to 0 μm and 1000 μm, respectively. The step-size increment was set at 25 μm. The EER was plotted as a function of number of iterations occurring at each step-size, as shown in Figure 4-22.

![Figure 4-22: EER as a function of random steps](image)

As previously discussed, the calculated matrix invariants contained less error as the number of iterations increased. However, the drive for increased accuracy needed to be balanced with computational realities. Increasing the number of random steps occurring at each step-size would slow the algorithm. After examining the data in Figure 4-22, it was decided
to set the number of iterations occurring at each step-size at 2,000. Following this experiment, all parameters of the scale-spectra generating function were optimized. The efficacy of the optimized algorithm was quantified by recreating the Year 2000 Fingerprint Verification Competition.

4.6: Recreating FVC2000

The quantification of algorithm performance culminated in a recreation of the FVC2000 competition. The fingerprint identification algorithm was applied to all four A-set fingerprint databases from FVC 2000 [2]. The preprocessed fingerprints were transformed into their scale-spectra via the previously described preprocessing and scale-spectra generating functions. The scale-spectra generating function parameters were set at their previously optimized values:

1.) A starting step-size of 0 $\mu$m
2.) An ending step-size of 1000 $\mu$m
3.) A step-size increment of 25 $\mu$m
4.) 2,000 random steps performed at each step-size

Thus, there were a total of 40 Monte Carlo Integrations employing 2,000 random steps, for a total of 80,000 random steps per fingerprint. After the transition probabilities of the fingerprints were estimated, the genuine matching scores and impostor matching scores were calculated, as described in Section 4.4.1. These scores were used to calculate the FNMR, FMR, and EER, as described in Section 4.4.2. The FNMR vs. FMR curves for 2000 DB1A are presented in Figure 4-23.
Two surprising results were observed; the best EERs were significantly lower for the FVC2000 DB1A as compared with FVC2002 DB1A, and the hybrid metric performed more poorly than both the determinant and trace of which it was composed. Data suggested that three matrix invariants (the trace, determinant, and second Eigen-value) yielded similar EERs of approximately 19% for FVC 2000 DB1A, whereas the same invariants yielded EERs of 32% to 37% for FVC 2002 DB1A. Examining the fingerprints from each set visually, there were no obvious differences. The other surprising result was that the hybrid metric performed more poorly than the two invariants of which it was composed. The determinant and trace yielded EERs near 19%, while the hybrid metric yielded an EER of only 27%. This suggested that the Met
may have been over fit to the fingerprints presented in FVC 2002 DB1A. In an attempt to elucidate the differences between a relatively effective difference metric (the Determinant) and a relatively ineffective difference metric (the hybrid metric), the genuine matching score and imposter matching score histograms were plotted. These appear in Figure 4-24 and Figure 4-25.

![Figure 4-24: Genuine and impostor matching scores for determinant metric](image-url)
Examining Figures 4-24 and 4-25, it was clear that there was more separation between the genuine matching scores and impostor matching scores for the difference metric based on the determinant. For the hybrid metric, the genuine matching scores and impostor matching scores were more closely aligned. It was clear that as the genuine matching scores and impostor matching scores were more closely aligned, it was more difficult to identify fingerprints correctly. A more extreme case of confounded genuine and impostor matching scores should be observed on worse performing metrics, such as the first Eigenvalue. These data are presented in Figure 4-26.
As observed in Figure 4-26, the histogram of genuine matching scores lies completely within the histogram of impostor matching scores. Obviously the difference metric based on the first Eigenvalue was ineffective at resolving differences between genuine and impostor fingerprints.

The above experiments and analyses were performed for all four of the A set databases used during the FVC2000 competition. In addition to the EER, the times required for enrolling and matching fingerprints were recorded as well. These results are summarized in Table 4-1.
Table 4-1: Performance of algorithm in simulated FVC2000 competition

<table>
<thead>
<tr>
<th>Database</th>
<th>Time (seconds/print)</th>
<th>EER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Enroll</td>
<td>GMS</td>
</tr>
<tr>
<td>2000 DB1A</td>
<td>10.93</td>
<td>0.010</td>
</tr>
<tr>
<td>2000 DB2A</td>
<td>10.93</td>
<td>0.010</td>
</tr>
<tr>
<td>2000 DB3A</td>
<td>14.67</td>
<td>0.010</td>
</tr>
<tr>
<td>2000 DB4A</td>
<td>11.38</td>
<td>0.010</td>
</tr>
</tbody>
</table>

From Table 4-1, it was observed that the determinant was the best all-around metric of fingerprint identification, producing an average EER (across all databases) of 19.8%. The second Eigenvalue was close behind, with an average EER of 20.1%. The custom metric, which was developed using FVC2002 DB1A, performed relatively poorly, with an average EER of 29.3%. The reason that these metrics performed differently across the various databases was not known. However, as will be discussed later, it was hypothesized that the algorithm operated chiefly by measuring the ridge width of fingerprints. Thus, it was hypothesized that if the average ridge widths for all fingerprints were quantified, databases for which these performance metrics performed poorly would contain fingerprints with more homogenous ridge widths.

Moving onto performance metrics of a temporal nature, the average enroll time was 11.98 seconds. Approximately 60% of the time (7.2 seconds) was required to preprocess the fingerprint, and 40% of the time (4.8 seconds) was required to transform the preprocessed
fingerprint into its corresponding scale-spectra. The enroll time was fairly constant for three of the four databases, but was slightly longer for DB3A. This was attributed to the nature of fingerprint images in DB3. In DB3, the area of the fingerprint was small compared to the green mask that surrounded the fingerprint. This is illustrated graphically in Figure 4-27, which shows an average fingerprint from DB1A next to an average fingerprint from DB3A.

![Figure 4-27: F*_{66,2} from DB1A (left-side) compared to F*_{54,2} from DB3A (right-side)](image)

The larger masked area meant that more random steps were “thrown away” during the scale-spectra generation function. Because more random steps were ignored, the scale-spectra generating function took longer to accumulate the desired number of random steps. Finally, note that the preprocessing algorithm was not optimized; it was written in Python and there was potential for significant time savings in the preprocessing sub-module. The next section will compare and contrast the performance above with other algorithms that entered the FVC2000 competition.
4.7: **Comparison with Other FVC2000 Entrants**

The algorithm discussed herein was compared with other algorithms which entered FVC 2000. The results presented previously alongside the results from FVC 2000 participants are presented in Table 4-2.

**Table 4-2: Comparison between FVC2000 entrants and algorithm discussed herein (Some results reproduced from FVC 2000 report [2])**

<table>
<thead>
<tr>
<th>Name</th>
<th>EER</th>
<th>Enroll</th>
<th>Match</th>
<th>Time (seconds/print)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sag1</td>
<td>1.73%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>3.18 1.22</td>
</tr>
<tr>
<td>Sag2</td>
<td>2.28%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>1.11 1.11</td>
</tr>
<tr>
<td>Cspn</td>
<td>5.19%</td>
<td>0.14%</td>
<td>0.31%</td>
<td>0.2 0.20</td>
</tr>
<tr>
<td>Cetp</td>
<td>6.32%</td>
<td>0.00%</td>
<td>0.02%</td>
<td>0.95 1.06</td>
</tr>
<tr>
<td>Cwai</td>
<td>7.08%</td>
<td>4.46%</td>
<td>3.14%</td>
<td>0.27 0.35</td>
</tr>
<tr>
<td>Krdl</td>
<td>10.94%</td>
<td>6.86%</td>
<td>6.52%</td>
<td>1.08 1.58</td>
</tr>
<tr>
<td>Utwe</td>
<td>15.24%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>10.42 2.67</td>
</tr>
<tr>
<td>Fpin</td>
<td>15.94%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>1.22 1.27</td>
</tr>
<tr>
<td>Uinh</td>
<td>19.33%</td>
<td>3.75%</td>
<td>5.23%</td>
<td>0.71 0.76</td>
</tr>
<tr>
<td>Stoffa</td>
<td>19.80%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>11.98 0.01</td>
</tr>
<tr>
<td>Diti</td>
<td>20.97%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>1.24 1.32</td>
</tr>
<tr>
<td>Ncmi</td>
<td>47.84%</td>
<td>0.00%</td>
<td>0.09%</td>
<td>1.44 1.71</td>
</tr>
</tbody>
</table>
One note before proceeding – as described in Section 3.2, the computer used for research was very similar to, but not exactly the same, as the computer used for FVC 2000. As such, the temporal metrics (time to enroll and time to match) were not directly comparable.

The algorithm developed by this research would have performed relatively poorly in the competition. The EER was higher than that achieved by other algorithms. The bright spots for the algorithm were its lack of rejection events and its ultra-fast matching times. Because the algorithm calculated a set of transition probabilities for a given image regardless of what the image was, a fingerprint never failed to be enrolled. This contrasted with other algorithms that may be looking for minutiae, and may fail to enroll a fingerprint if they cannot find the minutiae. Likewise, the algorithm never failed to match a fingerprint, because the difference between two fingerprints was simply the difference between two arrays. Finally, because the fingerprint was represented as a small (on the order of 1,000) array of scalar values, comparisons between fingerprints were extremely fast. The average match time for the algorithm developed herein was less than 1% of the average match time of the other algorithms which entered FVC 2000.

There were several pros and cons unique to the algorithm developed during this research. A benefit to this algorithm was that it did not catalogue minutiae or require permanent storage of the original fingerprint image. Thus, even when provided the scale-spectra that represented a fingerprint, it was not-apparent that the fingerprint could have been reverse engineered from these data. This was a benefit for applications where there were privacy concerns regarding disclosure of the original image. Also, the algorithm was of a very general nature, it could have been applied to any number of pattern recognition problems. The
scale-spectra generation function could have been applied to any set of images, fingerprints, paintings, air-particles, etc. This was also somewhat of a con, as the algorithm developed herein was not adapted to process the information normally used for fingerprint identification (minutiae).

The preceding sections have described the performance of the algorithm in great detail. However, these pieces of information have not clarified the mechanism by which the algorithm operated. For this, integral geometry was employed to determine how real scale-spectra compare to scale-spectra obtained analytically using a simplified model of a fingerprint. This experiment is discussed in the next section.

4.8: Application of Integral Geometry

This experiment was designed to test the hypothesis that the fingerprint identification algorithm discussed herein identified fingerprints by their ridge width. This hypothesis was tested using integral geometry and a simplified fingerprint model. For this experiment, Fingerprint F*14_5 from DB1A of FVC 2002 was utilized. This fingerprint was chosen because it had ridges and valleys (black and white stripes) of approximately equal width, as confirmed by its ratio of black to white pixels (49.9% black and 50.1% white). A cursory examination of the print suggested that the ridges and valleys were approximately five pixels, or 250 μm, in width. A preprocessed version of this fingerprint is reproduced in Figure 4-28.
Fingerprint $F_{14.5}^*$ was modeled as a set of parallel black and white stripes of equal width (5 $\mu$m). Thus, the integral geometry equations developed to solve the problem of “Buffon’s Needle” in Section 3.11 were applied to analytically obtain scale-spectra for the simplified fingerprint model. The “needle lengths” (step-sizes) under consideration were limited to less than 250 $\mu$m because Equation 3-36 and Equation 3-37 were valid only when the “needle lengths” (step-sizes) were less than the width of the “wooden planks” (ridges and valleys). These analytically obtained scale-spectra invariants were compared to the experimentally obtained scale-spectra for $F_{14.5}^*$. This graph is presented in Figure 4-29.
The calculated scale-spectra were fairly close to the experimentally obtained scale-spectra below 200 μm. Recalling Figure 4-19, the difference metric based on the trace produced its best-demonstrated EER at step-sizes less than 200 μm. It seemed fair to conclude that at step-sizes less than the ridge width, the simplified fingerprint model was a good approximation of the real fingerprint. This suggested that at lower step-sizes, the scale-spectra were a measurement of ridge width. These conclusions obtained via integral geometry were the first clues as to how the algorithm described herein operated. Following this experiment, all research into utilizing this algorithm for fingerprint recognition was complete. The next sections will focus on the results of experiments that tested the algorithm’s potential for cloud and precipitation particle and camouflage recognition.
4.9: Other Areas of Applicability

This section will investigate other areas of applicability for the pattern recognition algorithm described herein. Specifically, this section will investigate the algorithm’s applicability as a precipitation particle recognition algorithm and as a camouflage recognition algorithm. The results discussed in the previous sections suggested that the algorithm experienced difficulty discerning between patterns that were similar on a global scale, so it was hypothesized that the algorithm may be more applicable to classifying patterns which were not similar on a global scale. The experiments that were designed to test this hypothesis are discussed in the following sections.

4.9.1: Cloud and Precipitation Particle Recognition

The U.S. Navy has solicited the atmospheric research community for new image recognition techniques to group cloud and precipitation particles into the basic meteorological groups [48]. The desired attributes for this image recognition technique were sensor independence and fast operation. These requirements have arisen because new imaging probes have been producing enormous amounts of data that the current recognition techniques have been ill-suited to handle [45]. It was hypothesized the current algorithm would be well-suited because it should:

1.) Operate well across many resolutions (suggesting sensor independence)

2.) Allow for fast enrollment and matching
The scale-spectra created by the algorithm described herein could be quickly produced and matched if the parameters used during preprocessing and scale-spectra generation were carefully controlled. A small exploratory trial was initiated to yield a gross assessment of algorithm’s potential as a cloud and precipitation particle recognition algorithm.

To illustrate how the algorithm applied to cloud and precipitation particle images, the scale-spectra generating function was applied to the three particle images shown in Figure 4-30 [45]. There were 121 step-sizes examined, from 0 μm to 231 μm. There were 200,000 random steps used to estimate the transition probabilities at each step-size examined. The determinant scale-spectra for the three particles appear in Figure 4-31.

![Particle Images](image)

Figure 4-30: From left to right, Particle 1, Particle 2, and Particle 3 Captured using a 2D-S probe, 10 μm resolution [45]
As illustrated in Figure 4-31, different particles exhibited different determinant scale-spectra. These determinant scale-spectra displayed different derivatives, minima and maxima, and inflection points. It therefore seemed plausible that different classes of particles could be recognized by their scale-spectra. In addition to particle classification, scale-spectra may have been able to reveal particle size. To illustrate, an image of a particle was compared to a scaled version of itself. The scaled version of the image was 150% of the size of the original image. The original and scaled versions of the particle image are presented in Figure 4-32 [45]. The determinant scale-spectra for these two particles are presented in Figure 4-33.
Figure 4-32: Particle 1 (left) and Particle 1a (right)  
Particle 1a is 150% the size of Particle 1 [45]

Figure 4-33: Determinant scale-spectra Particle 1 and Particle 1a  
Error bars represent 95% confidence interval, $2\times10^5$ random steps at each step-size
As illustrated in Figure 4-33, the two particles yielded determinant scale-spectra that were qualitatively indistinguishable but quantitatively unique. For example, Particle 1 displayed a local minimum at a step-size of 62 μm (determinant value of 0.007) and a local maximum at a step-size of 92 μm (determinant value of 0.015). Particle 1a displayed the same local minimum at a step-size of 96 μm and the same local maximum at a step-size of 142 μm. Thus, the particle that was 50% larger displayed the same local minimum and maximum at step-sizes that were 50% larger. Ergo, by examining the step-sizes at which the identifying features of the scale-spectrum occur, it may be possible to quantify particle size.

If a software package were to appeal to the entire atmospheric research community, it must be compatible with various cloud and precipitation particle imaging probes. One area where probes differ is resolution – the size of particle detail that can be captured [45]. It was hypothesized that the algorithm described herein was mostly unaffected by resolution differences. As a rough test of this hypothesis, consider three variations of a particle image. The native image was captured at a resolution of 10 μm. The other images were less detailed copies of the first image. The first less detailed image had a resolution equivalent to 40 μm, and the second less detailed image had a resolution equivalent to 100 μm. These three images (one original and two less detailed) are presented in Figure 4-34. The determinant scale-spectra for these three images are in Figure 4-35.
Figure 4-34: From left to right, Particle 1, Particle 1b, and Particle 1c [54]
Images represent resolutions of 10 μm, 40 μm, and 100 μm respectively

Figure 4-35: Determinant scale-spectra of same particle at various resolutions
Error bars represent 95% confidence interval, 2x10^5 random steps at each step-size

The determinant scale-spectra differed slightly at smaller step-sizes, where intuitively the loss of resolution should have had the largest effect. However, at larger step-sizes, the
scale-spectra overlaid one another. The lowest resolution image (100 μm resolution) contained only 1% as much information as the most detailed image (10 μm resolution), yet their determinant scale-spectra suggested they were the same particle. Thus, the resolution of the imaging probe used by researchers may not necessarily hinder data comparison among atmospheric researchers.

Another benefit of the algorithm was independence from the orientation of the particle image. Orientation independence was significant because it reduces preprocessing requirements. Orientation independence was due to the selection of a random angle during the random steps. To illustrate orientation independence, the algorithm was applied to three images of the same particle, each with a unique angle of rotation. The three particle images are presented in Figure 4-36. Their determinant scale-spectra appear in Figure 4-37.

![Figure 4-36: From left to right, Particle 2, Particle 2a, and Particle 2b [54]](image)
Figure 4-37: Determinant scale-spectra for rotated images. Error bars represent 95% confidence interval, $2 \times 10^5$ random steps at each step-size.

Figure 4-37 illustrated the orientation independence of the pattern recognition algorithm – the determinant scale-spectra from the three rotated particles fell well within the error of one another. Thus, it may be that the algorithm described herein could be modified to create a cloud and precipitation particle recognition algorithm that may be; sensor independent, orientation independent, and able to measure particle size. This section is concluded with the statement that more research seems warranted for this application and similar pattern recognition applications.
4.9.2: Camouflage Recognition

Another potential application identified during research was camouflage recognition; the ability to identify objects which blend into their surroundings. A cursory examination of the camouflage recognition potential of the algorithm was performed by searching for a flounder hidden in a bed of gravel, illustrated in Figure 4-38. Four regions were extracted from the image that contained the flounder, one region contained a portion of the flounder, and the three remaining regions were flounder free. The native image and the regions that were extracted from the image are presented in Figure 4-38 and Figure 4-39, respectively.

Figure 4-38: A photograph of a flounder over a bed of gravel [55]
The determinant scale-spectra for the four regions are presented in Figure 4-40. The flounder free regions yielded determinant scale-spectra which overlaid one another, while the flounder containing region was distinct. Although this analysis was extremely limited, initial results suggest that a more thorough set of exploratory experiments may be warranted.
In summary, preliminary results suggested that the algorithm discussed herein may be well-suited to pattern recognition problems where recognizing global patterns is more important than recognizing differences in minutiae. It appeared that further research into the pattern recognition problems discussed above is warranted. The conclusion of the exploratory experiments described in this chapter represented the conclusion of this research effort. The next section will summarize the results presented earlier in this chapter.
4.10: Summary of Results

This chapter opened with a discussion of development and design of the computer code that was implemented for this research. Six sub-modules were conceived and then designed using the Waterfall Software Engineering Method. After the sub-modules were well-defined, they were written in either Python or Pyrex, depending on the computational requirements of the sub-module. After these program sub-modules were written, they were optimized.

The parameters that controlled the preprocessing function were adjusted so that grayscale fingerprint images could be transformed into binary images while simultaneously masking regions of the image that were not part of the fingerprint. The pre-processed images were then directed to the scale-spectra generation function. The scale-spectra generating function was written in Pyrex for computational speed. The more ancillary sub-modules were then programmed, and all of these sub-modules were used in experiments designed to justify earlier assumptions.

It was demonstrated that error associated with transition probability estimations were normally distributed. This was significant because it meant that the error associated with scale-spectra could be quantified. Next, the equivalence between a random walk and many random steps was demonstrated. This was significant because it allowed an inherently serial process to be represented as a parallel process, enabling the application of increased computational power through parallel computing. After these assumptions were justified, the chapter turned to a discussion of the difference metrics.

The scale-spectra-generating function parameters were adjusted to determine how they affected fingerprint recognition algorithm performance and speed. Difference metrics produced
quantities that other sub modules needed to calculate genuine matching scores and impostor matching scores. These scores were use to generate the FNMR and FMR curves, which yielded the EER. The EER’s relationship with the number of random steps, step-size range, step-size increment, and difference metric was illustrated. Multiple difference metrics were combined to form a hybrid metric. With the scale-spectra-generating function optimized, it was time to recreate FVC2000.

FVC2000 has become such a widely cited metric of fingerprint recognition algorithm performance that it was somewhat of a *de facto* standard. The fingerprint identification algorithm described herein was applied to FVC2000 so that the algorithm could be compared to its peers. The results suggested that the algorithm would have performed relatively poorly if it were entered in FVC2000. The poor performance of the algorithm was attributed to its inability to capture the type of fingerprint information utilized by most algorithms (minutiae). Following analysis of the FVC2000 results, integral geometry was utilized to elucidate the mechanism by which the algorithm operated. Integral geometry suggested that scale-spectra at smaller step-sizes (less than the ridge width) were well-matched with scale-spectra analytically obtained using integral geometry. These results suggested that the fingerprint identification algorithm discussed herein identified fingerprints by their ridge width.

The final sets of experiments were designed to qualify the algorithm’s performance for other pattern recognition algorithms. These experiments suggested that further research is warranted to develop the algorithm for cloud and precipitation particle recognition and camouflage recognition. This summarizes work described in this chapter.
Chapter 5: Summary

The research described herein investigated a novel Markov Chain Monte Carlo based pattern recognition algorithm. The positive attributes of the algorithm were its ultra-fast matching times, orientation independence, lack of rejection events, relative insensitivity to resolution difference, and one-way transformations. This transformation occurred via many iterations of a random step, occurring across many step-sizes, yielding an array of estimated transition probabilities known as the scale-spectra. It was shown that images could be compared by comparing their scale-spectra – similar images yielded smaller differences and dissimilar images yielded larger differences. This was demonstrated by quantifying the efficacy of the algorithm as a fingerprint recognition algorithm. The history and development of various fingerprint recognition techniques were researched. The unique place of this algorithm within the landscape of fingerprint recognition techniques was highlighted, including a discussion of the algorithm’s attributes and shortcomings. The sets of experiments that would be needed to quantify the efficacy of the algorithm’s performance as a fingerprint recognition algorithm were then discussed. Experiments were designed to test assumptions underlying the algorithm’s operation, train the algorithm, quantify fingerprint recognition efficacy, and compare experimentally obtained data to analytically calculated data.

The efficacy of the algorithm as a fingerprint recognition technique was quantified according to the rules set forth in the Year 2000 Fingerprint Verification Competition [2]. Results suggested that the algorithm was ill-suited as a fingerprint recognition technique, managing a below average EER of 19.8% across the four fingerprint databases examined. This poor performance was highlighted by examining histograms of genuine and impostor matching
scores, which illustrated that the algorithm produces genuine matching scores that are
couidned with impostor matching scores, making it difficult to differentiate between the
two. The mechanism behind this poor performance was explored using tools provided by
Integral Geometry. It was demonstrated that at small step-sizes the algorithm was an indirect
measure of ridge width. Fingerprints from different fingers can have similar ridge widths, so the
relatively poor performance of the algorithm is supported by the insights gained from integral
geometry. Furthermore, this analysis suggested that the algorithm ignored minutiae, which are
one of the richest sources of information available in a fingerprint and have been the basis of
most fingerprint identification techniques. These insights led to the conclusion that the
algorithm, at least in its current form, was ill-suited to observing minutiae but was perhaps
well-suited to observing global patterns.

The applicability of the algorithm as a cloud and precipitation particle recognition
algorithm was qualified through several experiments. These experiments showed that different
particle shapes yielded different scale-spectra, which may allow the algorithm to categorize
particles into their basic meteorological groups. Further experimentation showed that the
algorithm was relatively insensitive to resolution differences, suggesting that it may be able to
process and compare cloud and precipitation particle images captured using different probes.
Furthermore, it was also demonstrated that scale-spectra features could be correlated to
particle size, suggesting that the algorithm may be able to both categorize and measure the size
of cloud and precipitation particles. It was demonstrated that the algorithm is insensitive to
particle orientation, which decreases preprocessing requirements. On the subject of
camouflage recognition, an experiment demonstrated that the algorithm could differentiate a
camouflaged object sitting on a uniform background. The cloud and precipitation particle recognition and camouflage recognition experiments were generally a success, and suggested that further research is warranted.

In conclusion, this research extended the field of pattern recognition by developing, assessing the performance, and determining the mechanism of operation of a novel pattern recognition algorithm that is applicable to any set of binary images, in accordance with the enumerated research goals:

Quantify the performance of the pattern recognition algorithm as applied to fingerprints

a. Optimize algorithm parameters for fingerprint identification

2) Increase understanding of the algorithm’s operation and abilities

a. Justify tacit assumptions underlying algorithm operation

b. Identify mechanism by which the algorithm operates in certain instances

i. Apply concepts from integral geometry

3) Qualify the applicability of the algorithm to other pattern recognition problems

a. Cloud and precipitation particle recognition and camouflage recognition

All of the stated goals were accomplished. The current algorithm was optimized for fingerprint identification and its performance for that application was assessed. Assumptions underlying the operation of the algorithm were justified and integral geometry was used to elucidate the mechanism by which the algorithm operates. Finally, the ability of the algorithm to address other pattern recognition problems was qualified.
Chapter 6: Recommendations

It is recommended that the algorithm developed herein undergo further testing and development as a fingerprint identification algorithm. Although the first iteration of the algorithm did not perform particularly well, its performance could be enhanced by combining it with another algorithm. Specifically, there would be advantages to combining the algorithm discussed herein with an algorithm that pre-classifies fingerprints according to their global pattern (arch, tent, loop, and whorl). Additionally, it would be interesting to test the ability of the current algorithm to classify fingerprints according to their global pattern. Pre-classifying fingerprints by global pattern would decrease the number of potential matching fingerprints the algorithm would have to examine when looking for matches, which is likely to improve (reduce) the EER.

Additionally, it is possible to conceive of many variations on the current algorithm. One particularly exciting conception entails centering the mid-point of the random step on minutiae. Conventional fingerprint preprocessing algorithms could be used to highlight and identify the type and location of minutiae within a fingerprint. Then, a scale-spectrum could be generated for each minutiae point in the fingerprint. Thus, each fingerprint would have a number of scale-spectra associated with it. These scale-spectra would capture information only in the region near minutiae. This variation might provide a significant EER improvement while maintaining the advantages inherent to the algorithm. In summary, there needs to be more research before it can be definitively stated that the algorithm developed herein is ill-suited to fingerprint recognition.
Beyond fingerprint recognition, there are many pattern recognition problems to which the algorithm developed herein may be applicable. It is recommended that a review of the pattern recognition field be completed, with the goal of identifying areas of applicability for the algorithm discussed herein. The algorithm seems well-suited to global pattern recognition, and there may be a particular application to which it is especially well-suited, but this is impossible to know with a comprehensive review of the current field of pattern recognition applications.

Additionally, the algorithm could use further development to expand its use beyond binary two-dimensional images. The possibility of extending the algorithm to one-dimensional, three dimensional, or n-dimensional signals should be explored. This would allow the algorithm to seek application in the fields of text analysis, voice analysis, 3D image analysis, etc. Also, it may be beneficial to extend the algorithm beyond binary images. This would require the addition of new transition probabilities, but the increased complexity may be worthwhile depending on the application. In conclusion, there are many potential variations that could be envisioned for this algorithm. However, the advantages provided by this algorithm derive from its straightforward and simple manner of operation. Therefore, it is recommended that any future algorithm variations remain as simple and as straightforward as possible to ensure that transition probabilities can be estimated quickly and with confidence, and to ensure that the advantages inherent to the algorithm are maintained.
Bibliography


Appendix A: Fingerprint Preprocessing Function

import struct
import cStringIO
import Image

# Import C functions necessary for creating and destroying large arrays
cdef extern from "stdlib.h":
    void *malloc(int size)
    void free(void *ptr)

def transformOutline(imageTuple, variableList):
    # Define the necessary C variables
cdef short int height
cdef short int width
cdef short int edgeLength
cdef short int xLower
cdef short int xUpper
cdef short int yLower
cdef short int yUpper
cdef short int iLoop
cdef short int iLoop2
cdef short int iLoop3
cdef short int iLoop4

cdef int areaSum

cdef int index

cdef short int pixel

cdef short int *image

cdef short int *string

cdef float averagePixel

cdef float variance

cdef float varianceLimit

cdef short int whiteLimit

cdef short int blackLimit

cdef int leftSum

cdef int rightSum

# Store the python outline variables as C variables

width = variableList[0]

height = variableList[1]

edgeLength = variableList[2]

varianceLimit = variableList[3]

blackLimit = variableList[4]

whiteLimit = variableList[5]

# Use malloc to dynamically create potentially large c arrays

image = malloc(2*height*width)
string = malloc(2*height*width)

if(image == NULL or string == NULL):
    # Malloc was unsuccessful, we can not continue
    return -101

yLower = edgeLength/2
yUpper = height - yLower - 1
xLower = yLower
xUpper = width - xLower - 1

# Transform python image tuple into a C array
for iLoop from 0 <= iLoop < height:
    for iLoop2 from 0 <= iLoop2 < width:
        index = iLoop * width + iLoop2
        image[index] = imageTuple[index]

# Perform the outlining transformation
for iLoop from 0 <= iLoop < height:
    for iLoop2 from 0 <= iLoop2 < width:
        # Check to see if we are in bounds
        if(iLoop >= yLower and iLoop <= yUpper and iLoop2 >= xLower and iLoop2 <= xUpper):
            # We are in bounds, we can continue
            # Initialize the sum of our local area and the calculated variance
areaSum = 0
variance = 0
for iLoop3 from 0 <= iLoop3 < edgeLength:
    for iLoop4 from 0 <= iLoop4 < edgeLength:
        # Calculate the index
        index = (iLoop + iLoop3 - edgeLength/2) * width
        index = index + iLoop2 + iLoop4 - edgeLength/2
        areaSum = areaSum + image[index]
    # We have the sum over the area, no we can determine the average
    averagePixel = areaSum/(edgeLength * edgeLength)
    # Now that we have the average, we can calculate the variance
    for iLoop3 from 0 <= iLoop3 < edgeLength:
        for iLoop4 from 0 <= iLoop4 < edgeLength:
            # Calculate the index
            index = (iLoop + iLoop3 - edgeLength/2) * width
            index = index + iLoop2 + iLoop4 - edgeLength/2
            variance = variance + (image[index] - averagePixel) * (image[index] - averagePixel)
        # Perform final variance calculation
        variance = variance / (edgeLength * edgeLength)
    # Determine if pixel should be masked or unmasked based on variance
    index = iLoop * width + iLoop2
    if(variance < varianceLimit):
        # There’s not a lot of variance, we should mask this pixel
        string[index] = -1
else:

    # There's variance here, probably not a smudge or open area
    # Now we must determine if the pixel is white or black
    if(averagePixel > whiteLimit):
        # The pixel is easily classified as white
        string[index] = 1
    elif(averagePixel < blackLimit):
        # The pixel is easily classified as black
        string[index] = 0
    else:

        # We must do more work to determine if the pixel is black or white
        if (image[index] > averagePixel):
            # The pixel is lighter than it's surroundings, call it white
            string[index] = 1
        else:

            # The pixel must be black
            string[index] = 0
    else:

        # We are not within the x and y boundaries, make the pixel green
        index = iLoop * width + iLoop2
        string[index] = -1

# Use a series of loops to transform the C string array into a python string
out = cStringIO.StringIO()
for iLoop from 0 <= iLoop < height:
    for iLoop2 from 0 <= iLoop2 < width:
        index = iLoop * width + iLoop2
        if(string[index] == -1):
            # Our point should be masked, make it green
            bin_str = struct.pack("BBB", 0, 255, 0)
            out.write(bin_str)
        elif(string[index] == 0):
            # Our point is black
            bin_str = struct.pack("BBB", 0, 0, 0)
            out.write(bin_str)
        else:
            # Our point is white
            bin_str = struct.pack("BBB", 255, 255, 255)
            out.write(bin_str)

# Don't forget to free the memory associated with the malloc'd arrays
free(image)
free(string)

# Return an image variable
tempTuple = (width, height)
return Image.fromstring("RGB", tempTuple, out.getvalue())
Appendix B: Scale-Spectra Generating Function

# Import the C functions we will need during our random walk
cdef extern from "math.h":
    double cos(double theta)
    double sin(double theta)
    double acos(double negOneToOne)
    double sqrt(double number)
    double pow(double base, double exp)
    float floor(float decimal)

# Import C functions necessary for creating and destroying large arrays
cdef extern from "stdlib.h":
    void *malloc(int size)
    void free(void *ptr)

def gridWalk(path, name, variableList):

    # Import the necessary python extensions
    import random

    # Define the necessary C variables
    cdef short int iHeight
    cdef short int iWdth
cdef float fScale

cdef short int iEdgeLength

cdef int iterations

cdef int iRow

cdef int iColumn

cdef int iNetRow

cdef int iNetColumn

cdef short int iIsGreen

cdef short int iIsOB

cdef int iCounter

cdef int iCounter2

cdef float fHyp

cdef float fX

cdef float fY

cdef float fXStar

cdef float fYStar

cdef int iXOffset

cdef int iYOffset

cdef float fThetaRandom

cdef float fThetaRaw

cdef float fThetaAdjusted

cdef short in iIndex

cdef short int iPow

cdef short int iX
cdef short int iY

cdef char *image

cdef unsigned char *net

# Use malloc to dynamically create two potentially large c arrays
# The first array represents the image file
# The second array represents the walk data at a given scale
image = malloc(2 * iHeight * iWidth)
net = malloc(2 * iEdgeLength * iEdgeLength)

if(image == NULL):
    # Malloc was unsuccessful, we can not continue
    print "There was not enough memory to create the image array"
    return -101

if(net == NULL):
    # Malloc was unsuccessful, we can not continue
    print "There was not enough memory to create the net array"
    return -101

if(iIterations % 8 != 0):
    # We can not properly compress data unless iIterations % 8 is 0
    print "The number of iterations must be exactly divisible by 8"
    return -102
# Open a text file to store the results of the walk
fout = open("C:\netWalkScale22.txt", "w")

# Determine and store information related to the image
self.im = Image.open(path + name)
iWidth = self.im.size[0]
iHeight = self.im.size[1]

# The following code places the images BnW values into a C array
orig_pixels = self.im.getdata()

pixelList = []
for i in range(0, len(orig_pixels)):
    pixelList.append(orig_pixels[i])

imageTuple = tuple(pixelList)

# We now transfer the Python list representation of the image into
# a C array of characters where:
# A value of 0 represents black
# A value of 1 represents white
# A value of -1 represents the green mask
for iRow from 0 <= iRow < iHeight:
    for iColumn from 0 <= iColumn < iWidth:
        # Check to see if we are outside of the green mask
if((imageTuple[iRow * iWidth + iColumn][0] + imageTuple[iRow * iWidth + iColumn][1])
!= 255):
    # We are outside of the green mask, transfer the value into a C array
    if(imageTuple[iRow * iWidth + iColumn][0] == 0):
        image[iRow * iWidth + iColumn] = 0
    else:
        image[iRow * iWidth + iColumn] = 1
    else:
        # The pixel is green
        image[iRow * iWidth + iColumn] = -1

# Store the python random walk variables as C variables
fScale = variableList[0]
iEdgeLength = variableList[1]
iIterations = variableList[2]

# Seed the random number generator with the system time
random.seed()

# Initialize our net array
for iCounter2 from 0 <= iCounter2 < pow(iEdgeLength, 2):
    net[iCounter2] = 0

# Loop where iterations are performed
for iCounter from 1 <= iCounter <= iIterations:

    # Initialize our variables that ensure valid point selection
    iIsGreen = 0
    iIsOB = 0

    # Enter a loop where we pick points until we get valid ones
    while(iIsGreen == 0 or iIsOB == 0):
        # Assume all points in our net are valid -- until we find otherwise
        iIsGreen = 1
        iIsOB = 1

        # Choose the center and angle of our net
        fX = iWidth * random.random()
        fY = iHeight * random.random()
        fThetaRandom = 2 * 3.14159 * random.random()

        # Enter a loop check all n x n grid points for validity
        for iNetRow from 0 <= iNetRow < iEdgeLength:
            for iNetColumn from 0 <= iNetColumn < iEdgeLength:
                iXOffset = (iNetColumn - floor(iEdgeLength/2))
                iYOffset = (iNetRow - floor(iEdgeLength/2))
                fHyp = sqrt(iXOffset * iXOffset + iYOffset * iYOffset)

                if(fHyp != 0):
                    # Determine which of the four quadrants we are in
                    if(iYOffset >= 0):
                        # We are in quadrant I or quadrant II

fThetaRaw = acos(iXOffset/fHyp)

else:
    # We are in quadrant III or quadrant IV
    fThetaRaw = 2 * 3.14159265 - acos(iXOffset/fHyp)

# Adjust our random angle with our net point angle
fThetaAdjusted = fThetaRandom + fThetaRaw

# Calculate real hypotenuse
fHyp = fHyp * fScale / 2
fHyp = fHyp / sqrt(2 * floor(iEdgeLength/2) * floor(iEdgeLength/2))
fxStar = fX + fHyp * cos(fThetaAdjusted)
fyStar = fY + fHyp * sin(fThetaAdjusted)

# Now we have the x and y coordinates of a particular net point
# Now we can check to see if these points are in bounds
iX = fxStar
iY = fyStar
if(iX < 0 or iX >= iWidth or iY < 0 or iY >= iHeight):
    iIsOB = 0
    break

# If the point was not OB, it may be green
if(image[iY * iWidth + iX] == -1):
    iIsGreen = 0
    break

# End of iNetColumn for loop

# Check to see if we are OB or the point is green
if(iIsOB == 0 or iIsGreen == 0):
    break
# End of iNetRow for loop
# End iIsOB or iIsGreen while loop

# If we've made it this far we have a valid net of points
for iNetRow from 0 <= iNetRow < iEdgeLength:
    for iNetColumn from 0 <= iNetColumn < iEdgeLength:
        iXOffset = (iNetColumn - floor(iEdgeLength/2))
        iYOffset = (iNetRow - floor(iEdgeLength/2))
        fHyp = sqrt(iXOffset * iXOffset + iYOffset * iYOffset)
        if(fHyp != 0):
            # Determine which of the four quadrants we are in
            if(iYOffset >= 0):
                # We are in quadrant I or quadrant II
                fThetaRaw = acos(iXOffset/fHyp)
            else:
                # We are in quadrant III or quadrant IV
                fThetaRaw = 2 * 3.14159265 - acos(iXOffset/fHyp)
            # Adjust our random angle with our net point angle
            fThetaAdjusted = fThetaRandom + fThetaRaw
            # Calculate real hypotenuse
            fHyp = fHyp * fScale / 2
            fHyp = fHyp / sqrt(2 * floor(iEdgeLength/2) * floor(iEdgeLength/2))
\[ fXStar = fX + fHyp \times \cos(fThetaAdjusted) \]
\[ fYStar = fY + fHyp \times \sin(fThetaAdjusted) \]

# Now we have the x and y coordinates of a particular net point

# Now we can check to see if these points are in bounds

iX = fXStar

iY = fYStar

# Test the color of one particular net pixel

iIndex = iNetRow * iEdgeLength + iNetColumn

iPow = iCounter % 8

if(image[iY * iWidth + iX] == 0):
    # Our pixel is black
    # We always add zero if the pixel is black, so we take no action

elif(image[iY * iWidth + iX] == 1):
    # Our pixel is white
    net[iIndex] = net[iIndex] + pow(2, iPow)

else:
    # We should never see this!
    print "We have an error in the code that compresses the eight net values"

# See if we have 8 values in our array, if so compress them into a single line in a .txt file

if(iIterations % 8 == 0):
    # Compress the values into a single line of a .txt file
    for iCounter2 from 0 <= iCounter2 < pow(iEdgeLength, 2):
        # If we need any leading zeros, add them here
        if(net[iCounter2] > 99):
str = str + net[index]
elif(net[iCounter2] > 9 and net[index] < 100):
    str = str + '0' + net[index]
elif(net[iCounter2] >= 0 and net[index] < 10):
    str = str + '00' + net[index]
else:
    print "We have an error in the code section that adds leading zeros"
    # Add a space after the value
    str = str + ' '
    # Write the string to the file
    fout.write(str)
    # initialize our array for the next round
    net[iCounter2] = 0
    # End of iCounter2 for loop
    # End of .txt line write
    # End of iNetColumn for loop
    # End of iNetRow for loop
    # End of iIterations for loop
    # Free the memory associated with the malloc call
    free(image)
    free(net)
    # Close the text file we've created
fout.close()

# Successfully return
return 0
Appendix C: Template Generating Function

# Import native Python modules
import comparison, os, shutil, sys, time

# Import our custom made Pyrex and Python extensions
import fpBnW01, fpWalk01
import ImageAnalyzer

# Add the path where the fingerprints are stored
imageDirectory = 'C:\Python25\Lib\site-packages\Pyrex\Distutils\2002 fvc 110 by 8\Processed\'
textFileDirectory = 'C:\Python25\Lib\site-packages\Pyrex\Distutils\2002 fvc 110 by 8\textFiles\'
os.sys.path.append(imageDirectory)
os.sys.path.append(textFileDirectory)

# Take care of some pre-loop needs
startTime = time.time()
iterations = 500000
startingScale = 0
endingScale = 30
increment = 0.5

# A text file will be created for each of the 880 outlined images
for iLoop in range(6,7):
for iLoop2 in range(5,6):

    sName = str(iLoop) + '_' + str(iLoop2) + '.bmp'

    image = ImageAnalyzer.ImageAnalyzer(imageDirectory, sName)

    results = image.monochromeWalk(iterations, startingScale, endingScale, increment)

    if (type(results) == type(1)):
        print '\nprint was skipped
'

    else:
        p1TextFile = open(textFileDirectory + str(iLoop) + '_' + str(iLoop2) + '.txt','w')
        p1TextFile.write(str(iLoop) + '_' + str(iLoop2) + 'b.txt' + '\n')
        p1TextFile.write('Created on ' + str(time.asctime()) + '\n')
        p1TextFile.write('File created by Joseph M. Stoffa\n')
        p1TextFile.write(' D(mm)\tt Pbb\tt Pww\tt Pwb\n------\tt------\tt------\tt------\n')
        for iLoop3 in range(0, int(float(endingScale)/float(increment)) + 1):
            scale = str('%.4f' % (results[iLoop3][0] * 0.08467))
            p1TextFile.write(scale + '\t' + str(results[iLoop3][1]) + '\t')
            p1TextFile.write(str(results[iLoop3][2]) + '\t' + str(results[iLoop3][3]) + '\n')
        p1TextFile.close()
        print 'Text file for ', sName, ' has been created'

    totalTime = time.time() - startTime
    print 'The total time taken was ', totalTime
    print 'FIN'
Appendix D: Matching Score Calculation Function

# Import the necessary Python libraries
import random

def comparison(printOne, printTwo, metric, quantity, startingScale = 0, endingScale = 9999):
    # This is where the module documentation is stored
    ""
    The comparison module compares two fingerprints
    The listData input is a python list containing data on two fingerprints
    printOne[0][0...n] is the scale data for fingerprint one
    printOne[1][0...n] is the alpha1 data for fingerprint one
    printOne[2][0...n] is the alpha2 data for fingerprint one
    printOne[3][0...n] is the beta data for fingerprint one
    printTwo[0][0...n] is the scale data for fingerprint two
    printTwo[1][0...n] is the alpha1 data for fingerprint two
    printTwo[2][0...n] is the alpha2 data for fingerprint two
    printTwo[3][0...n] is the beta data for fingerprint two
    The "metric" variable refers to the method of comparison
    The following text arguments are valid for the metric variable
    linear -- Computes the linear distance between two spectra
    square -- Computes the square of the linear distance between two spectra
    FFT -- computes the linear difference between the Fourier transform of two spectra
    The "quantity" variable determines what the "metric" compares
The following text arguments are valid for the quantity variable

alphaOne -- The quantity measured by the metric will be the probability of white-white

alphaTwo -- Quantity compared will be probability of black-black

beta -- Quantity compared will be probability of black-white + white-black

determinant -- Quantity compared will be the determinant of the 2x2 matrix

eigenValues -- Quantity compared will be the Eigenvalues of the 2x2 matrix

trace -- Quantity compared will be the trace of the 2x2 matrix

The startingScale and endingScale determine which section of spectra undergoes comparison

If these arguments are omitted, all scales will be compared

The startingScale and endingScale are inclusive, these scales will be compared

Any two numerical arguments for the startingScale and endingScale are valid given that

The startingScale is less than the endingScale

# Check to see if arguments are valid

# Check if listData is a variable of type list

if(type(printOne) != type([]) or type(printTwo) != type([])):
    # Our input is not a list, we need to return an error value
    return -230

# Check if data representing alphas and beta is in integer form

for iLoop in range(1,4):
    if(type(printOne[iLoop][0]) != type(1) or type(printTwo[iLoop][0]) != type(1)):
        return -231

# Check that the user has selected an acceptable metric
if(metric != 'linear' and metric != 'percentage' and metric != 'square'):
    return -232

# Check that the user has selected a valid quantity to compare
if(quantity != 'alphaOne' and quantity != 'alphaTwo' and quantity != 'beta'):
    if(quantity != 'determinant' and quantity != 'eigenValues' and quantity != 'trace'):
        if(quantity != 'ndeterminant' and quantity != 'random'):
            return -233

# Check to ensure the starting scale is smaller than the ending scale
if(startingScale > endingScale):
    return -234

# We need to determine the number of entries in printOne
entries = len(printOne[0])

# We need to determine the index of our startingScale
startingIndex = 0
while(startingScale > printOne[0][startingIndex]):
    startingIndex = startingIndex + 1

if(startingIndex > entries):
    # Our startingScale is greater than the last scale in the data list
    return -235

# Determine the number of iterations that occur at each scale of fingerprint 1
# This assumes the number of iterations that occurred at the first scale occur at all scales
# Also, fingerprint 1 and 2 must have the same number of iterations at each scale
# Otherwise, the comparison result will be fingerprint order specific
iterations = printOne[1][0] + printOne[2][0] + 2*printOne[3][0]
alphaOneW0 = printOne[1][0]
alphaOneB0 = printOne[2][0]
alphaTwoW0 = printTwo[1][0]
alphaTwoB0 = printTwo[2][0]

# Initialize some variables before entering our loop
differenceSum = 0
iCounter = startingIndex
iScalesCompared = 0

# See if we reach the endingScale before we reach the last index in the print
while(iCounter < entries and printOne[0][iCounter] <= endingScale):
    # Make certain our two scales are equal
    if(printOne[0][iCounter] != printTwo[0][iCounter]):
        return -236

    # Determine the difference based on our metric
    if(quantity == 'alphaOne'):
        # Calculate the difference between the alpha ones
        difference = abs(printOne[1][iCounter] - printTwo[1][iCounter])
    elif(quantity == 'alphaTwo'):
        # Calculate the difference between the alpha twos
        difference = abs(printOne[2][iCounter] - printTwo[2][iCounter])
    elif(quantity == 'beta'):
        # Calculate the difference between the betas
        difference = abs(printOne[3][iCounter] - printTwo[3][iCounter])
    elif(quantity == 'determinant'):
# Calculate the difference between the two determinants

difference = (printOne[1][iCounter] * printOne[2][iCounter] - pow(printOne[3][iCounter], 2))

difference = abs(difference - (printTwo[1][iCounter] * printTwo[2][iCounter] - pow(printTwo[3][iCounter], 2)))

elif(quantity == 'random'):

    # Calculate the difference between the two determinants
    difference = random.random()

elif(quantity == 'ndeterminant'):

    # Calculate the difference between the two determinants
    difference = (printOne[1][iCounter] * printOne[2][iCounter] - pow(printOne[3][iCounter], 2))/alphaOneW0/alphaOneB0

difference = abs(difference - ((printTwo[1][iCounter] * printTwo[2][iCounter] - pow(printTwo[3][iCounter], 2))/alphaTwoW0/alphaTwoB0))


elif(quantity == 'eigenvalues'):

    # Do something else
    return -237


elif(quantity == 'trace'):

    # Calculate the difference between the two traces
    difference = printOne[1][iCounter] + printOne[2][iCounter]

difference = abs(difference - printTwo[1][iCounter] - printTwo[2][iCounter])

else:

    # We should never see this line of code
    return -237
error = (pow(printOne[1][iCounter], 0.5) + pow(printTwo[1][iCounter], 0.5))/2*2*.7

if(quantity == 'determinant'):
    error = pow(error, 2)

if(metric == 'square'):
    difference = pow(difference, 2)
    error = pow(error, 2)

if(metric == 'percentage'):
    if(difference < error):
        difference = 0
    else:
        difference = 1

# We've completed on difference calculation
iCounter = iCounter + 1
if(metric == 'linear' or metric == 'square'):
    differenceSum = differenceSum + float(difference)/float(error)
if(metric == 'percentage'):
    differenceSum = differenceSum + difference
iScalesCompared = iScalesCompared + 1

# The main loop is over
# We can now return the average difference between two scale spectra

if(metric == 'linear' or metric == 'square'):
    return float(differenceSum/float(iScalesCompared))

if(metric == 'percentage'):
    return float(1.0 - float(differenceSum)/float(iScalesCompared))