GLOBAL IMAGE SEGMENTATION FOR TWO-PHASE AND MULTI-PHASE GEOMATERIAL CHARACTERIZATION

by

Kokeb Abay Abera

A thesis submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Master of Civil Engineering

Spring 2017

GLOBAL IMAGE SEGMENTATION FOR TWO-PHASE AND MULTI-PHASE GEOMATERIAL CHARACTERIZATION

by

Kokeb Abay Abera

Approved: _____
Kalehiwot Nega Manahiloh, Ph.D.
Professor in charge of thesis on behalf of the Advisory Committee

Approved: _____
Harry W. Shenton III, Ph.D.
Chair of the Department of Civil and Environmental Engineering

Approved: _____
Babatunde A. Ogunnaike, Ph.D.
Dean of the College of Engineering

Approved: _____
Ann L. Ardis, Ph.D.
Senior Vice Provost for Graduate and Professional Education

ACKNOWLEDGMENTS

First and foremost, I would like to thank God, for none of this would have been possible without him. With great sincerity, I would like to express my gratitude to my advisor and professor at the University of Delaware, Dr. Kalehiwot Manahiloh, for his patience, advice, and support during the past two years. It was an honor and a positively educational experience to have the opportunity to learn and work with him.

I acknowledge my family and friends for their support during my studies, because it would have been impossible for me to complete my degree and this thesis without their involvement. I thank my wonderful, loving parents, Abera and Fisaha, for raising me to achieve great things and to always strive to be the best person that I can be. I will always be thankful for all of the sacrifices they have made to give me a great life. I thank my younger brother Abel for being the best, loving brother I could ask for and always having a wonderful, direct impact on my life throughout the years.

I extend my gratitude to my fellow graduate students for their valuable suggestions and comments along my journey. There are too many to name, but most notably, I highly appreciate Mohammad Motalleb Nejad for his guidance in coding.

Finally, I would also like to thank the Center for Advanced Infrastructure and Transportation (CAIT) at the University of Delaware for the one year fellowship provided during my studies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The effectiveness of five global thresholding techniques, to accurately segment different two-phase geomaterials (solids and air), was evaluated in this work. X-ray computed tomography (CT) images taken from pervious concrete, glass bead, and silica sand specimens were analyzed for evaluating the five chosen methods. The core algorithm of each of these methods was coded as a standalone graphical user interface (GUI) application software. From the results, it can be said that, no single image segmentation technique performs well over a wide range of material and that the performance of each image segmentation technique varies depending on the type and state of the analyzed media.

X-ray CT images of three-phase (solids, water, and air) silica sand and glass bead specimens were analyzed and used for evaluating the segmentation performances of three methods. Based on the observations made on these methods, a refined statistical-based global segmentation algorithm is proposed for segmenting partially saturated granular geomaterials. The findings for the silica sand specimen showed that the proposed technique estimated void ratio with 1.52 percent error and degree of saturation with 4.35 percent error. On the other hand, void ratio for the glass bead specimen yielded a high percent error of 15.63 whereas degree of saturation had a very low percent error of 0.34. By comparing the results of the proposed technique with the other Otsu-based techniques, it is concluded

that the proposed algorithm performs better in segmenting three-phase granular geomaterials.

Chapter 1

INTRODUCTION

1.1 Image Segmentation

There are many different types of images. Thermal images, magnetic resonance images (MRI), and light intensity (visual) images are some examples. Of all image types, light intensity images are the most common. As alluded to by the name, this type of image represents the variation of light intensity in a scene (Pal and Pal 1993). Regardless of type, images can be viewed as digital images. Mathematically speaking, a digital image is a two-dimensional discrete function, f(x,y) that is digitized in both spatial coordinates and the magnitude of the feature value (e.g., temperature intensity, light intensity, and depth) (Pal and Pal 1993). The values of x and y in a discrete function represent row and column indices, respectively. The meeting of these indices marks a point referred to as a pixel. The pixel equivalent in three-dimensional space is referred to as a voxel (Razavi 2006; Manahiloh et al. 2012; Manahiloh 2013).

Image segmentation is a general term that is utilized to describe the process of separating an area of interest, a pattern, or a subset of pixels with common features in an image through usage of various techniques (Liao et al. 2001). The pixels in images acquired via X-ray computed tomography (CT) scanning contain gray-level intensity information. Each pixel's intensity varies from black (weakest shade of gray) to white (strongest shade

of gray) (Madra et al. 2014). The intensity value for each pixel is saved as an aggregate of bits. For example, 8-bit images have intensity values varying from 0 to 255. The common practice in image segmentation is to extract the pixels with the desired color or gray intensities.

1.2 Thresholding

Since the pioneering work of Brice and Fennema (1970), image segmentation techniques have undergone immense evolution, including in the direction of histogram thresholding. Thresholding is a simple yet popular concept that introduces one or more intensity values to an intensity distribution (i.e., image histogram) of an image where these values separate the foreground (i.e., objects of interest) from the background. The field of image thresholding has been well researched, yielding many different models that can be used to achieve the same result of effectively segmenting an image. Depending on the constituent elements (i.e., phases) of an image, thresholding techniques could be bi-level or multi-level (Leedham et al. 2003). Bi-level thresholding techniques introduce one threshold value to the image histogram and produce an image segmented into two distinct regions (Kohler 1981; Pal and Pal 1993). “Foreground” and “background” are the terms applied to the pixel values greater than and less than a threshold value, respectively (Kurita et al. 1992; Abdullah et al. 2012). Multi-level thresholding methods, on the other hand, introduce more than one threshold value to the intensity distribution (Kapur et al. 1985; Arora et al. 2008). Regardless of the approach, the number of segmented regions is always equal to the number of thresholds plus one.

Thresholding techniques are divided into two general classes of global and local thresholding. While global thresholding techniques use the statistical information of an intensity value distribution for the total pixels in a given image, local thresholding methods use the statistical information of a set of neighboring pixels to classify a pixel (Singh et al. 2011). In other words, global thresholding is used to choose a threshold value that is only dependent on gray-level values while relating to the characteristics of pixels, whereas a local thresholding approach determines a threshold value through usage of both the gray-level value and local property of a pixel (e.g., range and variance). Local thresholding divides an image into several subregions and chooses a threshold for each of these regions. After such thresholds are applied, a gray-level filtering technique is used to eliminate discontinuous gray-levels among the subregions.

The main problem associated with global thresholding techniques is that the effects of noise (i.e., random variation in pixel intensity) cannot be eliminated by these methods (Leedham et al. 2003). This results in the lack of prominent peaks forming for all of the features of interest in an image. However, for segmentation of images with clear distinct phases, as it is the case with granular materials, global thresholding techniques provide sufficiently accurate results. Therefore, the research conducted for this thesis focuses on global thresholding techniques.

1.3 Research Motivation

In the engineering world, laboratory tests are conducted on various specimens to determine important engineering properties, such as void ratio (e) and degree of saturation (S%). Depending on the shape of the grains, the grain size distribution, and the packing or

arrangement of the grains, granular materials can have a rather wide range of void ratios (Holtz et al. 2011). Void ratio is an essential component in geotechnical design with regards to shear strength and volume change. Shear strength plays a role in the evaluation of bearing capacity of different materials used in geotechnical engineering practice. Volume change and bearing capacity have a direct effect in the determination of the settlement of such materials, most notably soils. These two soil behaviors have a pivotal role in understanding soil-structure interactions, which are crucial for sustainable infrastructure design.

Degree of saturation is a critical component in understanding the behavior of unsaturated soils (i.e., partially saturated soils). Figure 1 provides this relation in the form of a soil-water characteristic curve (SWCC). As shown in the figure, the SWCC is created by relating soil suction to degree of saturation, or soil water content. Soil suction refers to total suction, which is defined as the thermodynamic potential of soil pore water relative to a reference potential of free water (Lu and Likos 2004). Assuming that the three soils in the figure are all equilibrated at the same matric suction, the clay would have the highest water content and degree of saturation due to having a very high specific surface area and charged surfaces; the specific surface areas of sand and silt are lower. Clay pores are also much smaller than those of sand and silt, which results in clay having the greatest potential for holding water. The general shape of the SWCC for various soils reflects the dominating influence of material properties including pore size distribution, grain size distribution, density, organic material content, clay content, and mineralogy on the pore water retention behavior (Lu and Likos 2004).

The performed work (explained in later chapters) is conducted in order to quickly and accurately determine the two engineering properties, void ratio and degree of saturation, on two-phase and multi-phase images of porous geomaterial specimens. X-ray CT images, taken from two-phase pervious concrete, glass bead, and silica sand specimens and three-phase glass bead and silica sand specimens, are analyzed with implemented thresholding techniques. The core algorithms for these techniques are coded using the MATLAB© (Mathworks 2015) programming language and packaged into a standalone application software, a graphical user interface (GUI). Rather than performing laboratory experiments on physical specimens of porous media, three-dimensional X-ray CT images of the media can be used in conjunction with thresholding techniques to quickly and accurately obtain the material properties. The chosen thresholding techniques, such as Otsu (1979), Pun (1980), and Kapur et al. (1985), were originally analyzed on common images of letters, people, structures, etc. The accuracy of these old yet successful algorithms is extended to test the segmentation of porous geomaterials.

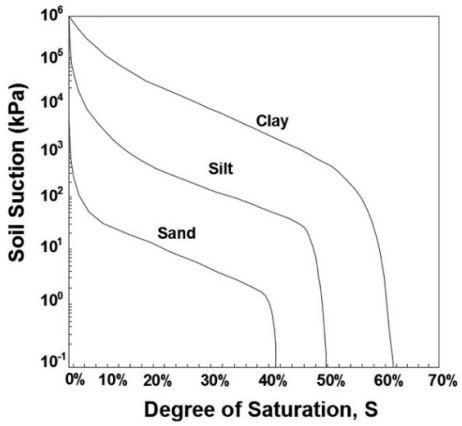

Figure 1. General behavior of soil-water characteristic curves for sand, silt, and clay (Modified from Lu and Likos (2004)).

1.4 Organization of the Thesis

The remainder of this thesis will provide lengthy, yet informative details on the process utilized to successfully perform image segmentation for both bi-level and multi-level geomaterial characterization. Chapter 2 is composed of background information that is necessary for understanding the logic and reasoning behind the segmentation approach. This chapter is subdivided into sections on image acquisition, image processing, and literature reviews, with the last focusing on the history and development of image acquisition systems, global image segmentation, and local image segmentation. Chapter 3 focuses on the implementation of global thresholding techniques for both the five two-phase and the three three-phase thresholding techniques chosen for this work. Chapter 4 contains a description on the application of global thresholding techniques. Most notably, it provides a description of the purpose of creating graphical user interface (GUI) executable software and corresponding features and capabilities. This chapter also includes explanations supporting the workability of a two-phase segmentation GUI and a three-phase segmentation GUI, which were designed for this study. Chapter 5 presents the results and discussion pertaining to the porous geomaterials analyzed. Chapter 6 comprises the conclusions of the results, as well as future work.

Chapter 2

BACKGROUND

2.1 Image Acquisition and Processing

In today's market, various X-ray image acquisition systems, such as microtomography, computed radiography, and computed tomography, can be utilized for imaging granular materials. The acquisition system chosen for this work is X-ray computed tomography (CT), which is an advanced imaging technique that allows for nondestructive and noninvasive imaging of specimens to depict cross-sectional and three-dimensional internal structures (Iassonov et al. 2009). Such a system is especially useful for highly porous materials (Maire et al. 2007).

Figures 2 and 3 provide general schemes for the setup of image acquisition systems in both two-dimensional and three-dimensional orientations. An X-ray beam, originating from an X-ray source, passes through a specimen resting upon a pedestal. The pedestal, which has four degrees of freedom, moves the specimen left and right, forward and backward, up and down, and rotationally about an axis perpendicular to the beam. The beam reaches the detector where data that is useful in projecting the internal structural details of the scanned media is created. During data acquisition, an image slice rendering of the specimen is obtained once the pedestal completes a full 360-degree rotation. This process is repeated as the pedestal moves upwards and downwards to acquire the remaining

image slices that make up the specimen. Therefore, each image slice represents a portion of the specimen, and combining all of the slices together yields a virtual three-dimensional model of the specimen (Madra et al. 2014).

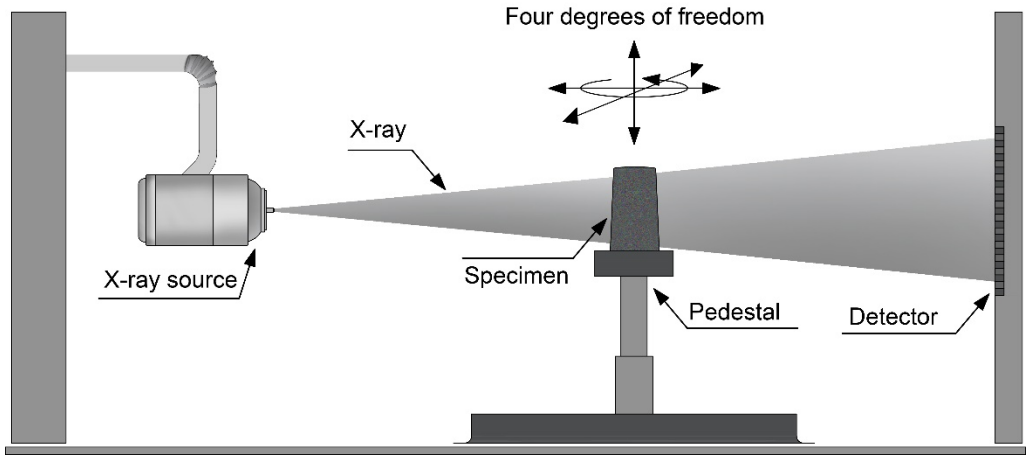

Figure 2. Two-dimensional schematic of specimen installation in an X-ray CT chamber.

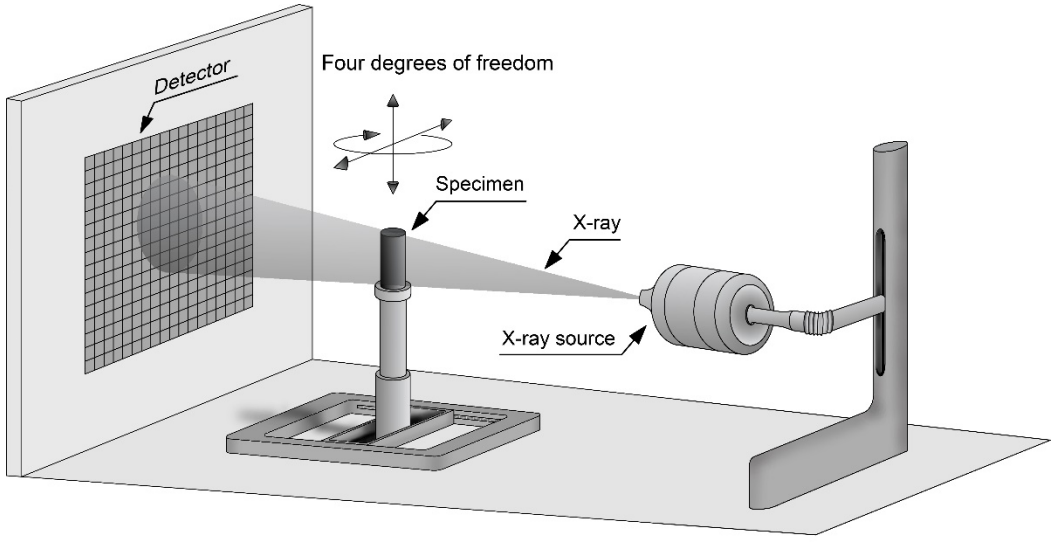

Figure 3. Three-dimensional schematic of specimen installation in an X-ray CT chamber.

Image processing deals with utilizing algorithmic programs to identify and extract information from images. Typically, image processing tools convert 8-bit or 16-bit grayscale images into binary images post-segmentation. For example, 8-bit images have 256 possible grayscale intensity values. Binary images would have grayscale intensity values of either 0 or 1 (Abdullah et al. 2012). Thus, by converting the grayscale images, there is a significant reduction in possible grayscale intensity values. This leads to binary images having a smaller storage space, being easier to manipulate, and being faster to process when a thresholding algorithm is applied (Arifin and Asano 2006).

2.1.1 Image Segmentation

The binarization of grayscale images is commonly referred to as image segmentation and is the foundation for object recognition and computer vision. Image segmentation partitions an image's overall histogram into two regions, with the goal of separating objects of interest (i.e., the foreground) from the background (Kohler 1981). Segmentation quality is controlled by how well the threshold that separates the regions is estimated.

Over the last few decades, advancements in technologies for image acquisition and processing have allowed for tremendous growth in both theory and application of image segmentation techniques. These techniques are very popular in many fields, such as agriculture (e.g., (Abdullah et al. 2012)), medicine (e.g., (Sund and Eilertsen 2003)), and forensics (e.g., (Wen and Chen 2004; Russ 2015)). The restoration of degraded (e.g., aging resulting in ink fading) scanned documents, including text, line drawings, or other graphics, can be made possible by image segmentation techniques.

Refer to Figure 4, which shows the typical structure of an image segmentation system. The first main step, preprocessing, is essential in reducing any notable noise (i.e., random variation in pixel intensity) if present in the image being analyzed. It is important to note that this process is completed prior to the image segmentation step. For image preprocessing, color spaces within the image are transformed into specifically given color spaces for identifying different portions of an image (Yang and Kang 2009). Techniques such as a Gaussian filter (Tsai 1995) are used for image smoothing to further minimize any noise that may hinder segmentation performance. Once an image segmentation algorithm is chosen, a determined threshold value is used to segment the image into different, identifiable regions. Post-processing work entails region merging, image marking, and region extraction. This last step is essential for extracting valuable information from portions of the segmented image; e.g., to count the number of solid, liquid, and air pixels for calculating void ratio and degree of saturation in both dry and partially saturated granular geomaterials. This portion of the segmentation process combines unreasonably discontinuous regions to allow for a successfully segmented image (Yang and Kang 2009). Note that the first and third steps may not be used since these steps are dependent on the quality of the original, raw image and the effectiveness of the chosen image segmentation algorithm, respectively.

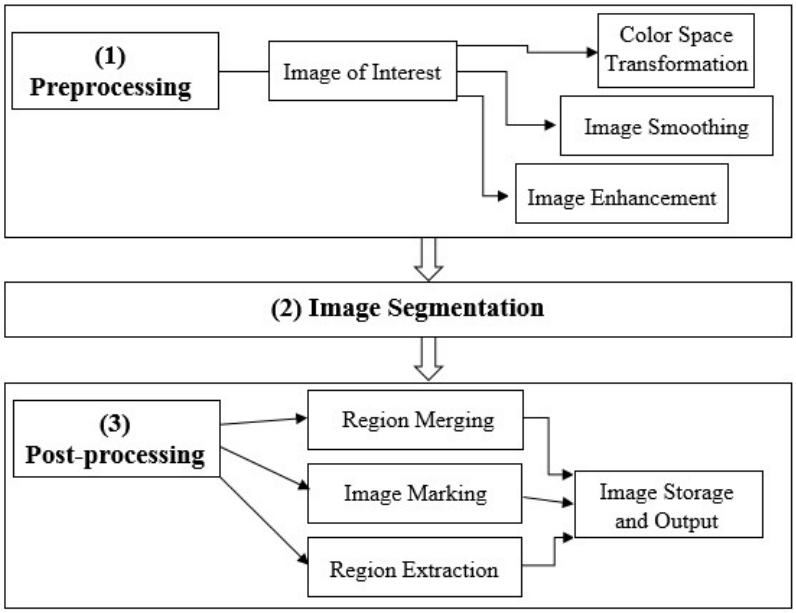

Figure 4. Image segmentation system structure (Yang and Kang 2009).

Image segmentation is broken down into three main categories: region-based segmentation, edge-based segmentation, and special theory-based segmentation (Yang and Kang 2009). Since this thesis focuses on thresholding techniques, region-based segmentation is the category of interest (Table 1).

Table 1. Region-based segmentation category (Yang and Kang 2009).

Sub-classes		**Interpretation**
Thresholding	Otsu	Extract the objects from the background by setting reasonable gray threshold *Ts* for image pixels.
	Optimal thresholding	
	Thresholding image	
Region operating	Region growing	Partition an image into regions that are similar according to given criteria, such as gray character, color character, texture character, etc.
	Region splitting and merging	
	Image matching	

The sub-classes of region-based segmentation are thresholding (i.e., global thresholding) and region operating (i.e., local thresholding). The thresholding sub-class is further divided into Otsu, optimal thresholding, and thresholding image techniques. The region operating class is subdivided into region growing, region splitting and merging, and image matching.

Overall, image segmentation is a very powerful tool for obtaining and analyzing the properties of porous media, such as soil, rocks, concrete, and glass. Rather than performing laboratory experiments on physical specimens of porous media, images of the media can be used in conjunction with thresholding techniques to quickly and accurately yield material properties such as void ratio.

2.2 Literature Review

2.2.1 Image Acquisition Systems

This section contains a literature review for the medical and industrial usage of image acquisition systems as well as a brief overview of the history and development of these systems. Two papers are reviewed for medical purposes, two papers for industrial purposes, and one paper for describing the historical side of image acquisition systems.

2.2.1.1 Medical Usage

The following two papers are reviewed as an example of the medical benefits that image acquisition systems provide:

1. Digital Image Acquisition and Processing in Medical X-ray Imaging – Aach et al. (1999)

2. A Comparison of Computed Tomography, Computed Radiography, and Film-Screen Radiography for the Detection of Canine Pulmonary Nodules – Alexander et al. (2012)

The authors of the first paper discuss popular techniques and future applications of digital X-ray imaging in medicine. They go over the advantages of using digital imaging in comparison to conventional analog methods. A real-time image acquisition system, X-ray fluoroscopy, is examined, and the authors present a unified radiography/fluoroscopy solid-state detector concept.

A main requirement of successful digital imaging is the separation of image acquisition and display. Images digitally acquired through conventional analog methods (using screen/film combinations) can be processed in order to correct accidental overexposure or underexposure, or to enhance diagnostically relevant information before display (Aach et al. 1999). Figure 5 shows a simple schematic representing the overall principle of conventional X-ray image detection. As shown in the figure, the X-ray film is placed between two intensifying screens. These screens ultimately convert the x radiation into visible light. The conversion blackens the X-ray film, which can be viewed on a light box.

(Figure 5 has been removed due to copyright restrictions.
http://electronicimaging.spiedigitallibrary.org/article.aspx?articleid=1097476)

For modern digital imaging, the quality of the separation of image acquisition and display is determined by comparing the images acquired through digital imaging to those

by analog methods. X-ray radiograph is the term given to high resolution projection images. There are other alternatives for image acquisition, such as computed radiography (CR) systems. For CR systems, the image receptor is a photostimulable phosphorous plate in the form of a cassette, which absorbs and stores a significant portion of the incoming X-ray energy by trapping electrons and holes in elevated energy states (Aach et al. 1999). This energy can be read by scanning the plate with a laser beam.

Unfortunately, in both analog and digital systems, the quality of acquired images is diminished by system properties. For instance, limitations of contrast and resolution, detector sensitivity, and unwanted detector offsets may arise (Aach et al. 1999). Fortunately, digital systems prove to be superior to analog systems due to having the capabilities of counterbalancing these degradations with suitable processing approaches (e.g., offset correction, gain correction, modulation transfer function (MTF) restoration). As an added bonus, any exposure issues are eliminated thanks to the image receptors having approximately four orders of magnitude and the option of digitally adjusting the image's intensity range. Lastly, methods such as unsharp masking and harmonization can be used to enhance any relevant detail (Aach et al. 1999). Note that this enhancement is in regard to less important information in the image. To illustrate the effects that enhancement techniques can have, Figure 6a is a radiograph of a foot, and Figure 6b is the enhanced version of the foot.

Harmonization and then unsharp masking were applied for the enhancement of Figure 6a. Harmonization amplifies middle and high frequencies relative to low ones to make specific image details more visible. Unsharp masking applies an additional amplification of high spatial frequencies to give an image a sharper appearance (Aach et al. 1999). Therefore, it can be seen that Figure 6b is in fact an enhancement of Figure 6a.

X-ray fluoroscopy is a real-time dynamic X-ray imaging modality that allows a physician to monitor online clinical procedures such as catheterization (i.e., diagnosing and treating cardiovascular conditions by inserting a long, thin tube into the body) or injection of contrast agents (i.e., agents that enhance the visibility of blood vessels) (Aach et al. 1999). An illustration of the front end of a fluoroscopy image detection system is provided in Figure 7. The X-ray image intensifier (XRII) is a vacuum tube containing an input screen directly attached to a photocathode, electron optics, and a phosphorous screen output window. Images are detected by a fluorescent cesium iodide (CsI) layer on the input screen. This is where incoming x radiation is converted to visible photons leading to the photocathode. The photoelectrons accelerate as they pass through the electron optics, and visible images are eventually picked up by the camera. The last step of the system is amplification. For the digitization of 8-bit images, amplification is greater for smaller signal amplitudes than for larger ones in order to enhance any dark parts of the images (Aach et al. 1999).

(Figure 7 has been removed due to copyright restrictions.
http://electronicimaging.spiedigitallibrary.org/article.aspx?articleid=1097476)

The effects of noise are present in the majority, if not all, of image acquisition systems. For fluoroscopy image detection, there are two types of system-internal noise sources: fixed pattern noise and signal shot noise. Fixed pattern noise is caused by inhomogeneities of the XRII output screen. Signal shot noise is generated by the discrete nature of the conversion of information carriers, e.g., from luminescence photons into electrons in the XRII photocathode and the camera (Aach et al. 1999).

Even though XRII/TV camera systems have provided excellent image quality, there are still associated disadvantages. For one, these systems are particularly heavy and large in size. This makes the employment of these systems for bedside imaging with mobile fluoroscopy systems infeasible. Besides physical downsides, the systems suffer from electronic issues (e.g., vignetting, veiling glare, and geometric distortion). Vignetting is a phenomenon that results in a significant decrease in output image intensity toward the circular image boundary, which is caused by the convex shape of the XRII input screen (Aach et al. 1999). Veiling glare refers to disturbing light in the output images of an XRII caused by scatter of X-ray quanta, photoelectrons, and light photons inside the XRII, which reduces contrast (Aach et al. 1999). Vignetting and veiling glare issues can be digitally neutralized, but the best course of action would be to prevent these issues in the first place. These problems can be prevented with an all solid-state image sensor independent of electron and light-optical components, thus resulting in negligible effects of vignetting, veiling glare, and geometric distortion (Aach et al. 1999).

The paper by Alexander et al. (2012) discusses various image acquisition systems, specifically computed tomography, computed radiography, and film-screen radiography,

for detecting canine pulmonary nodules (i.e., swelling/lumps in the lung area). Computed tomography (CT) systems are commonly used in veterinary specialty practices to evaluate the lungs. CT is sometimes the only method used to evaluate the thorax (i.e., part of body between neck and abdomen) and is especially useful for patients that are already undergoing CT scanning for the evaluation of tumors. For the detection of pulmonary nodules, CT is more accurate than radiography due to its superiority of contrast, noise, and a lack of overlying anatomic noise (Alexander et al. 2012).

In the majority of veterinary practices, computed radiography (CR) replaced film-screen radiography due to film-screen radiography being more obsolete. CR also outperforms film-screen radiography in evaluating small structures (e.g., pulmonary nodules) because of the possibility of scanning a wider range of sizes and the ability to manipulate radiographic images (Alexander et al. 2012). Besides comparing the different systems to each other for the detection of pulmonary nodules, the authors compared nodule size and associated characteristics. An additional study on the experience level, confidence, and accuracy of the diagnoses of the analysts was performed.

Twenty-three client-owned dogs were chosen for the study, with 24 examinations completed. Of the 24 examinations, 13 were prospective (i.e., dogs had lumps) and 11 were retrospective (i.e., dogs had no lumps). Three of the patients were excluded from the study. One dog underwent surgical removal of liver cancer and had both prospective and retrospective examinations. Another dog was excluded due to the nodules having a bronchial distribution that made them very difficult to differentiate from each other (Alexander et al. 2012). Of the remaining 21 dogs, 7 were golden retrievers; 2 each were

German Shepherds, Bernese Mountain Dogs, Labrador Retrievers, and mixed-breeds; and six were of various other breeds. The dogs ranged from 6-13 years of age. All dogs had at least a three-view thoracic CR examination and a CT examination of the entire thorax within 72 hours of each other. Each image was read by two board-certified veterinary radiologists to create an efficient system for classification. (Alexander et al. 2012).

For each imaging modality, each study was classified as either positive or negative, where 0-3 represented definitely negative, probably negative, probably positive, and definitely positive, respectively. Ultimately, any result identified as probably negative or probably positive was classified as negative or positive, respectively. The lung tissue was divided into 6 different regions for nodule size and number: right and left middle (RM, LM), right and left cranial (RCr, LCr), and right and left caudal (RCd, LCd). Nodule characteristics were classified based on the following: definition (1 = well, 2 = intermediate, 3 = poor), contour (1 = regular/smooth, 2 = irregular/lobulated), shape (spherical, oval, polyhedral, other), areas of increased opacity/attenuation representing mineralization (present/absent), and cavitation and/or bronchogram (present/absent) (Alexander et al. 2012). Besides the two radiologist observers chosen for this study, four additional observers were recruited to further assess any findings.

Table 2 lists the results of the presence or absence of pulmonary nodules in the 21 dog subjects. From the 12 dogs that only had CR and CT testing, 7 had negative CR and CT and 5 had positive CR and CT; there was no disagreement between CR and CT for these two case types. From the other 9 dogs that also had film-screen radiographs performed, only one dog had a disagreement between modalities; CR found a single

pulmonary nodule whereas film and CT did not.

(Table 2 has been removed due to copyright restrictions.
http://onlinelibrary.wiley.com/doi/10.1111/j.1740-8261.2012.01924.x/full)

The authors found that the greatest number of nodules were detected with CT, as they expected. The majority of the nodules identified by CT were identified in the RM, RCd, and LCd regions. This may be due to the fact that the RCr and LCr regions are smaller than the other four, thus nodules would not be as easily identifiable (Alexander et al. 2012). Interestingly, there was no difference in the number of nodules between CR and film-screen radiography. More nodules were expected to appear with CR, because CR has the benefit of utilizing edge-enhancement filters and has the flexibility of image manipulation to improve the contrast and resolution of images. Older studies in humans have similarly shown little to no difference between digital and film-screen radiography in the detection of pulmonary nodules (Woodard et al. 1998; Månsson et al. 1999). CT detected significantly smaller nodules, and the minimum nodule size detected by CR and films was the same. Radiography should be inferior in detecting smaller nodules, particularly in areas of higher anatomic noise (Ehsan et al. 2003; Nemanic et al. 2006). The smallest nodule detected by CT, CR, and film-screen radiographs was 2 mm, similar to a minimum diameter of 3 mm reported on plain films (Armbrust et al. 2005).

Nodule characteristics resembled those commonly described for pulmonary metastases, a type of lung cancer, found in dogs (Burk and Feeney 2003; Nemanic et al. 2006). The majority of nodules were well defined and spherical. The authors expected the

nodules to be better defined with CT because of a lack of confounding overlying vascular structures and age-related changes with functional tissue of an organ (Alexander et al. 2012).

In conclusion, all three image acquisition systems performed similarly in determining the presence or absence of pulmonary nodules. However, for the radiographs, a false-negative diagnosis is a limitation that is associated with solitary nodules (Alexander et al. 2012). As a whole, CT detected a larger number of nodules than the other systems, and the observers had the strongest accuracy and agreement on the images acquired from this system.

2.2.1.2 Industrial Usage

The next two works provide substantial details on the industrial usage of X-ray acquisition systems:

1. Industrial Applications of Computed Tomography – De Chiffre et al. (2014)
2. X-ray Microtomography Applications for Quantitative and Qualitative Analysis of Porosity in Woven Glass Fiber Reinforced Thermoplastic – Madra et al. (2014)

The work by De Chiffre et al. (2014) reviews the industrial applications of X-ray computed tomography (CT) scanning systems. As reported by Kruth et al. (2011), the first CT scanner was built for medical imaging by Nobel Prize winner Godfrey Hounsfield in 1969. CT became a popular technique for non-destructive testing (NDT) and analysis of materials since the 1980s. In the industry, computed tomography scanning is useful for detecting flaws, such as cracks and voids, in various materials. This type of scanning is

also beneficial for analyzing a material's particles. Other than being non-destructive, CT scanning is non-invasive and is the only up-to-date technology that allows for measuring the inner and outer geometry of a component while keeping the specimen perfectly intact. CT systems can be considered the third step in the development of revolutionizing coordinate metrology (i.e., the process of calibrating and using physical measurements to quantify the size of a given object or the distance from it). The first step was the introduction of tactile 3D coordinate measuring machines (CMMs) in the 1970s, and the second step was the creation of optical 3D scanners in the 1980s (De Chiffre et al. 2014). An overview of the main advantages and disadvantages of CT scanning are presented in Figure 8.

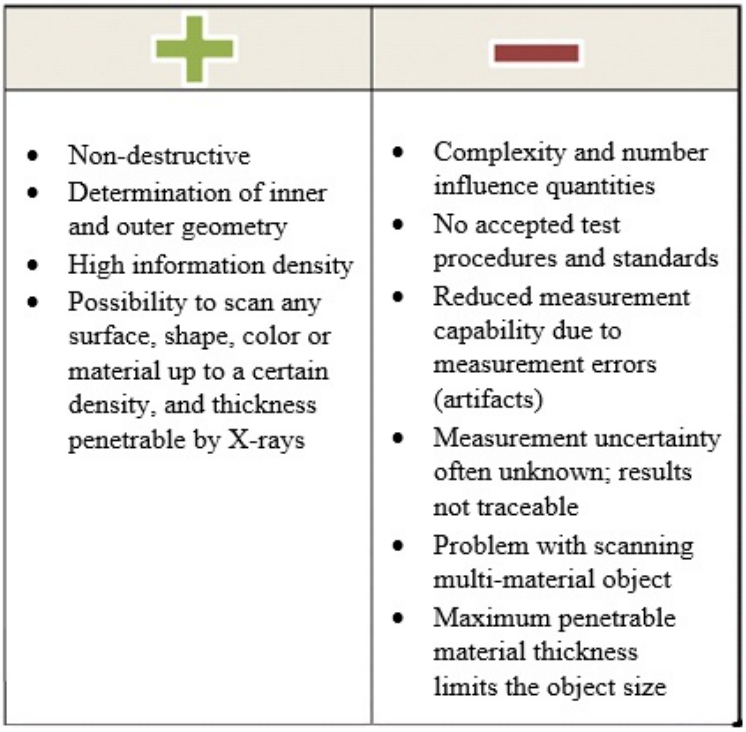

Figure 8. CT scanning advantages and disadvantages (Müller 2013).

The usage of CT systems increased tremendously following the development of the first CT system for human-medical purposes. In fact, research and development of NDT for industrial work in the 1980s was an essential component in identifying pores and microstructural defects (flaws) of materials (De Chiffre et al. 2014). Testing of two-dimensional defects extended to electronic production, i.e., printed circuit boards (PCBs). However, application of CT scanning for three-dimensional measurements posed accuracy issues. Thankfully, this issue was solved in the early 2000s; a solution was proposed to apply a conventional three-dimensional coordinate system for traceability and calibration (De Chiffre et al. 2014). The first coordinate measuring machine with X-ray sensor capabilities was developed by Werth Messtechnik in 2005, where the findings were presented to the market during the international fair "Control" in Stuttgart, Germany (De Chiffre et al. 2014).

Currently, X-ray systems are commonly used for medical applications and are an emerging market for industrial use. As a result, the needs and features of X-ray systems are different for these two separate markets. Figure 9a and Figure 9b represent the revenue estimations in the X-ray market from 2009-2017 and the global distribution of these systems, respectively. From Figure 9a, the revenue from X-ray systems in 2009 was estimated to be $344.2 million and was predicted to reach $591.9 million in 2017, thus showing great growth and interest in this field of work. Figure 9b shows that the majority of systems are installed in North America (32.2%) followed by Europe (27.5%) and the countries composing the Pacific region (27.3%), most notably Japan, China, India, and Russia (De Chiffre et al. 2014). The peak of interest in these systems over the past few

decades has opened up many opportunities in other branches of the industry. One such example is the food industry. CT scanning is favorable for packaging lines so that the contents of sealed packages, preserving jars, or cans can be checked to see if there are any contaminants contained within. Another instance is for butcheries and food processing factories where each piece of meat can be scanned to accurately calculate fat content to set fair market prices (De Chiffre et al. 2014).

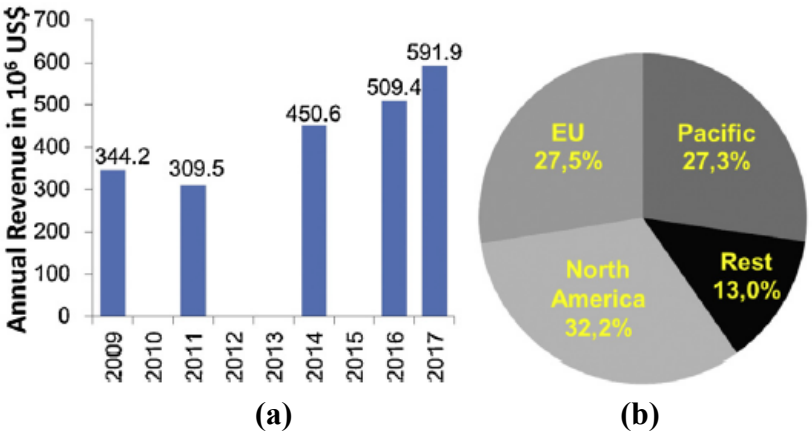

Figure 9. Market data. (a) Revenue estimations in the X-ray market, (b) Global distribution of CT systems (De Chiffre et al. 2014).

Many people have developed different types of CT systems due to the peak of interest in said systems. Two commonly used types are clinical CT scanners and CT systems for material analysis and implementation for the industrial market. For clinical CT scanners, the X-ray source is rotated continuously around a stationary material or patient (if for medical usage) to obtain tomographic images representing slices or portions of the material or patient. Research on increasing the efficiency of clinical CT scanners has been

ongoing for over four decades (Hsieh 2009). CT systems for material analysis and industrial purposes work differently than clinical scanners. Instead of the X-ray unit moving around a stationary item, the X-ray unit is kept stationary while the object being scanned is rotated. Also, since resolution and accuracy requirements are different between these two types of scanning systems, scanning parameters usually differ significantly (Kastner 2012). The resolution and accuracy for industrial CT systems can be adjusted by moving the axis of rotation supporting the object either closer to the source or detector (Kruth et al. 2011). This cannot be done for standard clinical scanners since the rotation axis is located between the source and detector. Another difference is that most CT systems, unlike clinical scanners, apply cone beam geometry and flat panel detectors to tremendously reduce scanning time while providing good image quality (Goebbels and Zscherpel 2011).

The five major factors affecting the effectiveness of CT systems are scanning time, measuring range, maximum penetrable material thicknesses, resolution, and accuracy. Contrary to coordinate measuring machines (CMM), the scanning time of CT systems is independent of the number of features that the system measures (Figure 10). The figure shows that tactile CMM considerably increases in scanning time when the number of features increases, whereas video CMM does not show as abrupt of a change. Scanning time is dependent on multiple parameters, including exposure time, number of projections, and performance of data processing (Kruth et al. 2011). Typical scanning time for industrial CT systems has shown to range from a few minutes to a few hours (Christoph and Neumann 2011).

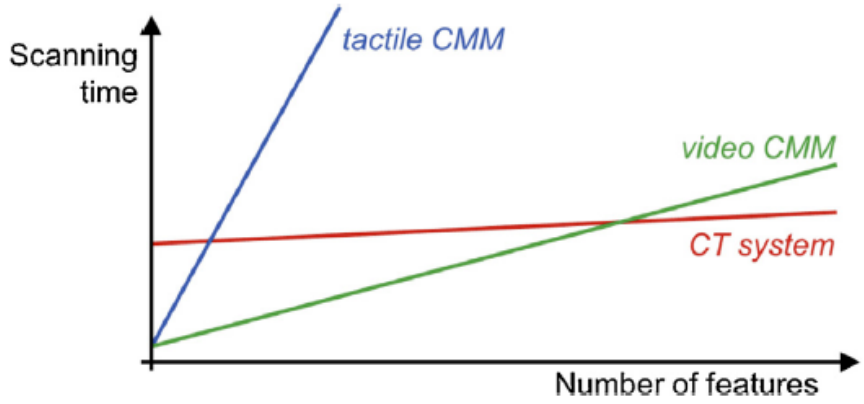

Figure 10. Scanning time vs number of features for CMM and CT systems (De Chiffre et al. 2014).

The measuring range capabilities of CT systems vary depending on the type of CT system being used. The dimensions of objects for scanning are limited by the measuring volume between the source and detector, system magnification, and the maximum penetrable material thickness (De Chiffre et al. 2014). The maximum penetrable material thickness that is penetrable by X-rays depends on the material's attenuation coefficients and the X-ray photon energy (De Chiffre et al. 2014). Table 3 provides examples of typical values for common materials. To optimize the scanning step, the object of interest should be oriented in a way to reduce the maximum penetrated material thickness. Optimizing orientation also involves minimizing the variation of penetration depth during object rotation in order to avoid pixel saturation or extinguishment in X-ray projections (Weckenmann et al. 2008; Müller 2013).

Table 3. Maximum penetrable material thicknesses for common industrial materials (Weckenmann et al. 2008).

X-ray voltage	130 kV	150 kV	190 kV	225 kV	450 kV
Steel/ceramic	5 mm	<8 mm	<25 mm	<40 mm	<70 mm
Aluminum	<30 mm	<50 mm	<90 mm	<150 mm	<250 mm
Plastic	<90 mm	<130 mm	<200 mm	<250 mm	<450 mm

Resolution is affected by many factors which influence CT reconstructions, including performance of the detector, number of projections, focal spot size of the X-ray source, reconstruction algorithms, and data post-processing (De Chiffre et al. 2014). Systems with a focal spot size greater than 0.1 mm are referred to as conventional or macro CT. Microfocus systems (μCT) have a spot size of a few micrometers. Nanofocus systems (nanoCT) can reach sizes as small as 0.4 μm (Heinzl 2009; Kastner 2011). Synchrotron CT (sCT) can reach 0.2 μm resolution which can be improved to 0.04 μm resolution when Kirkpatrick-Baez optics are applied (sCT+KB) (Requena et al. 2009). Figure 11 shows these values in a plot of spatial resolution versus object size. Lastly, CT accuracy is dependent on the specific object being measured and the specific parameters chosen for the measurement process (De Chiffre et al. 2014).

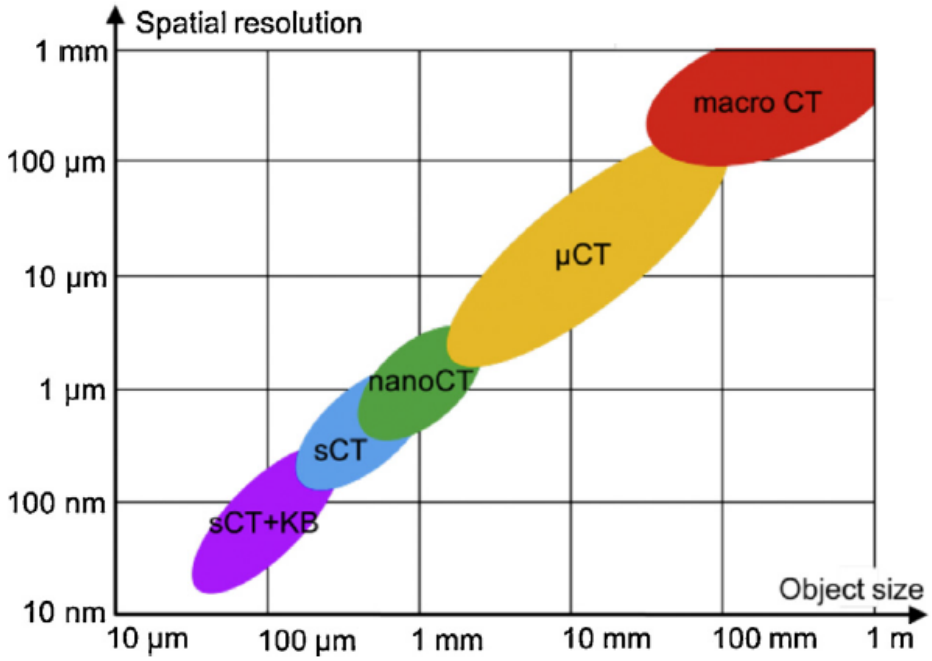

Figure 11. Typical spatial resolutions and object sizes for different CT systems (De Chiffre et al. 2014).

The manufacturing industry is dealing with the issue of products having shorter life cycles while the variety of the products is increasing, resulting in higher costs and a greater need for time-efficient labor. Industrial X-ray CT systems would then contribute to benefiting the industrial development and production chain. CT systems can be used for the inspection of goods or products while keeping them completely intact. Various components assembled together can be inspected since there is no guarantee that the assembled item will properly function, even if each individual component is working prior to assembly. Figure 12 depicts an example of multi-material assembly of an insulin pen.

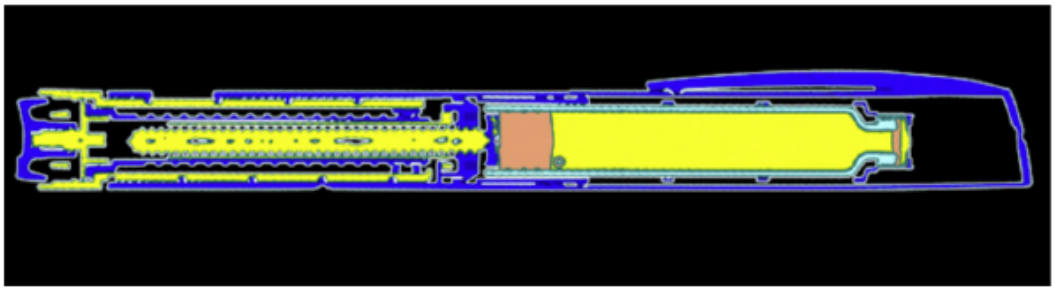

Figure 12. Multi-material assembly: drug delivery device (insulin pen) including components of different polymeric materials (Sørensen 2012).

In conclusion, there is a large number of industrial applications for X-ray computed tomography scanning. This market may not be old and proven, like for medical purposes, but industrial applications are rapidly growing, e.g., in the manufacturing industry, electrical devices, and food industry. Future concerns entail extending CT practicality to a greater range, improving worker operation, having a better economy of the measurements process, and having a faster cycle time for a measurement according to the cycle time of manufacturing (De Chiffre et al. 2014).

Madra et al. (2014) obtained a three-dimensional representation of woven glass fiber reinforced thermoplastic composite by means of X-ray microtomography. Polymer matrix composite materials have recently become very popular and were previously inaccessible due to high manufacturing and development costs. Limitations in controlling material parameters resulted in over-constrained designs and poor performance (Madra et al. 2014). Over the last few decades, advancements in the comprehension of the physics and mechanics driving materials has allowed for the invention of methods capable of processing thermoplastic matrices. Further studies on the manufacturing processes and

mechanical behavior of thermoplastic composites will allow for the better design of structural parts to optimize the performance to cost ratio (Madra et al. 2014). Most notably, thermoplastics reinforced with glass fiber are of great interest for the aeronautical and automobile industries.

The classical Archimedes method is a technique for measuring porosity content. The technique is useful for testing large amounts of material but not for micro-pores, which cannot be identified by the method. This leads to a more precise approach for porosity estimation, being optical or Scanning Electron Microscopy (SEM). Many other methods exist as well, such as ultrasonic detection and active thermography (Mayr et al. 2011). X-ray microtomography provides complete three-dimensional representations of scanned structures. Related studies help with identification of various mechanical parameters, monitoring of crack propagation, or characterization of phase composition for different types of materials (Scott et al. 2011; Eric et al. 2012; Cosmi and Bernasconi 2013; Etaati et al. 2013; Seltzer et al. 2013). With constant development of image processing tools, both two-phase (material+air) materials and multiphase composites can be examined (Maire et al. 2001; Maire et al. 2007; Requena et al. 2009; Scott et al. 2011). The authors applied various image segmentation techniques to acquire segmented images for analyzing porosity on the global, layer, and single yarn scale.

A laminate of thermoplastic reinforced with glass fiber, supplied by *Ecole des Mines de Douai*, France, was consolidated under pressure to prepare for testing. Two materials were considered: P1 with bulk porosity of 0.45% measured by the Archimedes method and P2 with bulk porosity 1% (Madra et al. 2014). Six specimens were cut from

manufactured sheets (Figure 13).

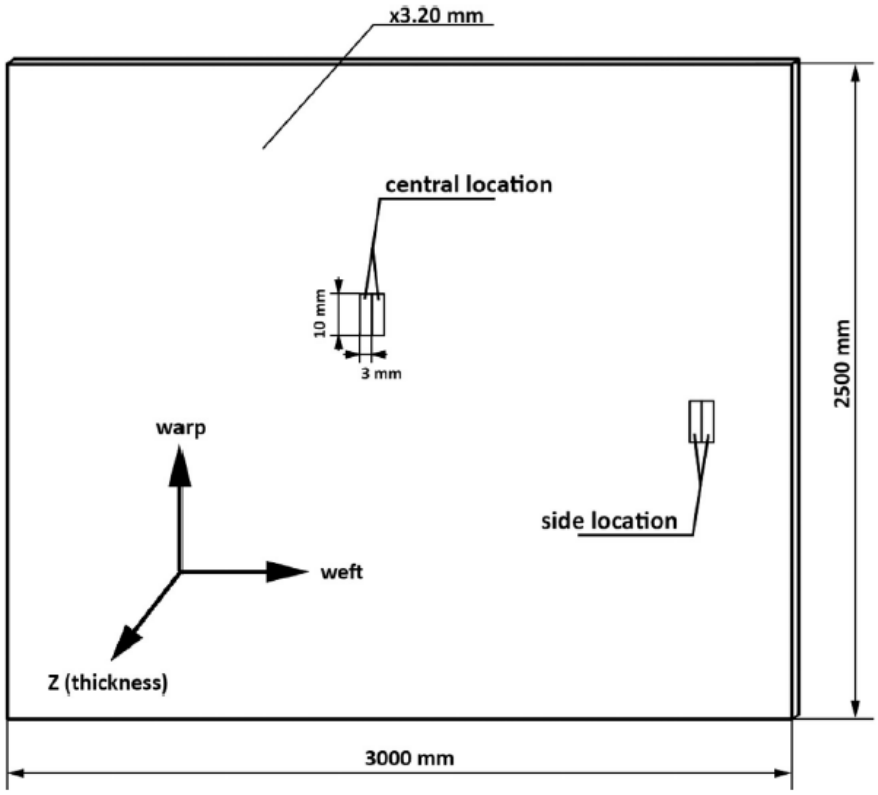

Figure 13. Location and dimensions of tested samples (Madra et al. 2014).

Microtomography scans were obtained at *Laboratoire Mateis* of *INSA Lyon*, France, using X-ray microtomography (Salvo et al. 2003). Images had attenuation coefficients relating to grayscale, with 0 representing white/fibers and 255 representing black/porosity. Any further image processing was completed by ImageJ/Fiji (Schindelin et al. 2012). Unfortunately, the results showed that standard thresholding methods would deem unsatisfactory results due to noise and low contrast caused by low resolution. Therefore, a learning algorithm, Trainable Weka Segmentation, was used. Training

features, such as Gaussian blur and Hessian were applied. Specifically for this study, images were segmented into three or four classes (Madra et al. 2014). Figure 14 illustrates the segmentation results from the techniques tested: thresholding and median filtering and the Trainable Weka Segmentation method. The black, gray, and white regions on the images correspond to resin, fibers, and porosity, respectively. Table 4 lists the segmentation average and standard deviation errors for the three different techniques utilized, with the third being standard thresholding without any additional filtering applied. Overall, the Weka Segmentation provides the best results with relatively low standard deviation. The thresholding without filtering method yields the greatest noise and misclassifies areas as porosity. Even though median filtering decreases the present noise, the technique still does not reach the effectiveness of learning algorithms (Madra et al. 2014).

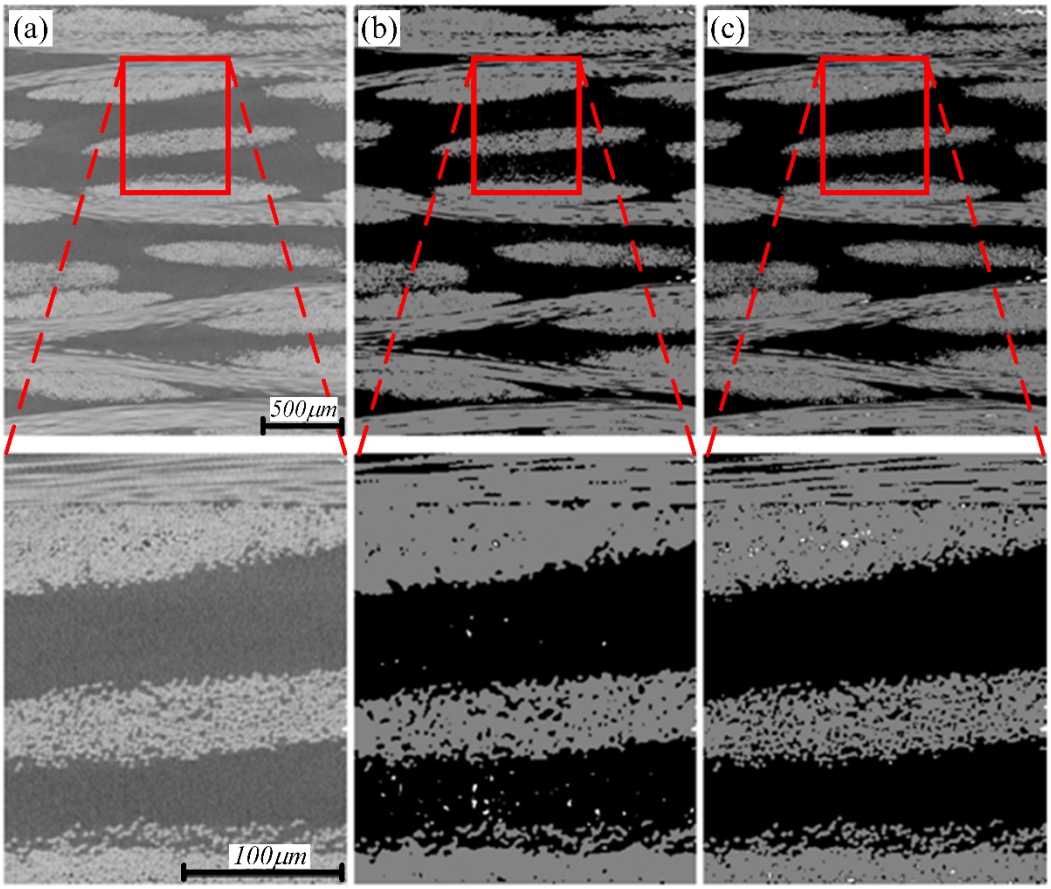

Figure 14. Comparison of segmentation results: (a) original slice from microtomography; (b) segmentation by thresholding and median filtering; (c) trainable weka segmentaion (Madra et al. 2014).

Table 4. Comparisons of errors in segmentation for various methods (Madra et al. 2014).

Method	**Porosity error**		**Fiber error**	
	Average (%)	Standard deviation (%)	Average (%)	Standard deviation (%)
Thresholding	80.1	30.4	42.3	23.7
Thresholding + median	53.6	37.1	38.2	19.2
Weka segmentation	9.8	4.7	12.3	6.7

The results provide a positive outlook for the future of deriving indicators for optimizing the manufacturing process. These indicators can be obtained from a porosity distribution at various scales (i.e., global, layer, and single yarn scales). The quality of the microtomography results as well as the segmentation process parameters directly affect microtomography quality (Madra et al. 2014). The implementation of learning algorithms allowed the scans of the glass fiber reinforced thermoplastics to yield acceptable results. Future work entails testing the learning algorithms on less contrasted images that have similar attenuation coefficients (e.g., carbon fiber).

2.2.1.3 History of Image Acquisition Systems

The final paper reviewed provides a summary on the history, growth, and future of image acquisition systems from 1895 to 2008:

1. The History and Future of X-ray Microscopy – Kirz and Jacobsen (2009)

The usage of image acquisition systems was made possible following Wilhelm Röntgen's discovery of X-rays in 1895 (Röntgen 1896). Pioneers of the image acquisition field understood the nature of X-rays and developed absorption and emission spectroscopy (Kirz and Jacobsen 2009). The pioneers also were able to establish point-projection microscopy with a resolution of a few microns (Malsch 1939). Even the famous physicist Albert Einstein worked with X-ray technology, more specifically X-ray optics. Einstein (1918) suggested that the index of refraction (i.e., deflection of X-ray beams off of an object) in most materials should be slightly less than 1.

Following the conclusion of World War II in 1945, numerous people became

interested in working with X-ray microscopy. For instance, Arne Engström of Sweden developed the technique of quantitative elemental imaging (Engström 1946). At Cambridge University, Cosslett and Nixon began to work in point projection X-ray microscopy (Figure 15) and extended their expertise to X-ray analysis in the electron microscope (Cosslett 1959).

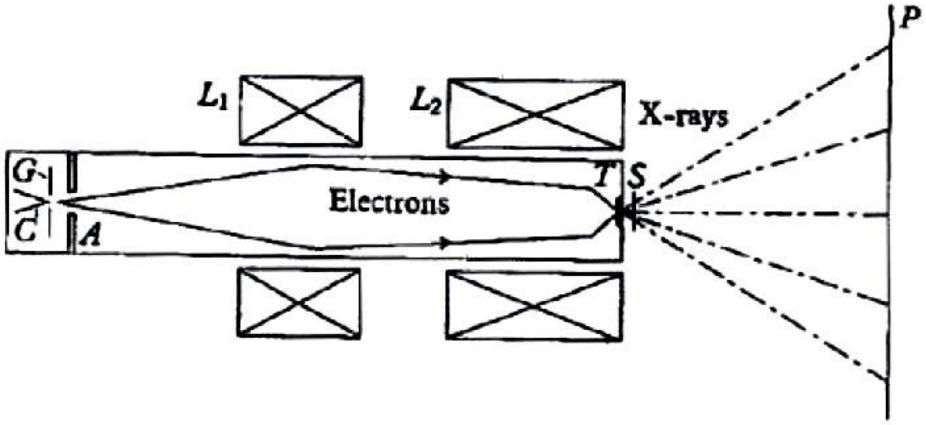

Figure 15. Point projection X-ray microscopy (Cosslett 1959).

The 1970s introduced the first light-based X-ray microscopes. Horowitz and Howell (1972) presented the first developed synchrotron-based X-ray microscope, as shown in Figure 16. The beam stop, located above the x,y transducer, absorbs emitted X-ray energy which reduces the scattering of the background, i.e., noise.

(Figure 16 has been removed due to copyright restrictions.
http://xrm.phys.northwestern.edu/research/pdf_papers/1972/horowitz_science_1972.pdf)

Attempts to create other variations of X-ray systems were made in the 1980s and early 1990s as a follow-up to the success of synchrotron light sources. In particular, a research group at King's College built a scanning transmission X-ray microscope (STXM) (Morrison et al. 1989). The first X-ray lasers were demonstrated at Livermore and Princeton (Trebes et al. 1987; Skinner et al. 1990; Da Silva et al. 1992), but were deemed unsuitable for microscopy experiments due to only outputting a few pulses per day.

At the time, the unfamiliarity and newness of the field of X-ray microscopy led researchers down a wrong path. Professor Baldini from Harvard teamed up with scientists at IBM for flash-contact-microscopy of blood platelets (Feder et al. 1985). One of the images of the blood platelets actually appeared in the Science section of *The New York Times* on January 15, 1985; however, the image was later discovered by the authors not to be an image but rather a platelet stuck on the detector.

X-ray microscopy reached a point in the early 2000s where there was no longer question of the practicality of this technology (Kirz and Jacobsen 2009). This technique has proven, through time, to be a reliable method for image acquisition. In fact, the community involved in such work (see Figure 17) has drastically grown in attendees and articles/abstracts written for conferences from 1983 to 2008. The growth was substantial for X-ray microscopy, where the technology's early potential (Horowitz and Howell 1972; Jacobsen 1992) was realized by new zone plates, compound refractive lenses, multilayer Laue lenses, and Kirkpatrick-Baez optics (Maser et al. 2004; Kang et al. 2006).

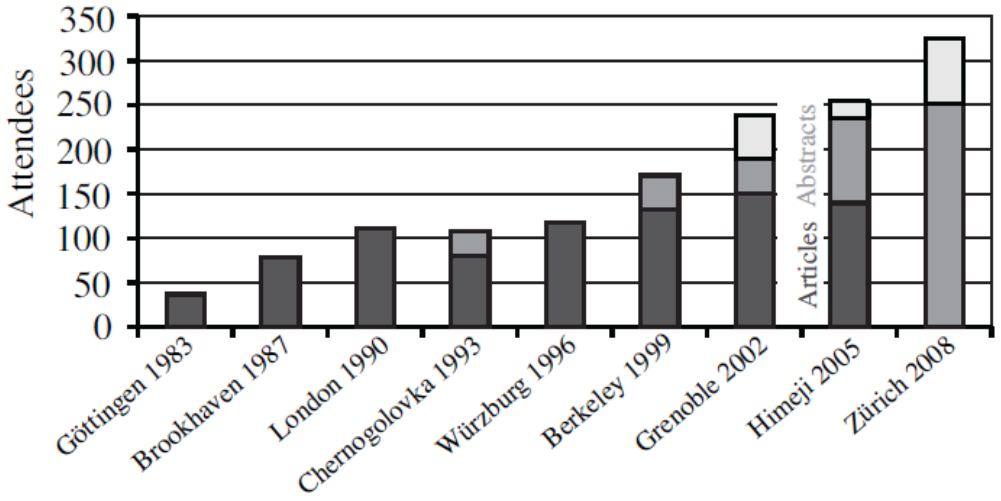

Figure 17. Attendees and articles/abstracts at X-ray microscopy conferences (Schmahl and Rudolph 1984).

The future of X-ray microscopy is a promising one; one specific area with the possibility of rapid growth is multi-dimensional microscopy. It is important to note that the added dimension, being space, energy, or time, requires extra precautions for preventing radiation damage (Kirz and Jacobsen 2009). One of the most important advantages of X-rays is their penetrating power, where much of the demand in applications is to examine samples too thick for electron microscopes (Sayre et al. 1976; Grimm et al. 1998; Jacobsen et al. 1998). As powerful as two-dimensional images are, they are not as informative as three-dimensional images, which can provide a better representation of the internal structures of the object scanned. One recent, major development in three-dimensional microscopy lies with the tomographic TXM instruments built by Xradia, which can provide images with features even smaller than 60 nm (Yin et al. 2006; Chen et al. 2008; Chu et al. 2008).

2.2.2 Global Image Segmentation

2.2.2.1 Two-Phase Image Segmentation Techniques

In order to provide further information on the development and testing of bi-level image segmentation techniques, this section contains literature reviews for the following four journal papers:

1. A Segmentation System Based on Thresholding – Kohler (1981)
2. A Threshold Selection Method from Gray-Level Histograms – Otsu (1979)
3. A New Method for Grey-Level Picture Thresholding using the Entropy of the Histogram – Pun (1980)
4. A Survey of Thresholding Techniques – Sahoo et al. (1988)

The first paper by Kohler (1981) proposes a segmentation algorithm for obtaining multiple threshold values. Generally, an image segmentation algorithm divides an image into sets of pixels referred to as regions. Successful segmentation is achieved when there is a high correlation between entities in the "real world" (e.g., objects) and segmentation regions. The algorithms can be classified into the following two groups: a group where regions are created in an image based on similar pixel characteristics or features and a group where edges in an image, corresponding to object discontinuities based on differences in pixel characteristics or features, are located. As previously mentioned in Chapter 1, a single threshold value is used to divide an image into two different regions. Some images with a clear foreground-background relationship have a single threshold value that detects all object boundaries. On the other hand, more complex images cannot

be expected to have a single threshold value that detects all object boundaries. In such a case, a segmentation algorithm is defined in terms of n thresholds rather than a single threshold, where the n thresholds are composed of n+1 classes, or regions.

There are three types of errors that can qualitatively deem the segmentation of an image unsuccessful (Figure 18). Type one error occurs when the segmented image contains boundaries that are not present in the "correct" segmentation and thus do not correspond to any real object discontinuities in the image. Type two error occurs when the segmented image misses edges that appear in the "correct" segmented image. Lastly, type three error occurs when an object boundary is in the proper location in the "correct" segmentation but is at a different location in the computed segmentation. These errors typically arise when there are inaccuracies with the transformation between a spatially continuous image and its discrete representation. In most cases, it is easier to correct type three errors than type one and two errors.

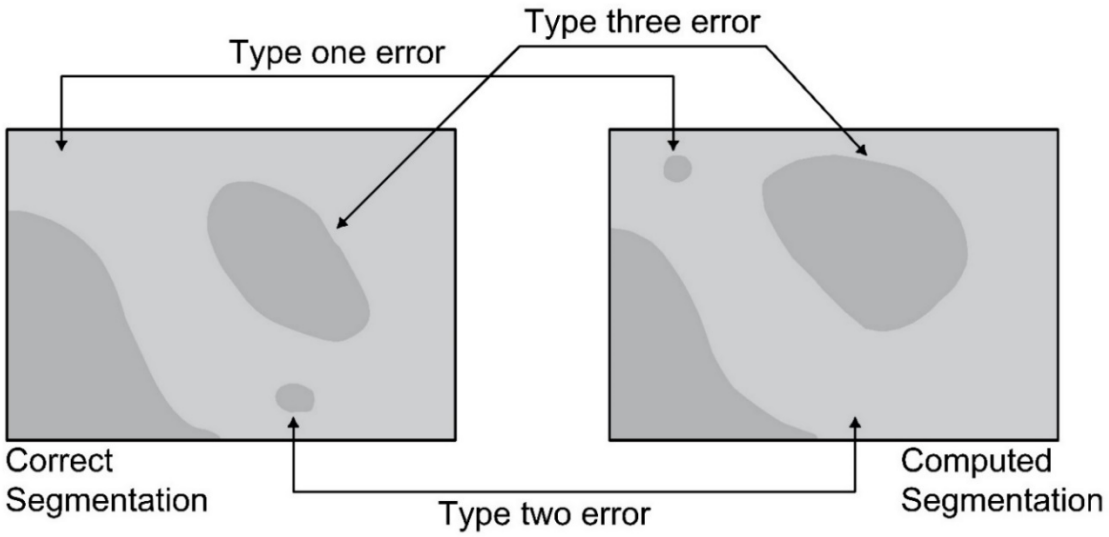

Figure 18. Possible types of errors associated with segmentation (Kohler 1981).

An optimal threshold value is one that minimizes the potential for any of the above errors, especially with types one and two being more common. Table 5 below lists seven (M1-M7) generally used threshold algorithms. The Standard Feature Histogram Method (M1) is one of the most used algorithms and is based on the assumption that different regions of an image are detected on the basis of feature activity, where different peaks in the feature histogram correspond to different image regions. However, there are two main concerns with using this method. If the valleys between histogram peaks are long and flat, threshold selection becomes difficult. Threshold selection is also difficult if the edge information in the image is not utilized (Kohler 1981). The Gradient Weighted Feature Histogram Method (M2) is another thresholding option that overcomes these limitations. Gradient information is added to the histogram through the reduction of the relative weight of histogram entries with high gradient magnitudes. This method would hopefully result in sharper peaks and valleys of the histogram.

(Table 5 has been removed due to copyright restrictions.
http://www.sciencedirect.com/science/article/pii/S0146664X81800159)

The second paper by Otsu (1979) proposed one of the most widely used global thresholding methods. An important step of image processing is the extraction of objects from the background through applying a threshold value determined from gray-level pixel information. According to Otsu (1979), the ideal threshold value would be an intensity value located at a sharp valley between peaks representing objects and the background of the gray-level intensity distribution. Though, the determination of this value is not that

simple; images can have histograms containing noise, the valleys could be flat and long, or the peaks can have significant differences in height. To ensure that these difficulties are overcome, Otsu created a method (Otsu 1979) that chooses an optimal threshold value through an automatic process without any prior knowledge of any pixel information; only the gray-level histogram is known.

The Otsu (1979) method calculates the optimal threshold through separability of the two classes (foreground and background) through minimizing within-class variance or maximizing between-class variance. In other words, the optimal threshold is located where the summation of the foreground and background spreads are at a minimum (Otsu 1979). Otsu (1979) ran several experiments to test the effectiveness of his proposed method. Figure 19a represents an original gray-level image of the character "A," Figure 19b is the resulting segmented image, Figure 19c is the original image's histogram with the selected threshold value marked with an arrow, and Figure 19d shows the results obtained by analysis. Figure 20 follows the same representation as Figure 19, with the original image instead being a texture. The original image in Figure 19 has 16 gray-levels (4-bit image) whereas the original image in Figure 20 has 64 gray-levels (6-bit image).

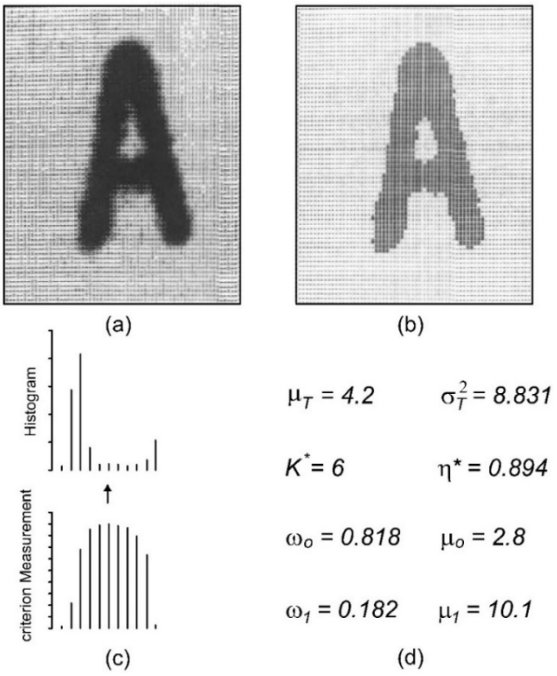

Figure 19. Proposed method applied to a character (Otsu 1979).

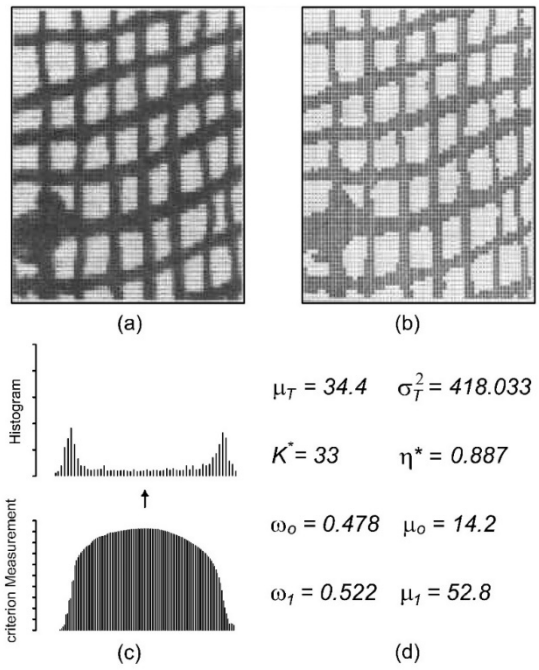

Figure 20. Proposed method applied to a texture (Otsu 1979).

The segmentation results prove that the Otsu (1979) method is simple to use and provides straightforward analyses for otherwise difficult images. An optimal threshold is stably, effectively, and reliably chosen, and this method also allows for other image properties to be determined rather than just a threshold value (e.g., estimation of class mean levels). Therefore, Otsu believed that this method should be viewed as a standard thresholding method, being quite possibly the simplest method for determining an automatic threshold value for a wide variety of images. For these reasons, the Otsu (1979) method is one of the chosen thresholding techniques for the geomaterial characterization of this work. The mathematical expressions for this method are presented in Chapter 3.

In the third paper, Pun (1980) realized the same disadvantages that arise with identifying thresholds for images containing noise, as Otsu (1979) did. Just how Otsu proposed a technique to combat these issues, Pun offered a method that he believed to be more global and efficient than the Otsu (1979) separability function. Also, as seen with the Otsu (1979) method, this proposed method does not require information about the image other than the gray-level histogram. The gray-level histogram is considered to depend only on the number of gray-levels and is hence independent of the image. An assumption is made that the relationship between the total pixels at a specific intensity value is statistically independent from the number of pixels at a nearby intensity value. For "realistic" pictures, this assumption is not necessarily true, but it is still utilized because the derivation of the thresholding algorithm is greatly simplified while still providing good results (Pun 1980).

The testing of the Pun (1980) algorithm began by obtaining experimental results

using two images with 256 gray-levels (8-bit images). Figure 21a represents the first image (a cameraman) and Figure 21b represents the second image (a building). Figures 22a and 22b correspond to the original gray-level histograms for these images with the selected entropic thresholds indicated on the histograms. Lastly, Figures 23a and 23b show the new, segmented images with the corresponding threshold values applied.

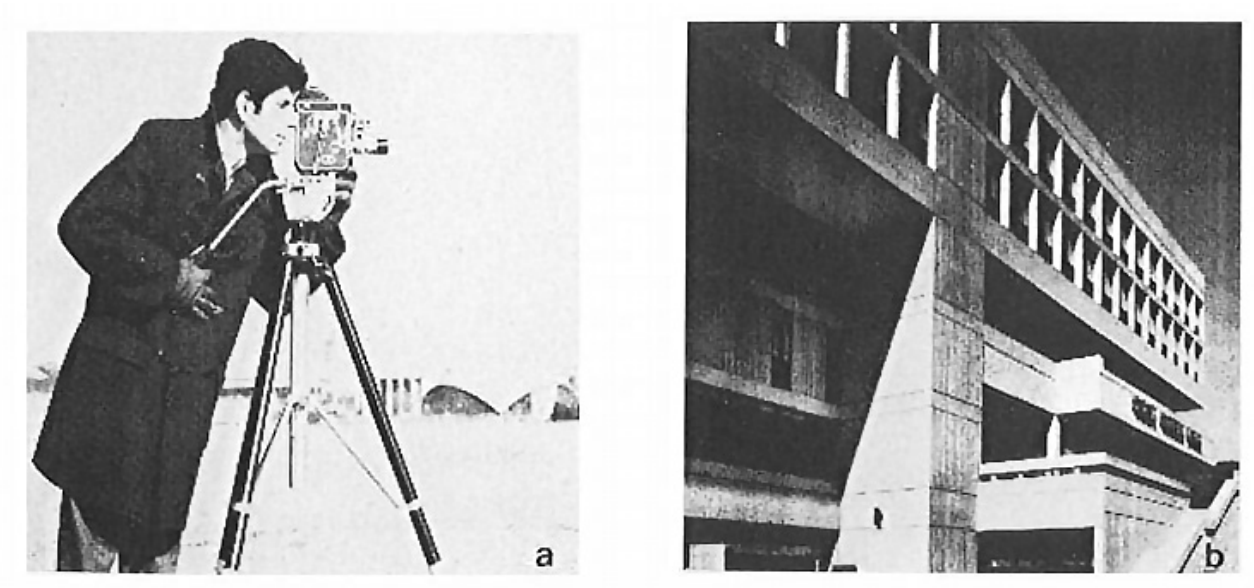

Figure 21. Original images used in Pun's study (Pun 1980).

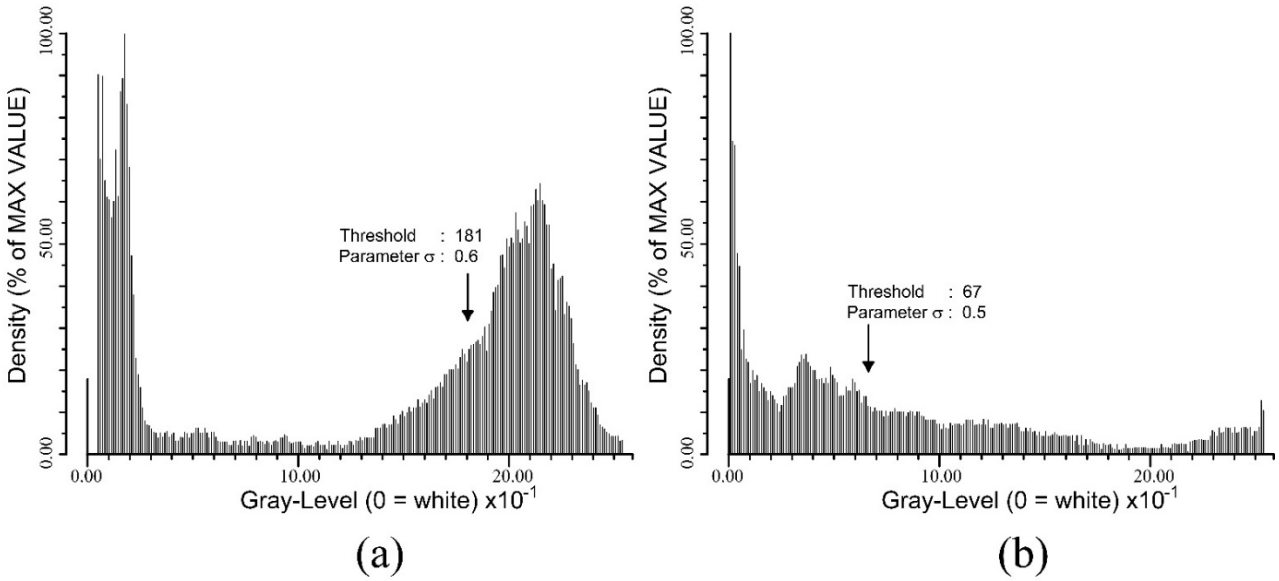

Figure 22. Gray-level histograms with selected entropic thresholds (Pun 1980).

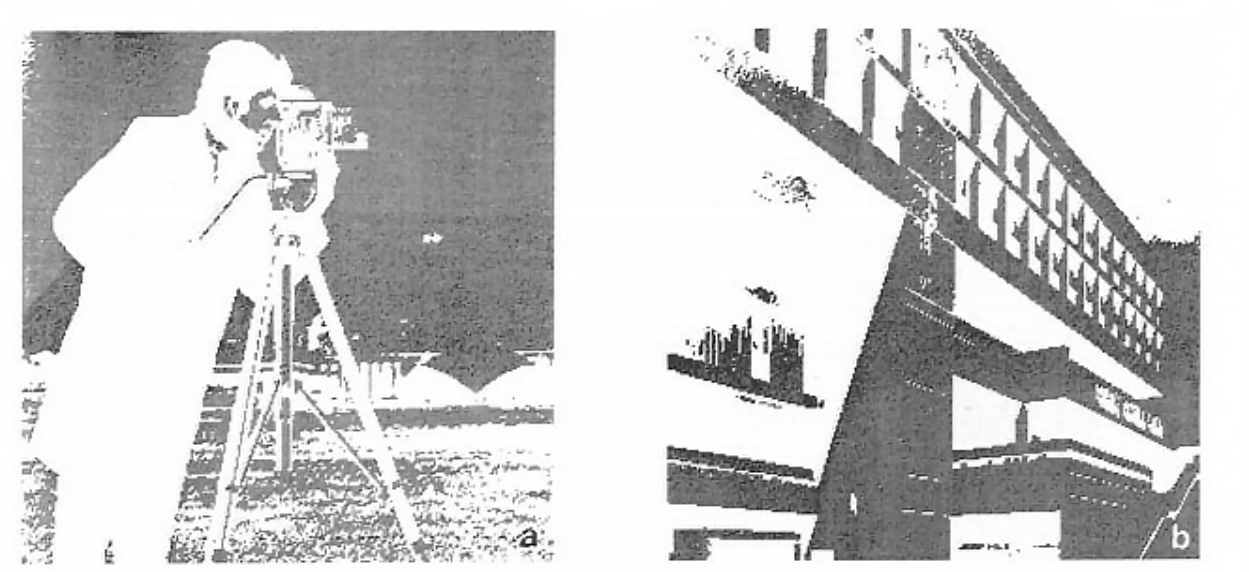

Figure 23. Segmented images with entropic thresholds applied (Pun 1980).

The results yielded through the Pun (1980) method show that Pun proposed a successful method of automatically selecting a threshold from a gray-level histogram through the concept of entropy. Generally speaking, the Pun (1980) method has several advantages. This method effectively and stably selects a threshold value from the global information of an image's histogram and the proposed algorithm is simple to implement while leading to decreased computational time. Thus, the Pun (1980) method is the second algorithm chosen for bi-level geomaterial characterization. As with the Otsu (1979) method, the equations for the Pun (1980) method are provided in Chapter 3.

The fourth paper by Sahoo et al. (1988) was very important since the remaining three thresholding techniques for bi-level image segmentation were chosen from this work. These methods, which will be discussed below, underwent a pre-analysis on a given set of test images from photographs of a portrait or natural scenes to prove their effectiveness. The global thresholding techniques analyzed in this work are divided into point-dependent and region-dependent techniques. A thresholding method is point-dependent if the optimal threshold is determined from the gray-level of each pixel. A thresholding method is region-dependent if the optimal threshold is determined from the local gray-level in the neighborhood of each pixel. Since the Otsu (1979) and the Pun (1980) methods were chosen for implementation, which are both point-dependent global thresholding techniques, the remaining three techniques were chosen from this same category. One such method, the p-tile method, assumes that an image consists of dark objects in a light background and that the threshold is determined through knowledge of the object area. The mode method is applicable to images with a bimodal histogram that has distinct objects

and background. The shortcoming of this method is that it cannot be applied to images with very unequal peaks or broad and flat valleys. Another subcategory of point-dependent techniques is the histogram concavity analysis method. This method can be applied when valleys are not present in the gray-level histogram of images, therefore overcoming some limitations of the mode method. Instead of focusing on valleys, the histogram concavity analysis method defines a good threshold value to be located at the "shoulder" of the histogram (Sahoo et al. 1988).

Even though the above methods proved to be successfully implemented, they were not considered as potential candidates for the remaining three techniques due to the ease and applicability of the following three methods. Two of the methods, suggested by Kapur et al. (1985) and Johannsen and Bille (1982), are included in the entropic methods subcategory of point-dependent techniques. For the Kapur et al. (1985) method, the optimal threshold value is determined by deriving two probability distributions of the classes from the original gray-level distribution of an image (Kapur et al. 1985). The Johannsen and Bille (1982) method finds the optimal threshold value by dividing the set of gray-levels into two parts to minimize the interdependence between them (Johannsen and Bille 1982). The fifth and final method used for analysis is the Kittler and Illingworth (1986) method. This technique determines the optimal threshold value by viewing the gray-level histogram as an estimation of the probability density function representing the gray-levels of the foreground and background pixels in an image. This method also assumes that these two categories of pixels are normally distributed with priori probability, mean, and standard deviation (Kittler and Illingworth 1986; Arifin and Asano 2006). Further details for these

three techniques are found in Chapter 3.

2.2.2.2 Multi-Phase Image Segmentation Techniques

This section contains the literature review of three journal papers supporting the evolution of multi-level image segmentation techniques:

1. A Recursive Thresholding Technique for Image Segmentation – Cheriet et al. (1998)
2. A Fast Algorithm for Multilevel Thresholding – Liao et al. (2001)
3. Multilevel Thresholding for Image Segmentation through a Fast Statistical Recursive Algorithm – Arora et al. (2008)

As mentioned previously, the Otsu (1979) method is considered to be a standard thresholding method, quite possibly one of the most reliable methods for determining the automatic threshold value for a wide variety of images. Not only can this method perform bi-level segmentation, but it can also be extended to perform multi-level segmentation as well (Otsu 1979).

The paper by Cheriet et al. (1998) presents a recursive approach that is an extension of the Otsu (1979) method for image segmentation. Over the last few decades, a substantial amount of research has focused on image segmentation and the creation of numerous methods. However, region-based and edge-based techniques are two main types that are widely used. Region-based techniques create boundaries for threshold selection by identifying areas of an image with pixels that have homogeneous properties. Edge-based techniques find local discontinuities between pixels and connect them to form boundaries

(Cheriet et al. 1998). Nevertheless, these types of techniques are complementary where practical usage of a type depends on the type of image being analyzed.

There is great interest in the study of image segmentation of document images (e.g., maps, engineering drawings, newspapers, and magazines) to allow for the simple extraction of relevant information. In the case of this study, the presented recursive approach utilizes region-based concepts to segment gray-level document images. As an extension of the Otsu (1979) method, the Cheriet et al. (1998) approach segments the brightest object at each repetition, ultimately leaving the darkest object in the image at the end of repetition. The recursive process is completed once no new peaks in the gray-level histogram can be found or when regions become too small. As an added bonus, the recursive process is not limited to a specific number of objects.

The Cheriet et al. (1998) technique was trained on 220 real-life bank checks to select an optimal value, S. S represents the criteria used to determine if two object classes have homogeneous properties or not. The training set of checks yielded an S value of 95%. Thus, if two classes do not share at least 95% of homogeneous properties (i.e., pixels belonging to the same object classification), than a threshold value is used to separate these classes. The performance of the technique was then tested on 505 different checks. Visual inspection was performed on these checks to create a classification system, as per Table 6. Table 7 lists the experimental results for both the training and testing sets of checks.

Table 6. Classification of 505 real-life bank checks (Cheriet et al. 1998).

Letter	Description
D	Dark background or dark handwriting
D+	Very dark background or very dark handwritten information
S	Simple and homogeneous background with no figures
C	Complex background with homogeneous figures considered as one object
C+	Complex background with nonhomogeneous figures considered as multiobjects
t	Thin handwritten information
T	Thick handwritten information
L	Very light handwritten information
L+	Light but not dark handwritten information

The results in Table 7 indicate that the training and testing check sets yield satisfactory results for when the target object for each recursion is the darkest object in the image. Interestingly, the Cheriet et al. (1998) method does not perform segmentation well when the target object is not the darkest object, as seen for the categories of 20 and 14 checks tested with 0% performance.

The second paper by Liao et al. (2001) proposes a faster version of the Otsu (1979) method to improve the computational efficiency for selection of multiple optimal thresholds. Normally, the Otsu (1979) method is very sufficient for real world images with regard to uniformity and shape measures (Sahoo et al. 1988). Although the Otsu (1979) method is highly effective, it has shown to take too much time, computationally, to be practical for multi-level threshold selection. The thorough search to maximize the between-class variance takes a greater amount of time as the number of classes in an image increase. The authors first analyzed how the Otsu (1979) method would be expanded for multi-level segmentation before the presented algorithm was created. Regardless of the number of classes being segmented, the summation of the cumulative probability functions of the

classes always equals one. Also, an image's mean is determined from the sum of the means of the classes weighted by the respective cumulative probabilities. With this information, the original between-class variance can be modified to find the optimal thresholds for separating all the classes of an image. As mentioned, the modification of the between-class variance does not speed up computational time, but portions of the modified between-class variance can be precomputed and stored in a "look-up table" to do so (Liao et al. 2001). The Liao et al. (2001) method uses the modified between-class variance in conjunction with the cumulative probability (zeroth order moment) and the mean (first order moment) of a class to determine the class's threshold value.

Table 7. Experimental results of presented recursive approach (Cheriet et al. 1998).

Number of Check Images	Image Size (MB)	Background/ Foreground Classification	Computation Time (s)	Performance Analysis
Training Set				
86	1.3	D S / t D+	16	99% - 100%
61	1.4	D C / t D+	18	96% - 100%
47	1.5	D C / {t, T} L+	21	85% - 100%
26	1.5	D C+ / {t, T} D+	23	73% - 84%
Testing Set				
68	1.4	D S / {t, T} D+	17	99% - 100%
74	1.5	D S / {t, T} L+	19	96% - 100%
58	1.4	D S / {t, T} L	16	85% - 100%
33	1.5	D + S / {t, T} {D+, L+, L}	18	73% - 84%
47	1.4	D C / {t, T} D+	20	98% - 100%
56	1.4	D C / {t, T} L+	19	93% - 100%
31	1.5	D C / {t, T} L	21	81% - 100%
21	1.4	D + C / {t, T} {D+, L+, L}	20	59% - 81%
48	1.6	D C+ / {t, T} D+	23	98% - 100%
35	1.6	D C+ / {t, T} L+	22	90% - 100%
20	1.4	D C+ / {t, T} L	19	0%
14	1.5	D+ C+ / {t, T} {D+, L+, L}	13	0%

Liao et al. (2001) tested their method on four different images: an F16 jet, a house, a woman named Lena, and peppers. These images are displayed in Figure 24, and the four images have 256 gray-level intensities (8-bit images). Table 8 lists the results of the method's threshold values for when 2, 3, 4, and 5 classes are segmented. Also, the table

provides the computational time for each image with both the Otsu (1979) method (with and without recursion) and the proposed algorithm. For qualitative purposes, Figures 25-28 show the bi-level, tri-level, four-level, and five-level segmented images based on the threshold values of Table 8.

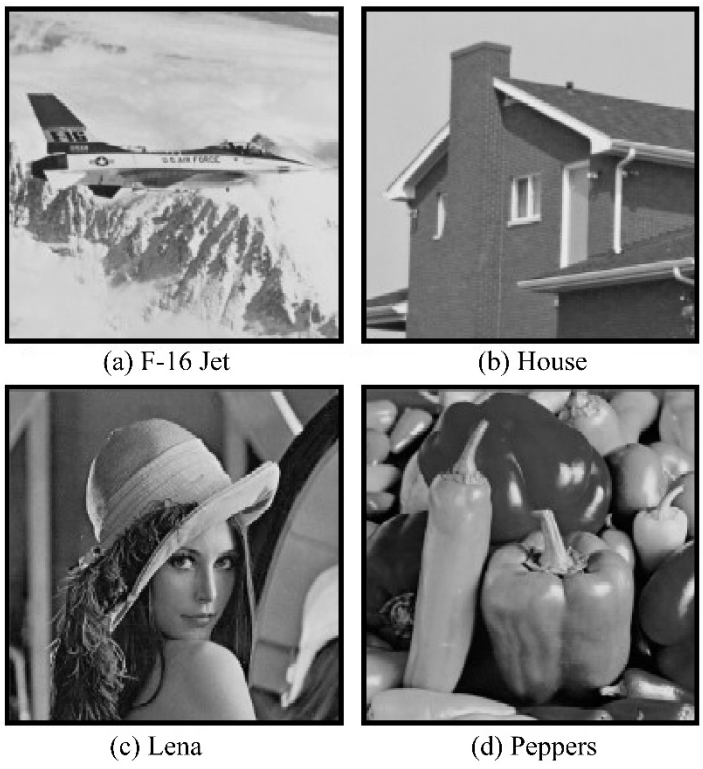

Figure 24. Four test images of study (Liao et al. 2001).

Table 8. Threshold values and computational times results of test images (Liao et al. 2001).

Images	Thresholds				Computation Times							
					Otsu's Method				Proposed Method			
					With Recursion Without Recursion							
Classes	**2**	**3**	**4**	**5**	**2**	**3**	**4**	**5**	**2**	**3**	**4**	**5**
F16 Jet	156	111 172	96 149 191	86 130 171 202	<1 s <1 s	<1 s 1 s	5 s 70 s	6 m 1 h	<1 s	<1 s	1 s	37 s
House	147	88 154	86 130 177	64 92 131 178	<1 s <1 s	<1 s 1 s	6 s 91 s	7.5 m 1.5 h	<1 s	<1 s	1 s	68 s
Lena	101	77 145	56 106 159	46 83 119 164	<1 s <1 s	<1 s 2 s	9 s 166 s	12 m 2.5 h	<1 s	<1 s	1 s	107 s
Peppers	102	81 142	43 98 152	40 88 134 173	<1 s <1 s	<1 s 1 s	7 s 105 s	8.5 m 1.7 h	<1 s	<1 s	1 s	77 s

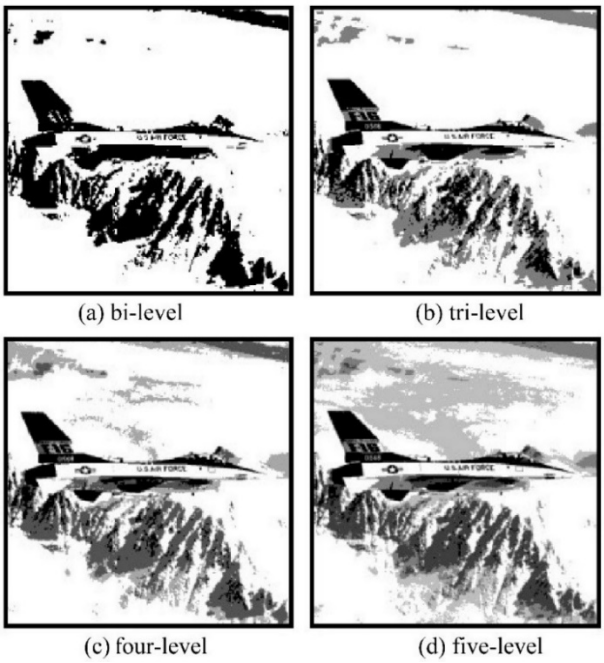

Figure 25. Segmentation results for F16 jet test image (Liao et al. 2001).

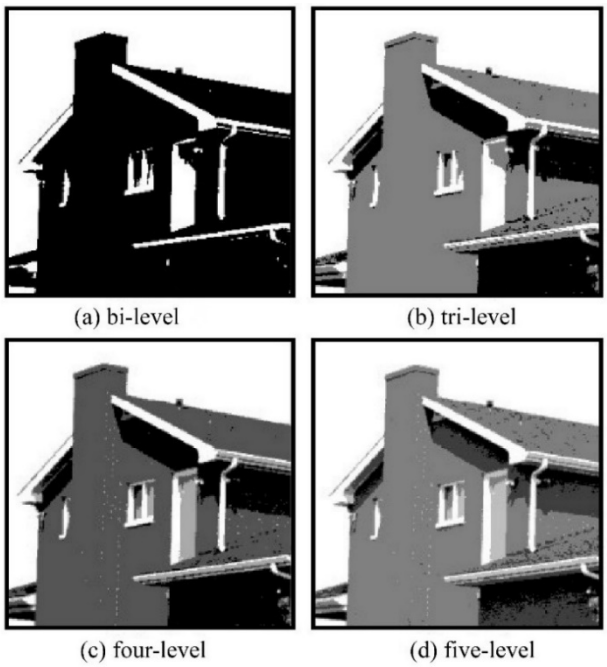

Figure 26. Segmentation results for house test image (Liao et al. 2001).

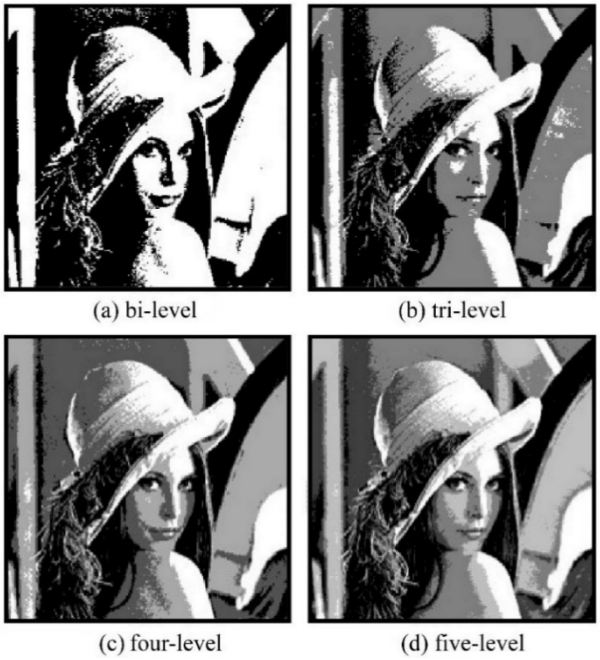

Figure 27. Segmentation results for Lena test image (Liao et al. 2001).

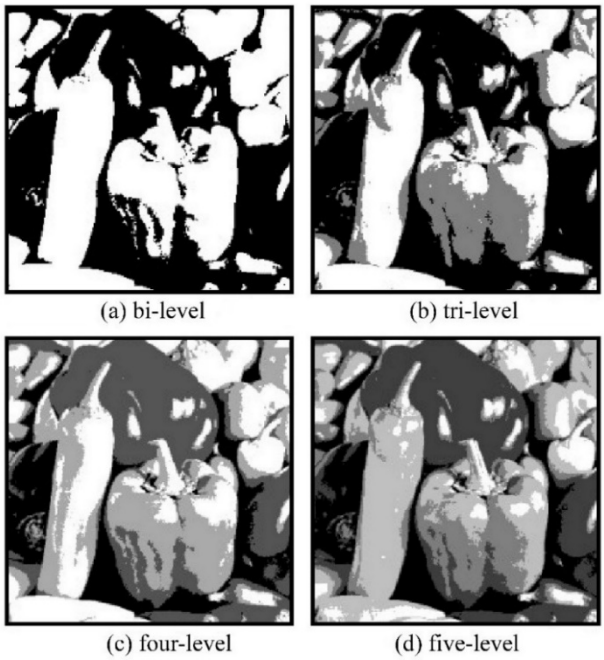

Figure 28. Segmentation results for peppers test image (Liao et al. 2001).

The computational times in Table 8 are essentially the same between bi-level and tri-level cases for the Otsu (1979) method regardless of whether recursion is used. On the other hand, recursion significantly decreases computational time for four-level and five-level cases. For example, recursion decreases four-level computation time for the F16 jet image by 65 seconds and by 54 minutes for the five-level case. The other three test images yield significant decrease in computation time for these two levels as well. Besides the algorithm of this study having a faster computational time than the Otsu (1979) original method, the Liao et al. (2001) algorithm yields the same threshold values as the Otsu (1979) method, therefore proving to be an overall superior technique to that of Otsu (1979).

The last paper by Arora et al. (2008) offers a fast, recursive algorithm for multi-level threshold image segmentation. The approach used an image's gray-level pixels to determine the mean and variance for segmentation of the image into multiple levels. Two main points were considered when the authors created this algorithm. The first is that a Gaussian distribution would represent an estimation of an image's histogram, because a majority of image histograms have pixels that tend to have high frequency values close to a specific value such as the mean. The other main point is that the human eye is not very sensitive to image features at extreme pixel values, but it is sensitive to features at mid-range intensity values. This suggests that it is once again useful to concentrate on the mean intensity value of an image.

The procedure followed for the Arora et al. (2008) method is quite simple. For example, assume that the image being segmented requires two threshold values to separate two objects and background classes from one another. The mean and standard deviation of

all of the pixels in the image (pixels ranging from a to b, where a is the smallest gray-level pixel value and b is the largest gray-level pixel value, initially) are determined and sub-range boundaries, T_1 and T_2, are found by utilizing this information. The pixels are then grouped into two intervals: pixels ranging from a to T_1 and pixels ranging from T_2 to b. Lastly, the weighted means of each interval are calculated, which provides the two threshold values for segmentation. Note that these steps can be repeated if more threshold values are desired. In such a case, the values of a and b are reassigned as T_1+1 and T_2-1, respectively. Two test images that are popular for image segmentation, a woman named Lena and peppers, were used to test the Arora et al. (2008) algorithm. Figure 29 represents the Lena image, where 29a is the original gray-level image, 29b is the gray-level histogram, and 29c-29f are the segmented image with 2 thresholds, 4 thresholds, 6 thresholds, and 8 thresholds applied, respectively. Figure 30, for the peppers image, follows the same representation as Figure 29. Table 9 also provides quantitative results for these two images for both the proposed (2008) algorithm and the conventional Otsu (1979) method for multi-level thresholding.

(Figure 29 has been removed due to copyright restrictions.
http://www.sciencedirect.com/science/article/pii/S0167865507002905)

(Figure 30 has been removed due to copyright restrictions.
http://www.sciencedirect.com/science/article/pii/S0167865507002905)

(Table 9 has been removed due to copyright restrictions.
http://www.sciencedirect.com/science/article/pii/S0167865507002905)

Figures 29 and 30 show that the Arora et al. (2008) algorithm successfully segments the original images; a segmented image looks visually more like the original image as the number of thresholds increase from 2 to 8. Table 9 provides evidence that the Otsu (1979) multi-level thresholding technique takes a much larger amount of time to calculate multiple threshold values in comparison to the Arora et al. (2008) recursive technique. Besides being computationally faster, the Arora et al. (2008) method provides the same threshold values as the Otsu (1979) method.

Three thresholding techniques were chosen for implementation to analyze multi-phase geomaterials. Otsu (1979) was chosen due to its history of being reliable. A technique called Iterative Otsu was created, which is a modification of the Otsu (1979) method to include a recursive process. Lastly, a technique named the Refined statistical-based method was developed as a modification of the Arora et al. (2008) method. In accordance with the formulations of the five bi-level techniques contained in Chapter 3, the formulations of the chosen three multi-level techniques are available in Chapter 3 as well.

Chapter 3

IMPLEMENTATION OF GLOBAL THRESHOLDING TECHNIQUES

3.1 Two-Phase Thresholding Techniques

Through previously conducted literature review, five global thresholding techniques, namely Otsu (1979), Pun (1980), Kapur et al. (1985), Johannsen and Bille (1982), and the Kittler and Illingworth (1986) methods, were chosen for application to multiple two-phase porous media. Table 10 lists some of the commonly applied thresholding techniques for two-dimensional "slice-by-slice" processing of porous media, which also includes the five implemented techniques of this study.

Table 10. Commonly applied thresholding techniques for porous media.

Technique	Material	Thresholding Technique
Baveye et al. (2010),	Soil	3D Thresholding
Carminati et al. (2007),	Soil	3D Thresholding
Culligan et al. (2006),	Glass Beads	3D Thresholding
Jassogne et al. (2007),	Soil	2D Thresholding
Johannsen and Bille (1982),	-	2D Thresholding
Kaestner et al. (2008),	Soil	3D Thresholding
Kapur et al. (1985),	-	2D Thresholding
Kittler and Illingworth (1986),	-	2D Thresholding
Kurita et al. (1992),	Glass Beads, Sandstone	2D Thresholding
Lee et al. (2008),	Soil	2D Thresholding
Nunan et al. (2006),	Aggregates	2D Thresholding
Ojeda-Magaña et al. (2014),	Soil	3D Thresholding
Otsu (1979),	-	2D Thresholding
Pun (1980),	-	2D Thresholding
Ridler et al. (1978),	Glass Beads	2D Thresholding
Schaap et al. (2007),	Glass Beads	3D Thresholding
Schlüter et al. (2010),	Soil	2D Thresholding
Van Geet et al. (2003),	Limestone, Sandstone	3D Thresholding
Vogel et al. (2005),	Sintered Glass	3D Thresholding
Wildenschild et al. (2002)	Sand	3D Thresholding

Each of the five chosen algorithms were programmed with MATLAB (Mathworks 2015). A subroutine that calculates and reports the void ratio from X-ray CT images was created and applied. Generally, the subroutine reads an image slice, crops it to a circle of appropriate size (so that the number of pixels are not over- or under-estimated), segments it by applying a chosen thresholding technique, and counts the number of void and solid pixels in the segmented image. Void ratio is determined by dividing the number of void pixels by the number of solid pixels. Since the scanned images represent a two-phase system, the total number of pixels are composed of air and solids. Once all of the image slices are processed, results are combined and reported as the void ratio of the specimen.

For computational efficiency, each analyzed image was converted to an 8-bit image, meaning that the grayscale pixel intensities range from 0-255, where pixels closer to the 0 end are colored black and pixels closer to the 255 end are white. For MATLAB© coding purposes, this range is taken to be from 1-256. Since the segmentation program is applied for two-phase images, void pixels are equal to air pixels. Therefore, any pixels less than a threshold value are black which corresponds to air and any pixels greater than a threshold value are white which corresponds to solids. In the images analyzed in this study, the process of reducing the bit depth from 16-bit to 8-bit does not result in a significant loss of information. This could be attributed to the fact that the specimens were made of "larger" sized granular media and important details were well preserved in the 8-bit images.

3.1.1 Otsu (1979) Method

Otsu (1979) calculates the optimal threshold through separability of the two classes, namely the foreground and background, by minimizing within-class variance or maximizing between-class variance. In other words, the optimal threshold is located where the summation of the foreground and background spreads are at a minimum (Otsu 1979). Equations 1-4 are used for this method, where the optimal threshold value (t) is taken as the pixel value yielding the maximum value from Equation 4.

$$mean = \frac{\sum_{i=1}^{c} i * p_i}{c} \tag{1}$$

$$meanli = \frac{\sum_{i=1}^{t-1} i * p_i}{c} \tag{2}$$

$$meangi = \frac{\sum_{i=t}^{c} i * p_i}{c} \tag{3}$$

$$t = \arg\max \{\sum_{i=1}^{t-1} p_i * \{meanli - mean\}^2 + \sum_{i=t}^{c} p_i * \{meangi - mean\}^2\} \tag{4}$$

In the above equations, *c* is the highest possible pixel value in an image (here, since 8-bit images are processed in MATLAB©, the highest possible value is 256); *i* represents a pixel value within the range of one up to c; *p* represents the individual pixel frequencies; *mean*, *meanli*, and *meangi* represent the mean of an image's pixel intensities ranging from one to *c*, one to *t*-1, and *t* to *c*, respectively.

3.1.2 Pun (1980) Method

In the Pun (1980) method, an assumption is made that the relationship between the number of pixels at a specific gray-level is statistically independent from the number of pixels at a nearby gray-level. Realistically, this assumption is not necessarily true, but is taken to be factual since this thresholding algorithm's derivation is greatly simplified while providing reasonable results (Pun 1980). Equations 5-8 are utilized for the Pun (1980) method, where the optimal threshold value (*t*) is determined when the criteria set by Equation 8 is achieved.

$$p_i = \frac{n(i)}{n} \tag{5}$$

$$\sum_{i=1}^{m} p_i \geq 0.5 \tag{6}$$

$$\alpha = \frac{\sum_{i=1}^{m} p_i * \log_e p_i}{\sum_{i=1}^{c} p_i * \log_e p_i} \tag{7}$$

$$\sum_{i=1}^{t} p_i = \begin{cases} \alpha, & \alpha > 0.5 \\ 1-\alpha, & \alpha \leq 0.5 \end{cases} \tag{8}$$

In Equations 5-8, p_i is the probability of occurrence of gray-level i; *n(i)* corresponds to the total number of pixels at a specific pixel value i; *n* represents the total number of pixels in an image; *m* is the smallest pixel value that satisfies Equation 6; α is the anisotropy coefficient representing the ratio of the average quantity of information of white and black pixels; *c* is the highest possible pixel value in an image.

3.1.3 Kapur et al. (1985) Method

The Kapur et al. (1985) method first derives two probability distributions (foreground and background) from the original gray-level distribution of an image and then determines the entropies associated with each distribution. The optimal threshold is taken as the gray-level that has the greatest summation of the two entropies (Kapur et al. 1985). Equations 9-15 represent this algorithm, where the optimal threshold value (*t*) refers to the pixel value that results in the maximum value of Equation 15.

$$p_i = \frac{n(i)}{n} \tag{9}$$

$$H_i = -\sum_{i=1}^{c} p_i * \log_e p_i \tag{10}$$

$$P_t = \sum_{i=1}^{t} p_i \tag{11}$$

$$H_t = -\sum_{i=1}^{t} p_i * \log_e p_i \tag{12}$$

$$H_a = \log_e P_t + \frac{H_t}{P_t} \tag{13}$$

$$H_b = \log_e (1 - P_t) + \frac{(H_i - H_t)}{(1 - P_t)} \tag{14}$$

$$t = \arg\max \{H_a + H_b\} \tag{15}$$

In the above equations, p_i is the probability of occurrence of gray-level i; c is the highest possible pixel value in an image; H_a and H_b represent the two probability distributions derived from the original gray-level distribution of the image.

3.1.4 Johannsen and Bille (1982) Method

The fourth algorithm, Johannsen and Bille (1982), is represented by Equations 16-18. Through utilizing the entropy of the gray-level histogram of an image, this method determines the optimal threshold value by dividing the set of gray-levels into foreground and background classes, in order to minimize the interdependence between them (Sahoo et al. 1988). The minimum value produced from Equation 18 corresponds to the optimal

threshold value (n^*).

$$S(n) = \log_e(\sum_{i=1}^{n} p_i) - \frac{1}{\sum_{i=1}^{n} p_i} * [p_n * \log_e p_n + (\sum_{i=1}^{n-1} p_i) * \log_e(\sum_{i=1}^{n-1} p_i)] \quad (16)$$

$$\bar{S}(n) = \log_e(\sum_{i=n}^{c} p_i) - \frac{1}{\sum_{i=n}^{c} p_i} * [p_n * \log_e p_n + (\sum_{i=n+1}^{c} p_i) * \log_e(\sum_{i=n+1}^{c} p_i)] \quad (17)$$

$$n^* = \arg\min(S(n) + \bar{S}(n)) \quad (18)$$

For Equations 16-18, p_i corresponds to the probability of occurrence of gray-level i; p_n represents the probability of occurrence of the pixel value following the one being analyzed (e.g., if i=25, p_n represents the probability for pixel value 26); $n = t+1$; c is the highest possible pixel value in an image; $S(n)$ and $\overline{S}(n)$ represent the two parts that the set of gray-levels are divided into.

3.1.5 Kittler and Illingworth (1986) Method

The fifth and final method that will be used for analysis is the Kittler and Illingworth (1986) method. The Kittler and Illingworth (1986) method determines the optimal threshold value by viewing the gray-level histogram as an estimation of the probability density function representing the gray-levels of the foreground and background pixels in an image. This method assumes that these two categories of pixels are normally distributed with priori probability, mean, and standard deviation (Sahoo et al. 1988). The algorithm for the Kittler and Illingworth (1986) method is presented by Equations 19-26, where the optimal threshold value (t) is the minimum value of the criterion function of Equation 26.

$$P_1 = \sum_{j=1}^{t} P_j \tag{19}$$

$$P_2 = \sum_{j=t+1}^{c-1} P_j \tag{20}$$

$$\mu_1 = \frac{(\sum_{j=1}^{t} P_j * j)}{P_1} \tag{21}$$

$$\mu_2 = \frac{(\sum_{j=t+1}^{c-1} P_j * j)}{P_2} \tag{22}$$

$$\sigma_1^{\,2} = \frac{\sum_{j=1}^{t} (j-\mu_1)^2 \times P_j}{P_1} \tag{23}$$

$$\sigma_2^{\,2} = \frac{\sum_{j=t+1}^{c-1} (j-\mu_2)^2 \times P_j}{P_2} \tag{24}$$

$$J = 1 + 2*(P_1 * \log_e \sigma_1 + P_2 * \log_e \sigma_2) - 2*(P_1 * \log_e P_1 + P_2 * \log_e P_2) \tag{25}$$

$$t^* = \arg\min(J) \tag{26}$$

For Equations 19-26, P_1 and P_2, μ_1 and μ_2, σ_1^2 and σ_2^2, and σ_1 and σ_2 represent the priori probability, mean, variance, and standard deviation of the foreground and background pixels, respectively; c is the highest possible pixel value in an image.

3.2 Three-Phase Thresholding Techniques

As seen in the above Table 10, various examples of thresholding techniques that have been applied for porous media analysis since 1978 are provided. These techniques are applicable in both two-dimensional and three-dimensional image processing. As listed in the table, images of granular soils and glass bead specimens were used for validating the thresholding algorithms chosen by the authors. For this work, multi-phase segmentation refers to three-phase segmentation. In an attempt to further the field of image segmentation of partially saturated granular media, this study proposes a new, three-phase image segmentation technique in which a refined statistical-based thresholding algorithm is employed.

As previously indicated in Section 3.1, the algorithms representing the modification and extensions of the Otsu (1979) method, as well as the one proposed here, were coded with MATLAB© (Mathworks 2015). Three of the techniques, the Otsu (1979) three-phase, Iterative Otsu, and the Refined statistical-based methods, were modified for proper application in MATLAB©. Once again, for coding purposes, the range of grayscale pixel intensities for the analyzed 8-bit images is taken to be from 1-256, rather than 0-255.

Three-phase images (solids, water, and air) of unsaturated granular geomaterials were used for characterization. Each technique searches for two optimum threshold values, namely threshold one (t1) and threshold two (t2). Pixels greater than t1 refer to solid particles and will be rendered with white color during segmentation. Pixels less than t2 refer to air pixels and will be rendered black during segmentation. Any pixel between t1 and t2 refers to water pixels and will be colored gray. Once a technique chooses the pixel

values yielding the optimal thresholds, three-phase segmentation is performed and the geomaterial's void ratio and degree of saturation are calculated from the segmented images. Void ratio is determined by dividing the number of void pixels by the number of solid pixels. Degree of saturation is determined by dividing the number of water pixels by the number of void pixels, expressed as a percentage. Since the scanned images represent a three-phase system, void pixels consist of water and air pixels.

3.2.1 Otsu (1979) Method

Nobuyuki Otsu proposed one of the oldest and most widely used global thresholding techniques, where an optimal threshold value is selected as the pixel value located at a sharp valley between peaks representing the objects and background of an image's gray-level histogram (Otsu 1979). The basis of this technique separates the two classes by minimizing within-class variance or maximizing between-class variance for optimal threshold selection. In other words, the optimal threshold value is the pixel value that results in the minimized summation of the foreground and background spreads (Sahoo et al. 1988).

Equations 27-30 are used for the Otsu (1979) three-phase method, where the optimal threshold values (i.e., t1 and t2) are taken as the pixel values yielding the maximum value of Equation 30. Note that these four equations are first applied to the entire range of pixels represented by the gray-level histogram to obtain threshold one. Threshold two is then determined by applying the same four equations, but this time to a refined histogram ranging from the smallest pixel value to the value of threshold one.

$$mean = \frac{\sum_{i=1}^{m} i * f_i}{m} \tag{27}$$

$$meanli = \frac{\sum_{i=1}^{t-1} i * f_i}{m} \tag{28}$$

$$meangi = \frac{\sum_{i=t}^{m} i * f_i}{m} \tag{29}$$

$$t = \arg\max \{\sum_{i=1}^{t-1} f_i * \{meanli - mean\}^2 + \sum_{i=t}^{m} f_i * \{meangi - mean\}^2\} \tag{30}$$

In the above equations, *m* is the highest possible pixel value in an image (here, since 8-bit images are processed in MATLAB©, the highest possible value is 256); *i* represents a pixel value within the range of one up to *m*; *f* represents the individual pixel frequencies; *mean*, *meanli*, and *meangi* represent the mean of an image's pixel intensities ranging from one to *m*, one to *t*-1, and *t* to *m*, respectively; *t* refers to the optimal threshold value of either t1 or t2.

3.2.2 Iterative Otsu Method

As with the previous method, the Iterative Otsu method calculates threshold one (t1) with Equations 27-30. The initial value for threshold two (t2) is calculated with Equations 31-33. The reason for expanding the Otsu (1979) three-phase method into the Iterative Otsu Method is because the procedure followed for finding the threshold two value through the Otsu (1979) three-phase method does not necessarily result in the optimal

value. The usage of Equations 34-36 determines a threshold value referred to as threshold new (tnew). Equations 34-36 are then used again in a loop (threshold new is assigned as threshold two) and the loop terminates once the value of threshold new is within two gray-level pixel values of the previous iteration. Once this criterion is met, the value associated with threshold new will be assigned as the optimal value of threshold two.

$$mean1 = \frac{\sum_{i=1}^{t1} i * f_i}{\sum_{i=1}^{t1} f_i} \tag{31}$$

$$mean2 = \frac{\sum_{i=t1+1}^{m} i * f_i}{\sum_{i=t1+1}^{m} f_i} \tag{32}$$

$$t2 = \frac{(mean1 + mean2)}{2} \tag{33}$$

$$mean3 = \frac{\sum_{i=1}^{t2} i * f_i}{\sum_{i=1}^{t2} f_i} \tag{34}$$

$$mean4 = \frac{\sum_{i=t2+1}^{m} i * f_i}{\sum_{i=t2+1}^{m} f_i} \tag{35}$$

$$tnew / t2 = \frac{(mean3 + mean4)}{2} \tag{36}$$

3.2.3 Refined Statistical-Based Method

Equations 37-38 represent the Refined statistical-based method where the optimal values for threshold one and two are determined for extracting three phases of a partially saturated granular media by assuming that the thresholding values are located within

unknown numbers of standard deviation from the left and right of the mean pixel intensity values. This proposed method follows the concept that numerous distributions have a tendency to follow the Dirac delta function with the peak located near to the mean pixel intensity values (Arora et al. 2008). The mean (μ) and standard deviation (σ) of the frequencies of all of the pixels in the image are determined. As mentioned in the work of Arora et al. (2008), many images contain normally distributed histograms. An estimation of such a histogram is a Gaussian distribution. Such histograms have high frequency values concentrated around a certain value (i.e., the mean pixel intensity value). Also, on a visual standpoint, it is much easier to distinguish objects from the background at intensity values near to the mean. However, not all images have histograms with normal distributions. Fitting parameters, k1 and k2, are then utilized to provide effectively segmented images with asymmetric or skewed histograms. Through alteration of the fitting parameters, the concepts applied for normally distributed histograms adapt to non-uniform distributions.

The work of Arora et al. (2008) provides similar equations to those in Equations 37-38. However, the methodology of that work consists of finding two or more thresholding values to represent an image with multiple shades of intensity. Here, the three different phases of partially saturated granular geomaterials are to be accurately captured, which implies the usage of two thresholding values and imposes a trial and error approach. The authors chose a value of one for both k1 and k2, for simplicity. The images used in that study were random (e.g., a women, peppers, jet, and a house). The proposed method does not provide successful segmented images for the geomaterials of this study when the values of k1 and k2 are equal to one. By adjusting these parameters, it is seen that the

method is more sensitive to the value of k1 than the value of k2. A trial and error process was implemented to search for k1 and k2 values to obtain a segmented image that effectively captures all of the portions of the raw image. For the geomaterials used in this study, it is recommended that k1 ranges from zero to two and k2 ranges from zero to three, while ensuring that k2 is always greater than k1. These conditions for the fitting parameters vary depending on the type of images being analyzed. Overall, the proposed method provides faster processing time than both the Arora et al. (2008) and Otsu (1979) three-phase methods and superior thresholding values than the Otsu (1979) three-phase method.

$$t1 = abs\left(\mu + \frac{\sigma * (k2 - k1)}{2} \right) \tag{37}$$

$$t2 = abs\left(\frac{\sigma * (k1 + k2)}{2} \right) \tag{38}$$

Chapter 4

APPLICATION OF GLOBAL THRESHOLDING TECHNIQUES

4.1 The Purpose of Creating a Graphical User Interface (GUI) Standalone Executable Software

A graphical user interface (GUI) is a type of user interface that allows for user-friendly interaction with electronic devices through icons such as pushbuttons and dropdown menus. There are five chosen thresholding techniques for two-level image segmentation and three techniques for three-level segmentation. Since the algorithms were written in MATLAB©, this software would need to be available to the user to allow for proper execution of these subroutines. Also, multiple script windows would have to be open to run all of the techniques which would be an inefficient approach. Therefore, creating a GUI bypasses software requirements and neatly displays all of the techniques and results on one screen. A GUI is treated as a standalone executable software, meaning that the interface can be installed on computers just like any other software (e.g., Microsoft Word, Google Chrome, and Adobe Acrobat).

4.2 Features and Capabilities

A GUI was created for two-phase and three-phase image segmentation, as shown by Figures 31-34. Figure 31 depicts the two-phase image segmentation GUI that was originally created for geomaterial analysis. As seen in the figure, the GUI mostly consists

of pushbuttons which makes overall execution time-consuming. For example, to determine the void ratio for Otsu's (1979) method, the button labeled "Otsu 2 Phase" and its corresponding "Void Ratio" button would both have to be pressed. Ten buttons would have to be pushed to determine the void ratio for all five techniques with many more for obtaining other parameters. In order to increase efficiency and make the GUI more user-friendly, a final version of the program was created (Figure 32). By comparing Figure 32 to Figure 31, it is clear that the GUI in Figure 32 is cleaned up with significantly less buttons that are replaced with dropdown menus. The same reasoning follows for the three-phase segmentation programs displayed in Figures 33 and 34.

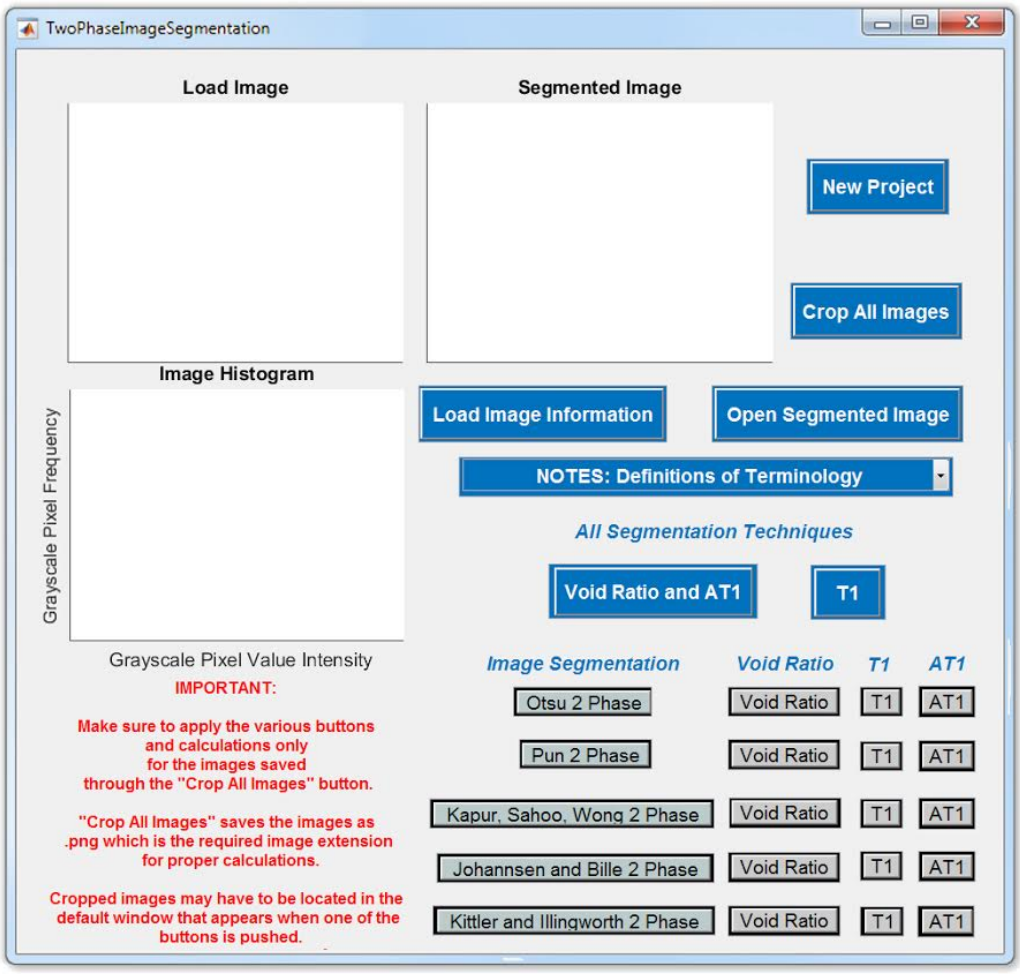

Figure 31. Original two-phase image segmentation GUI.

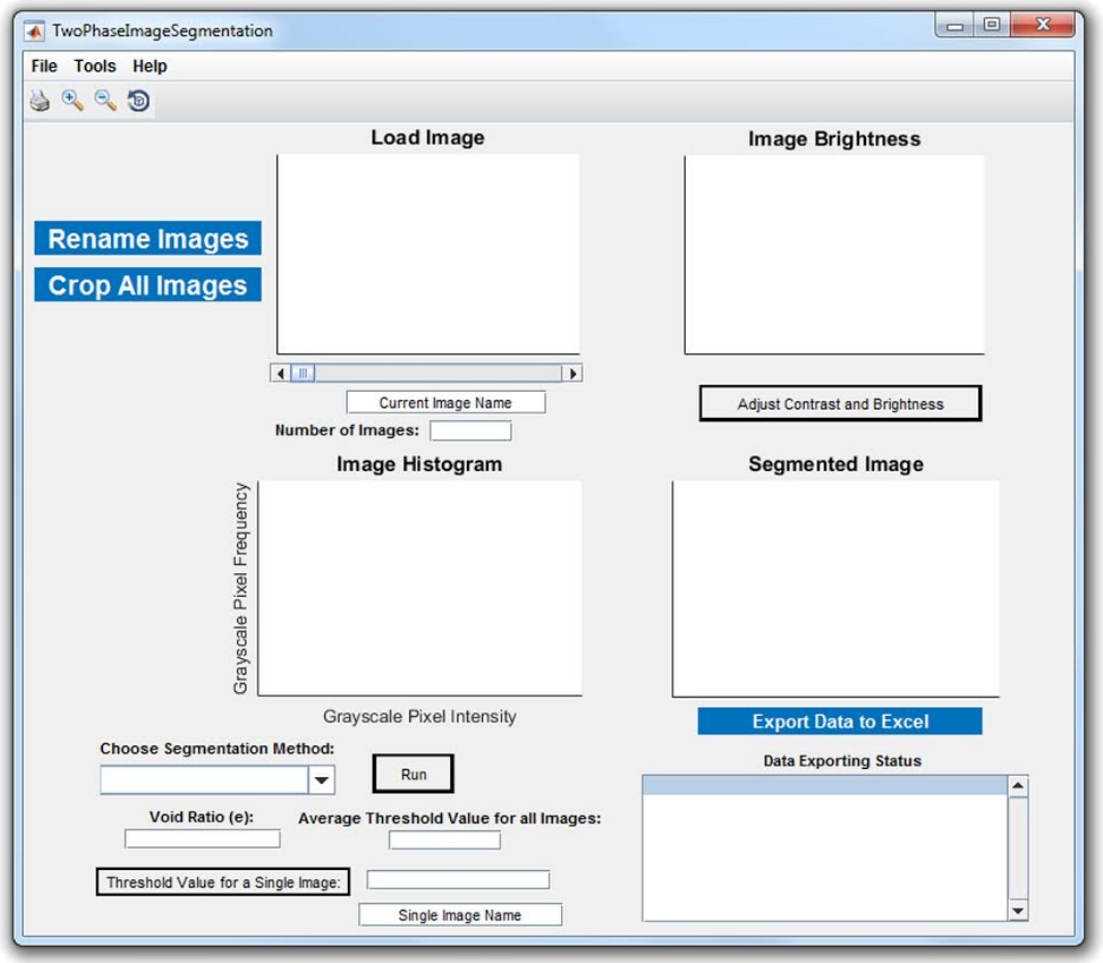

Figure 32. Final two-phase image segmentation GUI.

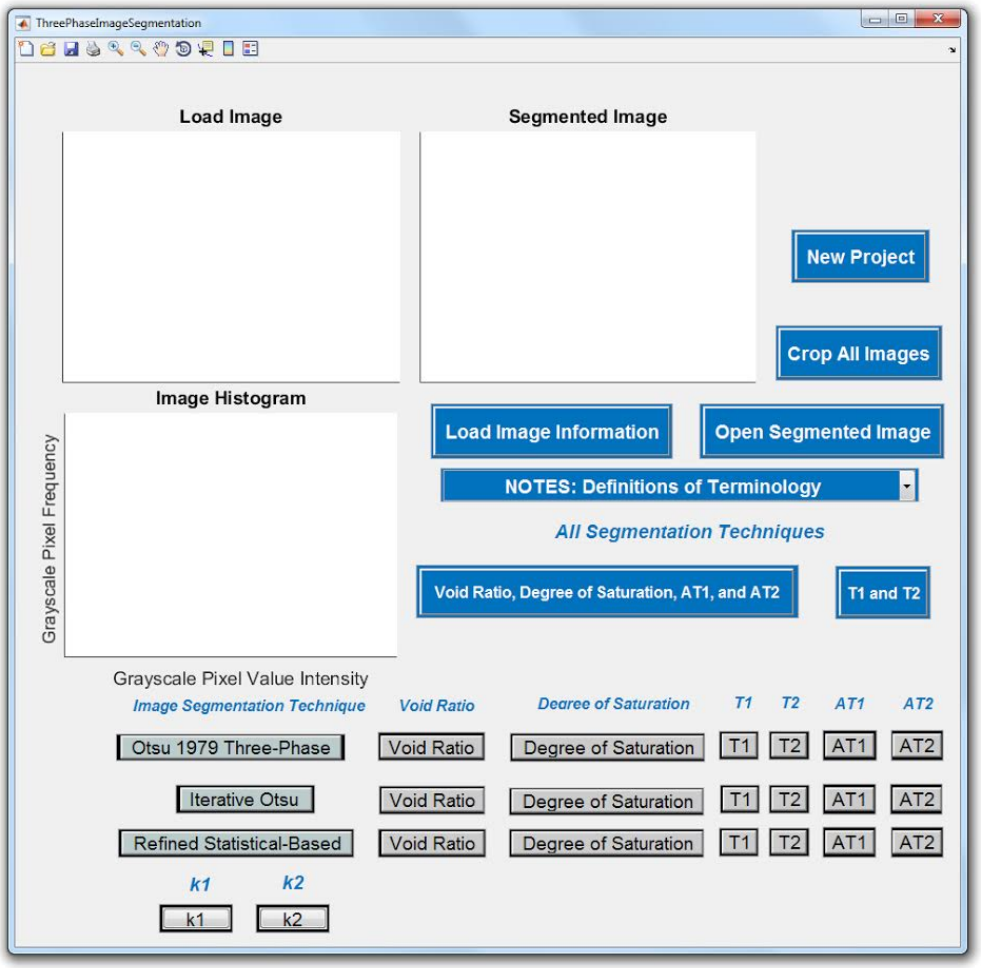

Figure 33. Original three-phase image segmentation GUI.

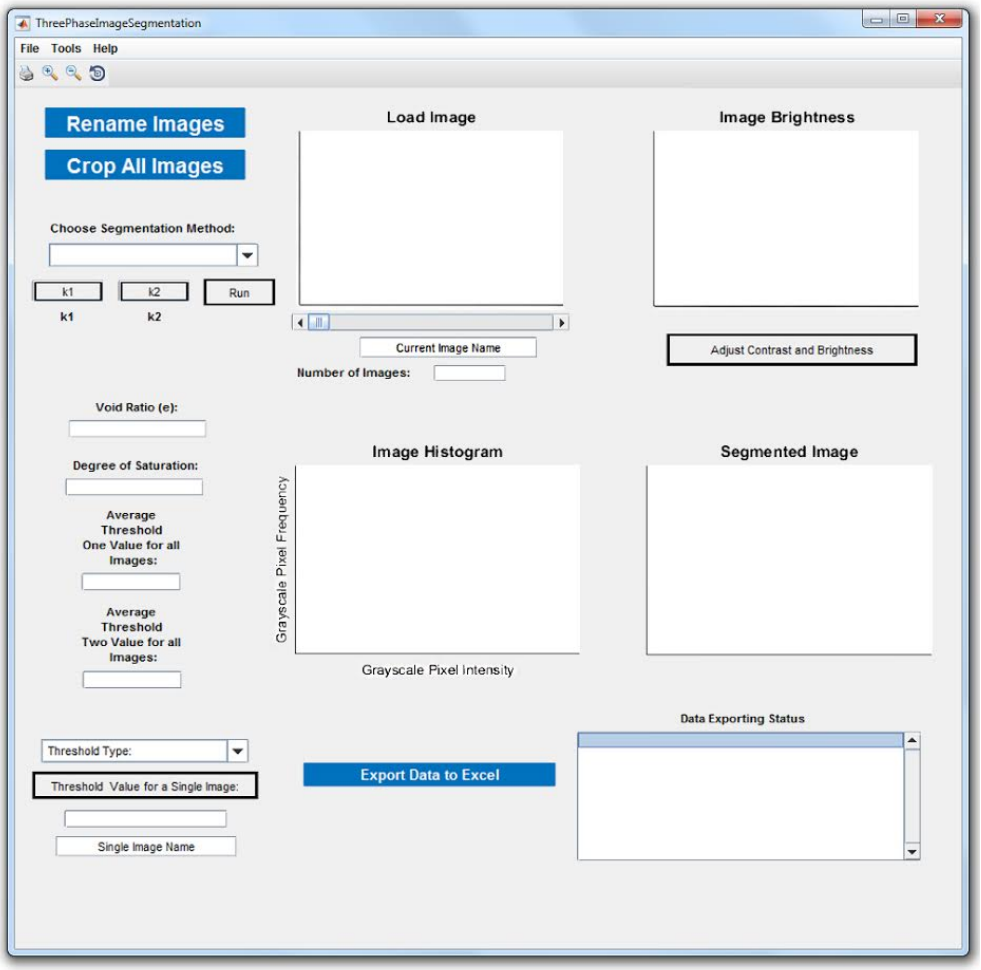

Figure 34. Final three-phase image segmentation GUI.

Even though two separate GUIs were created, they both have the same main features and capabilities. In Figures 32 and 34, there are three toolbar buttons: File, Tools, and Help. File subdivides into “Load Images,” “New Project,” “Close Program,” and “Save Segmented Images.” Tools branches into “Adjust Contrast and Brightness” and “Export Data to Excel.” Lastly, Help contains “User Guide” and “About Us” information. These tools were added to further increase the ease of using the programs.

For proper software execution, the first button to push is “Rename Images” which renames all of the images of interest to a format understood by the software. The next button, “Crop All Images,” first selects a folder to save the cropped images to and then displays the first image slice of the set in the “Load Image” window. A real-time ellipse is drawn around the specimen to ensure that the specimen is properly cropped with the capabilities of using a circular-shaped crop. The images that are to be segmented must be cropped to ensure that calculations are performed only for the specimen. In other words, images are composed of pixels whose information is stored in a matrix. Such matrices can be either square-shaped or rectangular-shaped. If the specimen in an image is circular-shaped, portions of the four corners of the image would not include any of the specimen. For clarification, see Figure 35 which shows the first image slice of the two-phase pervious concrete specimen.

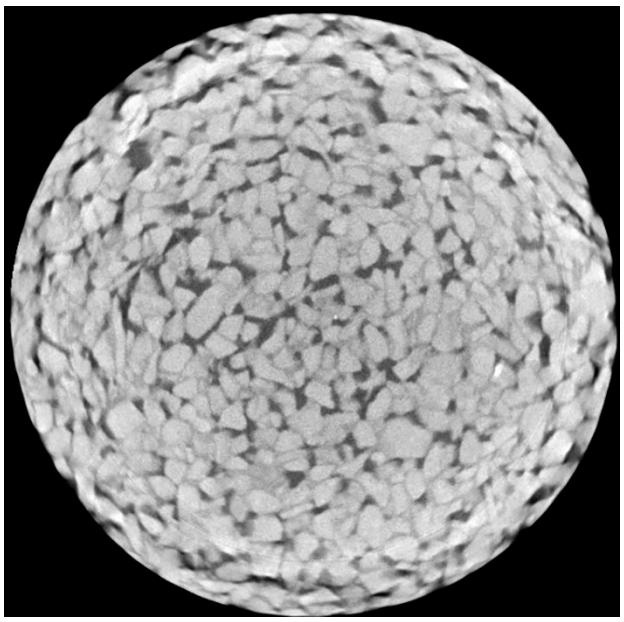

Figure 35. First original image slice of pervious concrete specimen.

As shown in the figure, the four black corners do not contain any of the specimen. If this image is not cropped, the thresholding techniques would mistake these black portions as air, thus greatly overestimating the specimen's void ratio. Depending on how the specimens were scanned and how the image slices were presented, image slice renderings could include the container that the specimen was scanned in and a scale for the sizes of the particles. Regardless, these inclusions would result in inaccurate results, further solidifying the need for image cropping. Automatic cropping has proven to be undependable, due to being greatly affected by the conditions in which a specimen is scanned and how the scanned images are rendered. Automatic cropping procedures attempt to locate the boundaries of a specimen based on pixel information and will incorrectly assume that the cropped boundaries include portions of the container and the scale. This occurrence also holds true if the specimen is not centered during imaging. Therefore, the program allows the user to manually draw the cropping window, on top of the opened X-ray image, and save it. The same cropping is automatically applied to the rest of the image slices and these slices are resaved as well. The cropped images are saved with a .png extension so that the images' backgrounds are transparent to prevent any inaccurate pixel counts. Once all of the scanned images are successfully cropped and saved, "Load Images" displays the first cropped image and its corresponding image histogram of grayscale pixel frequency versus grayscale pixel intensity. A horizontal scrollbar can be clicked to view the remaining image slices and their associated histograms. For viewing purposes only, the contrast and brightness of each image can be adjusted in the "Image Brightness" window.

The next task is to select the segmentation method of interest from the dropdown

menu and click “Run” to select the folder containing all of the cropped image slices. The GUI will run through the algorithm and displays the segmented image in the “Segmented Image” window, the void ratio value, and the average threshold one value for all of the images. The horizontal scrollbar can be used again to view the remaining segmented slices. The option to view the threshold one value for a single image slice is available. Lastly, all of the segmented images for each thresholding technique can be saved to the folder containing the cropped images and all of the results can be easily exported to Microsoft Excel. Between the two-phase and three-phase GUIs, the three-phase GUI also calculates the degree of saturation and the average threshold two value for all of the images.

Chapter 5

RESULTS AND DISCUSSION

5.1 Two-Phase Geomaterial Specimens

The five thresholding techniques described in Chapter 3 Section 3.1 were applied to X-ray CT images obtained for three different specimens. These specimens were composed of pervious concrete, glass bead, and silica sand, where the two phases are solids and air. In order to qualitatively analyze the results of the five techniques, and present that discussion here, the segmentation results of each method is represented by the first image slice for the image set of each specimen.

5.1.1 Pervious Concrete and Air

The pervious concrete specimen was originally 100 mm in diameter and was newly prepared in a laboratory setting. Three hundred and thirty-five image slices were obtained for this specimen through X-ray CT scanning. The laboratory measured gravimetric void ratio was 0.26. Figure 36 shows the first image slice with the applied manual crop as displayed in the standalone software developed as part of this study. The calibrated size of this cropped specimen was approximately 68 mm in diameter.

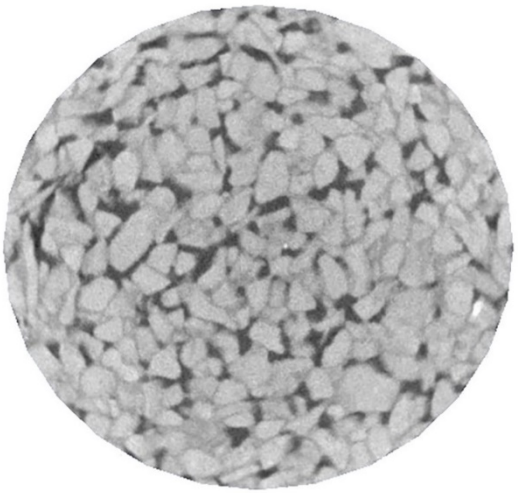

Figure 36. First cropped image slice of pervious concrete specimen.

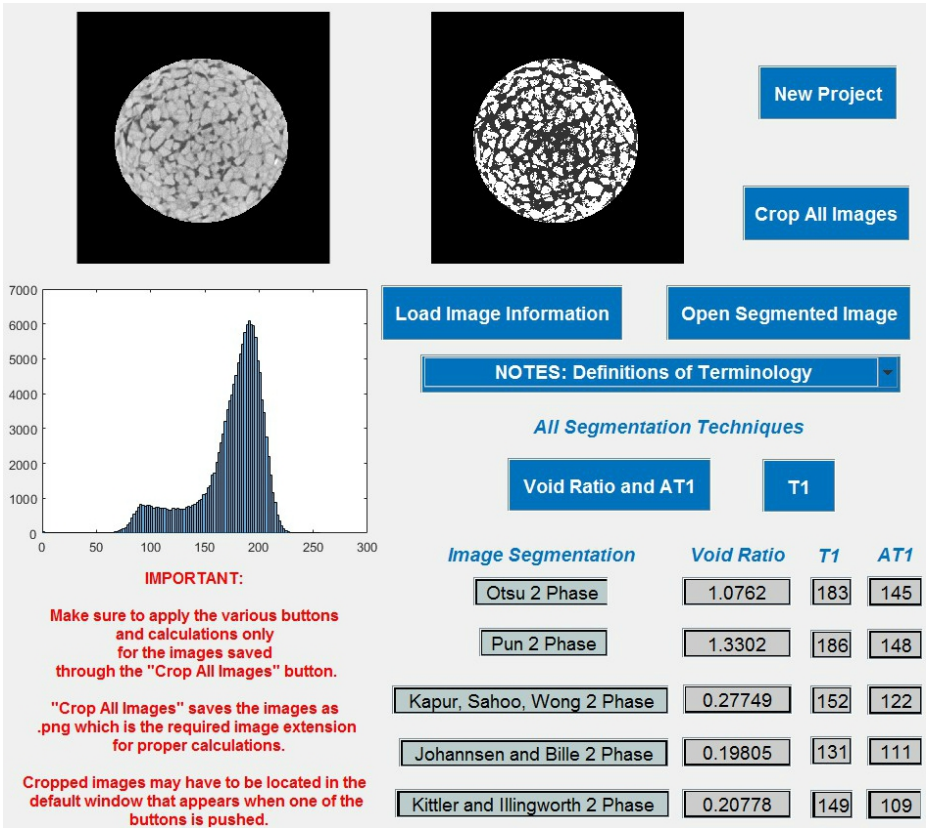

Figure 37. Pervious concrete results of thresholding techniques (displayed image slice segmented with the Otsu (1979) method).

Figure 37 above shows the quantitative results from the applied thresholding techniques. The void ratio, threshold value for the first image slice (T1), and the average threshold value (AT1) for the set of image slices per method is displayed in the figure. Also, for visual purposes, the figure depicts the cropped image of the original first image slice, its gray-level histogram, and the resulting segmented image with the Otsu (1979) method applied. To assist with qualitative analyses, Figure 38 presents the segmentation of the first image slice per thresholding method.

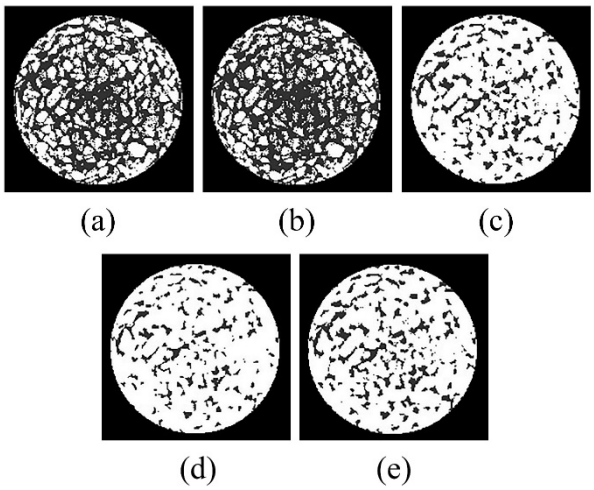

Figure 38. Segmented pervious concrete slice with different thresholding methods applied: (a) Otsu (1979) method, (b) Pun (1980) method, (c) Kapur et al. (1985) method, (d) Johannsen and Bille (1982) method, (e) Kittler and Illingworth (1986) method.

The void ratio for Otsu (1979), Pun (1980), Kapur et al. (1985), Johannsen and Bille (1982), and the Kittler and Illingworth (1986) methods were 1.076, 1.33, 0.28, 0.20, and 0.21, respectively. As previously mentioned, the laboratory measured void ratio was found to approximately be 0.26. For the first image slice, the Pun (1980) method had the

highest threshold value (T1=186), followed by Otsu (1979) (T1=183), Kapur et al. (1985) (T1=152), Kittler and Illingworth (1986) (T1=149), and the Johannsen and Bille (1982) methods (T1=131). On average, the threshold values for the entire image set for these five techniques were 148, 145, 122, 109, and 111, respectively.

From a quantitative standpoint, for the pervious concrete specimen, the results from the Otsu (1979) and Pun (1980) methods were the least accurate of the five thresholding techniques. Qualitatively speaking, the segmentation of this image slice by these two methods eroded away too much of the solid particles, more so with the Pun (1980) method. The average threshold values for these methods were very close to one another and helped validate this argument. Since their algorithms determined the thresholds to be this high, the methods yielded unrealistic void counts for the specimen. Therefore, it can be deduced that an acceptable segmentation technique must have an average threshold value much lower than what was seen with these two methods.

The void ratios from Kapur et al. (1985), Johannsen and Bille (1982), and the Kittler and Illingworth (1986) methods were very similar to one another. Relatively speaking, the best segmentation technique for this image set was the Kapur et al. (1985) method. Quantitatively, this technique's void ratio was the closest to the laboratory measured void ratio. For further validation, the image processing program, Image-Pro©, yielded a void ratio of 0.30 which was also relatively close to the result from the Kapur et al. (1985) method. Qualitatively, this method came the closest to accurately capturing the size and shape of the solid particles in the original image, whereas the other two methods had the solid particles being more filled and widened.

5.1.2 Glass Bead and Air

The original glass bead specimen was 10 mm in diameter and eleven image slices (first cropped image slice depicted as Figure 39) were obtained for this specimen through X-ray CT scanning. The cropped images contained the entire specimen, so the diameter remained as 10 mm. Figure 40 provides the quantitative results of the applied thresholding methods (first image slice used for T1) and the qualitative results of the first image slice with the Otsu (1979) method applied. Figure 41 presents the segmentation of the first image slice with each thresholding method applied.

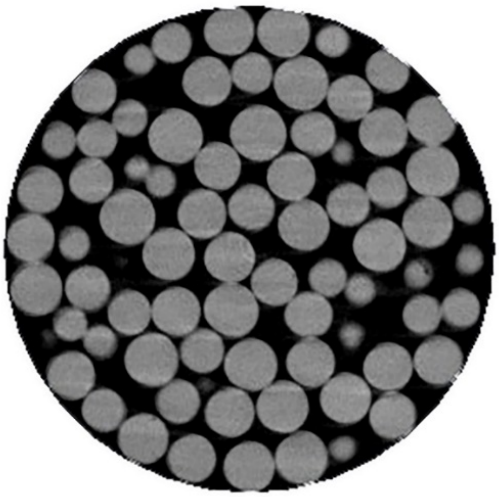

Figure 39. First cropped image slice of glass bead specimen.

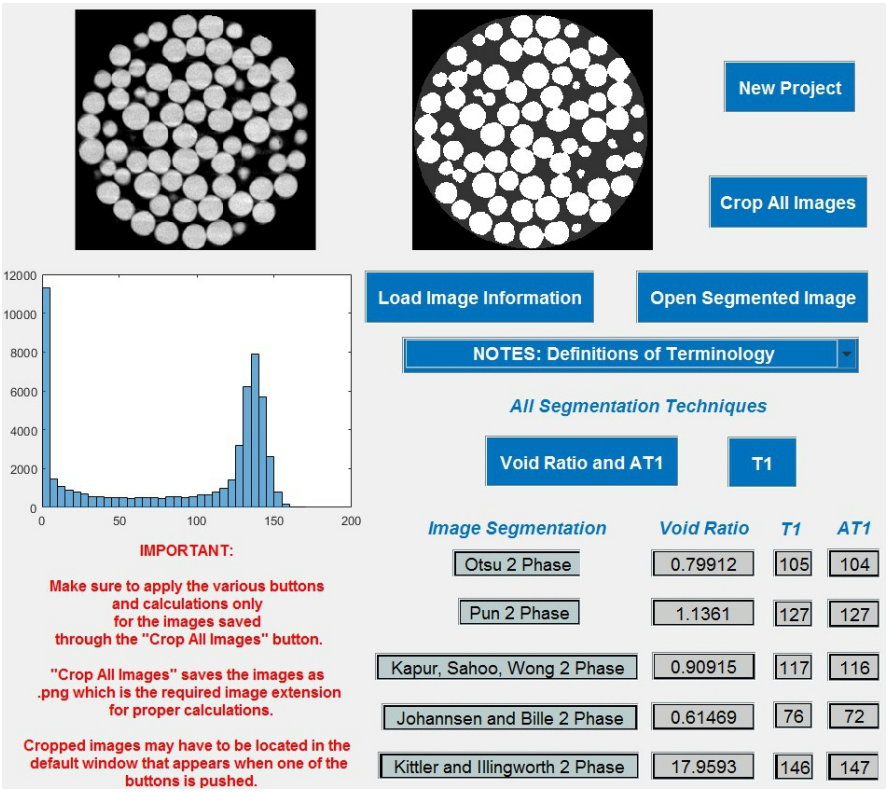

Figure 40. Glass bead results of thresholding techniques (displayed image slice segmented with the Otsu (1979) method).

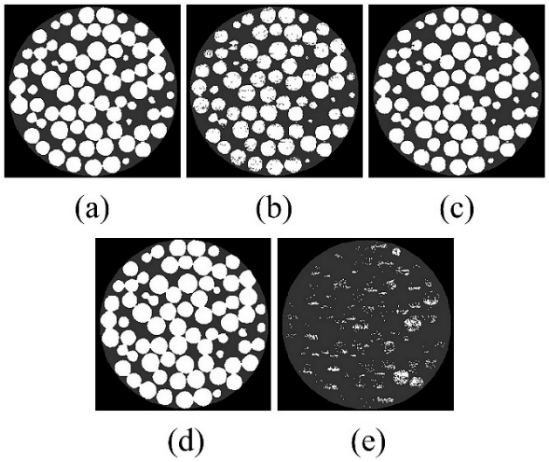

Figure 41. Segmented glass bead image slice with different thresholding methods applied: (a) Otsu (1979) method, (b) Pun (1980) method, (c) Kapur et al. (1985) method, (d) Johannsen and Bille (1982) method, (e) Kittler and Illingworth (1986) method.

Quantitatively, the void ratio of 17.96 obtained through the Kittler and Illingworth (1986) method was very inaccurate. From Figure 41e, it can be seen that this technique poorly segmented the image, treating the majority of solid pixels as void pixels. The threshold value of 146 for the first image slice and the average threshold value of 147 for the set of image slices were far too large. The Johannsen and Bille (1982) method yielded the smallest void ratio of 0.61, followed by Otsu (1979) (e=0.80), Kapur et al. (1985) (e=0.91), and the Pun (1980) methods (e=1.14). The corresponding threshold values for the first image slice were 76, 105, 117, and 127, respectively. The average threshold values for the set of image slices were 72, 104, 116, and 127, respectively. For a basis of comparison, Image-Pro© yielded a void ratio of 0.89. Thus, the Kapur et al. (1985) method provided the best segmentation, quantitatively.

By qualitatively analyzing the segmentation results, it can immediately be seen that Pun (1980) and the Kittler and Illingworth (1986) methods yielded unacceptable segmentations. The specimen segmented by the Pun (1980) method had small black dots scattered across the glass beads, indicating more voids and hence a higher threshold value than what is realistic. The Kittler and Illingworth (1986) method yielded the highest threshold value of all the techniques which explained why this method barely captured any of the glass beads. For the Johannsen and Bille (1982) method, the glass beads were slightly too filled and widened. This was visually identifiable since this segmentation resulted in the formation of contact points where gaps should have resided. Otsu (1979) and the Kapur et al. (1985) methods were the two best segmentation options which also had void ratios closest to the Image-Pro© void ratio. The Otsu (1979) method resulted in unwanted contact

points between the glass beads in various locations of the specimen, although not as profound as with the Johannsen and Bille (1982) method. So, overall, the Kapur et al. (1985) method yielded the best results quantitatively and qualitatively for the glass bead specimen.

5.1.3 Silica Sand and Air

The third and last specimen analyzed for two-level geomaterial characterization was silica sand with a specimen diameter of 6.35 mm. Ten image slices were utilized (first cropped image slice represented by Figure 42) for the analysis. The specimen was cropped to a diameter of 4.48 mm. Figure 43 shows the results from the standalone program with T1 calculated for the first image slice and the segmentation of this slice displayed with the Otsu (1979) method as well. Figure 44 presents the segmentation of the first image slice with each thresholding method applied.

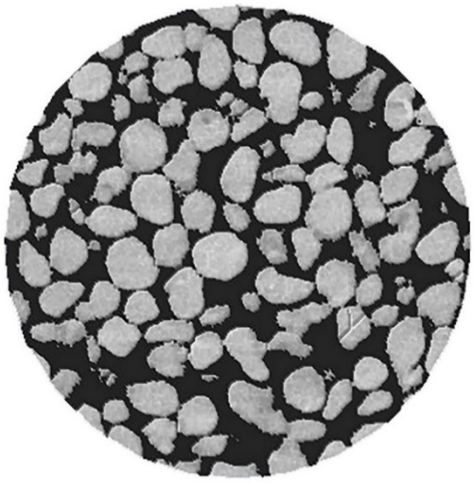

Figure 42. First cropped image slice of silica sand specimen.

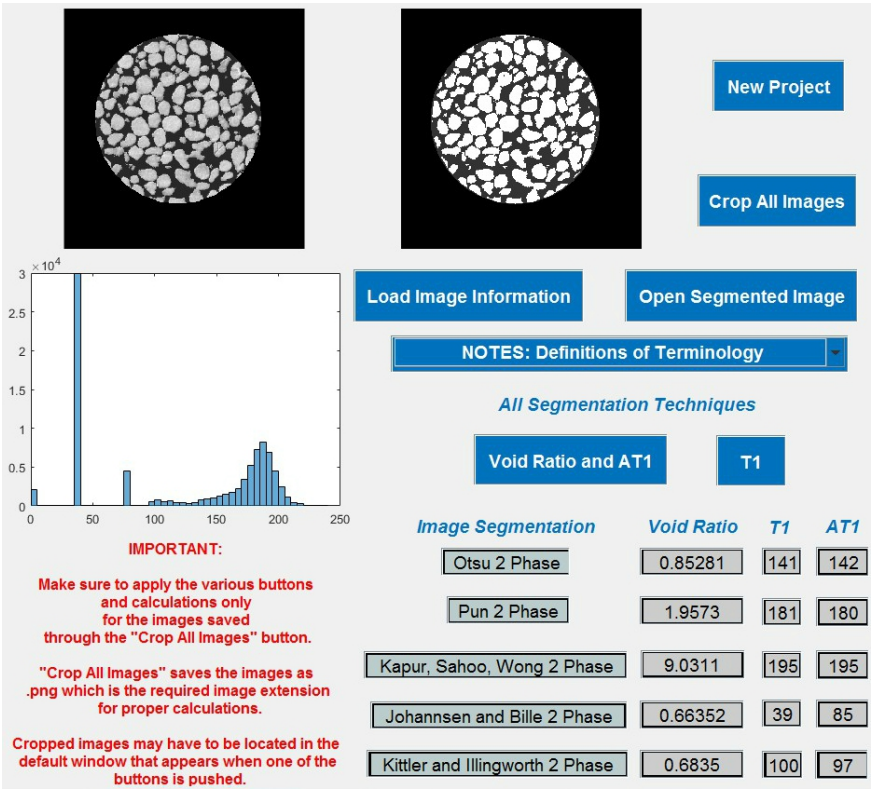

Figure 43. Silica sand results of thresholding techniques (displayed image slice segmented with the Otsu (1979) method).

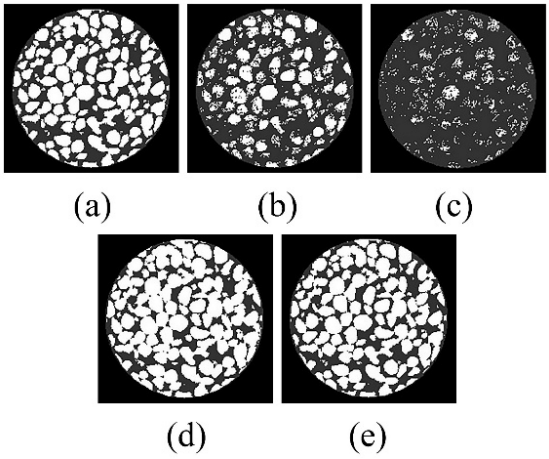

Figure 44. Segmented silica sand image slice with different thresholding methods applied: (a) Otsu (1979) method, (b) Pun (1980) method, (c) Kapur et al. (1985) method, (d) Johannsen and Bille (1982) method, (e) Kittler and Illingworth (1986) method.

From Figure 43, the void ratio of 9.031 obtained through the Kapur et al. (1985) method seems too high, meaning that the corresponding threshold values of 195, for the first image slice and the whole set of image slices, were not realistic. This value suggested that the original silica sand specimen was significantly composed of more air than sand particles, which was not the case. For the other image segmentation techniques, the method with the highest void ratio was the Pun (1980) method (e=1.96), followed by Otsu (1979) (e=0.85), Kittler and Illingworth (1986) (e=0.68), and the Johannsen and Bille (1982) (e=0.66) methods. The threshold values for the first image slice for these methods were 181, 141, 100, and 39, respectively. The average threshold values for the four methods were 180, 142, 97, and 85, respectively. Comparing to the void ratio of 0.77 that was obtained from Image-Pro©, the Otsu (1979) method was the best segmentation technique for the silica sand specimen.

Qualitatively, starting with the Pun (1980) method, the segmented image slice in Figure 44b shows that the solid particles are too eroded away, in comparison to the original image slice. In other words, the chosen threshold value was too high since the segmented image visually contained too much air. The Johannsen and Bille (1982) method had the lowest threshold value which caused the solid particles to be too filled (Figure 44d). Lastly, Otsu (1979) and the Kittler and Illingworth (1986) methods provided similar qualitative results. However, the Otsu (1979) method was more capable of capturing the visible cracks in the silica sand particles. As a result, the Otsu (1979) method was chosen as the best image segmentation technique for the silica sand specimen.

5.1.4 Summary of Two-Phase Image Segmentation Results

Table 11 provides a summary of the final void ratio results for the porous media analyzed in this study. Tables 12-14 provide statistical descriptors and comparisons of the five techniques for each of the porous media analyzed. Note that for Table 13 the Kittler and Illingworth (1986) method was excluded from the statistical comparison due to the produced void ratio being a wild outlier. The same reasoning applied for why the Kapur et al. (1985) method was not included in Table 14. The results of these tables collectively show that the methods as a whole work best for the glass bead specimen due to this specimen having the lowest coefficient of variation. This is followed by the silica sand specimen and the pervious concrete specimen. Individually, the application of the Kapur et al. (1985) method to the glass bead specimen yielded the least segmentation error (1.71%). Contrarily, the best segmentation technique with the greatest segmentation error (10.49%) was the Otsu (1979) method for the silica sand specimen.

Table 11. Final void ratio results for the porous media specimens.

Porous Media	Best Segmentation Technique	Chosen Technique's Void Ratio	Image-Pro Void Ratio	Percent Error
Pervious Concrete	Kapur et al. (1985)	0.28	0.30	6.89%
Glass Bead	Kapur et al. (1985)	0.91	0.89	1.71%
Silica Sand	Otsu (1979)	0.85	0.77	10.49%

Table 12. Statistical results and comparisons for the pervious concrete specimen.

Porous Media	Image Segmentation Technique	Void Ratio	Error	Standard Deviation	Coefficient of Variation (CV)
Pervious Concrete	Otsu (1979)	1.076	0.61	0.46	74.25%
	Pun (1980)	1.33	1.065		
	Kapur et al. (1985)	0.28	0.00042		
	Johannsen and Bille (1982)	0.20	0.0010		
	Kittler and Illingworth (1986)	0.21	0.0081		
	Image-Pro	0.30	----------		

Table 13. Statistical results and comparisons for the glass bead specimen.

Porous Media	Image Segmentation Technique	Void Ratio	Error	Standard Deviation	Coefficient of Variation (CV)
Glass Bead	Otsu (1979)	0.80	0.0090	0.19	21.82%
	Pun (1980)	1.14	0.059		
	Kapur et al. (1985)	0.91	0.00024		
	Johannsen and Bille (1982)	0.61	0.078		
	Image-Pro	0.89	----------		

Table 14. Statistical results and comparisons for the silica sand specimen.

Porous Media	Image Segmentation Technique	Void Ratio	Error	Standard Deviation	Coefficient of Variation (CV)
Silica Sand	Otsu (1979)	0.85	0.0066	0.54	51.49%
	Pun (1980)	1.96	1.41		
	Johannsen and Bille (1982)	0.66	0.012		
	Kittler and Illingworth (1986)	0.68	0.0078		
	Image-Pro	0.77	--------		

5.2 Three-Phase Geomaterial Specimens

The three thresholding techniques described in the preceding section were applied to X-ray CT images obtained for two different partially saturated specimens (composed of solids, water, and air), made out of silica sand and glass bead. For presentation purposes, and comparison of the performance of each thresholding technique, the first image slice of the image set for each specimen is used.

5.2.1 Silica Sand, Water, and Air

The silica sand specimen had an original and cropped diameter of 6.35 mm and was newly prepared in a laboratory setting. Ninety image slices were obtained for this specimen through X-ray CT scanning. In order to quantitatively verify the accuracy of the results of the three techniques, the void ratio and degree of saturation for this specimen were also determined with the image processing software, Image-Pro©. These values were found to be 0.66 and 37.71%, respectively. Figure 45 shows the first silica sand image slice with the applied manual crop as displayed in the standalone software developed as part of this study.

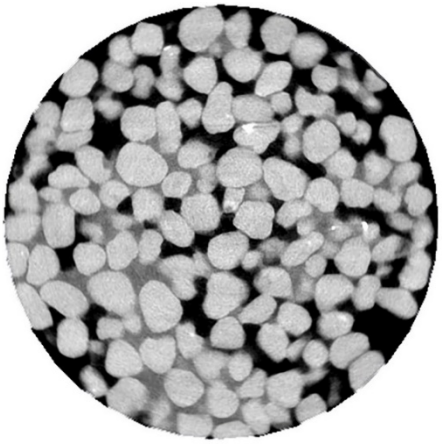

Figure 45. First cropped image slice of a partially saturated silica sand specimen.

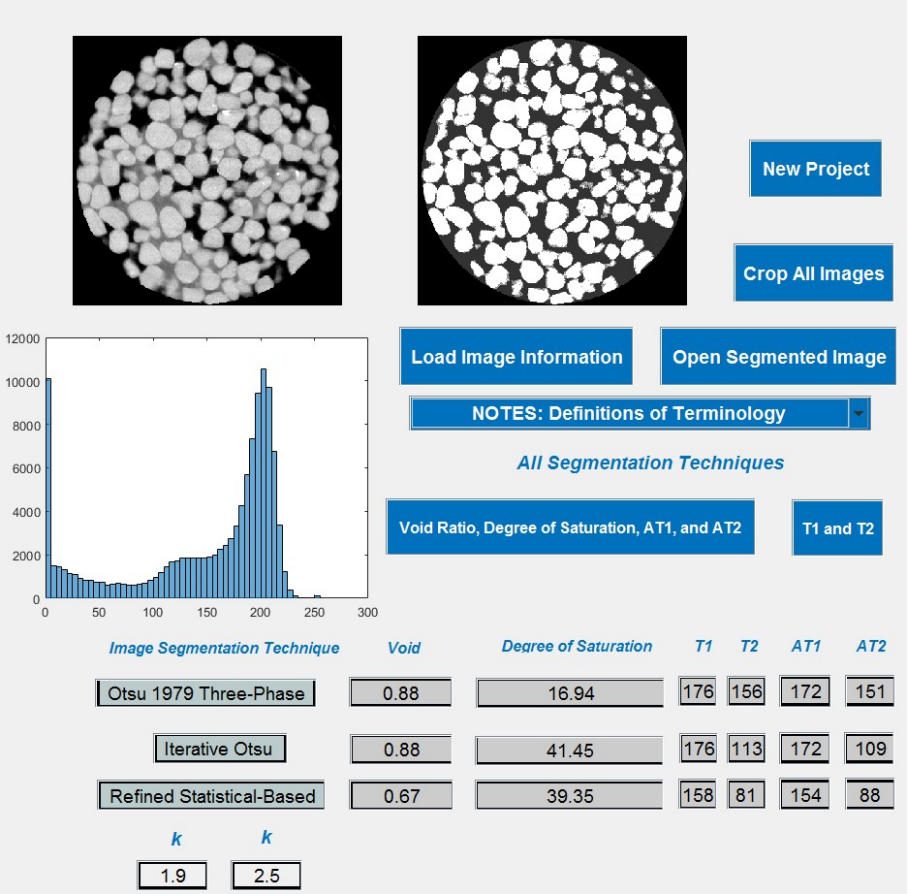

Figure 46. Partially saturated silica sand results of thresholding techniques (displayed image slice segmented with the Otsu (1979) three-phase method).

Figure 46 represents the quantitative results from the applied thresholding techniques. The void ratio and degree of saturation of the specimen, and average values of threshold one and threshold two per technique, are displayed in the figure. The values of threshold one and threshold two for the first image slice are also provided for each technique. The raw and segmented X-ray CT images of the first slice are displayed together with its corresponding gray-level histogram. In this specific case, the Otsu (1979) three-phase segmentation technique was applied. Figure 47 presents the segmented images, for the same image slice, when different thresholding methods are applied.

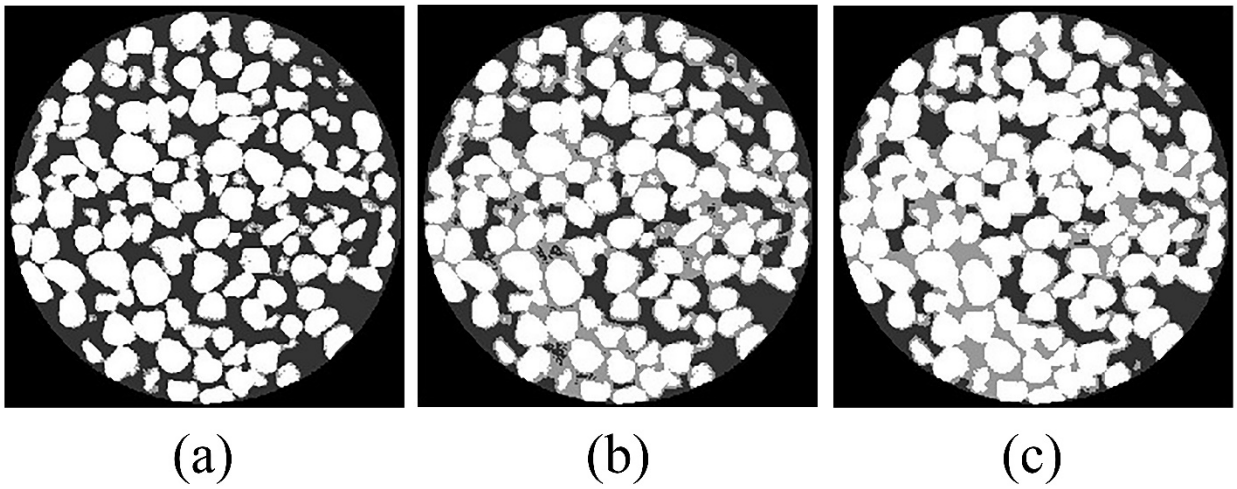

Figure 47. Segmented images of a partially saturated silica sand slice: (a) Otsu (1979) three-phase method, (b) Iterative Otsu, (c) Refined statistical-based method.

Figure 46 shows that the Otsu (1979) three-phase method, Iterative Otsu method, and the Refined statistical-based method calculated void ratios of 0.88, 0.88, and 0.67, respectively. The two Otsu-based methods produced the same average threshold one value of 172. In this context, "average" refers to the average of threshold values calculated for all images in the set. As a result, the Otsu (1979) three-phase method and the Iterative Otsu method yielded the same number of solid pixels, hence explaining the same void ratio value between these methods. The three methods had average threshold two values of 151, 109, and 88, respectively. The value for the degree of saturation varied with threshold two due to degree of saturation being a relationship between water and void pixels. Specifically for the Otsu-based methods, due to the number of solid pixels remaining the same, a lower threshold two value resulted in a higher degree of saturation, and vice versa. The degree of saturation for the Otsu (1979) three-phase method and the Iterative Otsu method were

determined as 16.94% and 41.45%, respectively. Quantitatively, these methods overestimated void ratio. This led to the solid particles in Figure 47a-47b being slightly too eroded. The Otsu (1979) three-phase method greatly underestimated the degree of saturation with more than 50% error, as compared to the degree of saturation obtained from Image-Pro© (visible in Figure 47a). The Iterative Otsu method slightly overestimated the degree of saturation, in regards to Image-Pro©, by approximately 10%. However, as observed in Figure 47b, the segmented silica sand specimen has portions containing water not appearing in the original image slice or portions lacking water and thus containing more voids.

On average, the Refined statistical-based method had the smallest threshold two value and a degree of saturation value of 39.35%. The k1 and k2 parameters for this method were chosen as 1.9 and 2.5, respectively. A trial-and-error process was utilized to obtain parameters that resulted in segmented images which effectively captured the three phases of the raw images. For this method, the average value of threshold one was found to be 158, indicating that more solid pixels and less void spaces were captured in the segmentation. By comparing the results of Figure 47a-47c, it is apparent that Figure 47c accurately contains more bridges of water between the solid particles and less voids within the solid particles, as seen in the original slice (Figure 45). Quantitatively, the void ratio and degree of saturation results of the proposed method are very accurate to the results of Image-Pro©, with approximately 1.5% and 4% errors, respectively. The effectiveness of the proposed method was tested against the Arora et al. (2008) method while qualitatively keeping the segmentations the same (Figure 48). Note that the values of k1 and k2 differ

between the methods since the role of these parameters in the algorithms are not the same.

The proposed method was found to have significantly faster processing time, than both the Arora et al. (2008) and Otsu (1979) three-phase methods, as shown in Table 15. For these reasons, the proposed method proved to be superior to the Arora et al. (2008) method. In conclusion, the segmentation of the silica sand specimen was best captured by the Refined statistical-based method, both qualitatively and quantitatively.

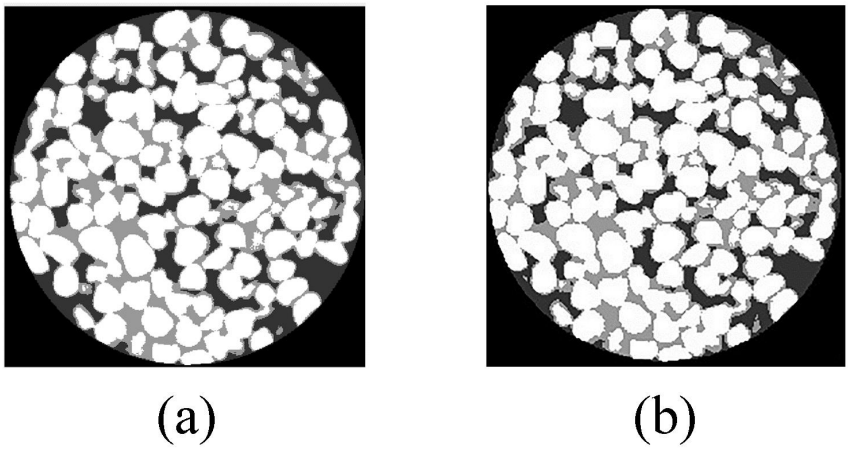

Figure 48. Segmented images of a partially saturated silica sand slice:
(a) Arora et al. (2008) method, (b) Refined statistical-based method.

Table 15. Processing time comparison for the silica sand specimen.

Method	**Processing Time (s)**	**Percent Decrease (Arora to Refined statistical-based)**	**Percent Decrease (Otsu to Refined statistical-based)**
Arora et al. (k1=0.62, k2=3.85)	5.79	70.12%	70.63%
Otsu's (1979) three-phase	5.89		
Refined statistical-based (k1=1.9, k2=2.5)	1.73		

5.2.2 Glass Bead, Water, and Air

The glass bead specimen had an original and cropped diameter of 10 mm and ninety image slices, first image slice depicted as Figure 49, were obtained for this specimen through X-ray CT scanning, as with the silica sand specimen. Figure 50 provides the quantitative results of the applied thresholding techniques (first image slice used for calculating T1) and the segmentation of the first image slice with the Otsu (1979) three-phase method applied. Lastly, Figure 51 presents the segmentation of the first glass bead image slice per thresholding technique. The Image-Pro© void ratio and degree of saturation values were found to be 0.64 and 43.49%, respectively.

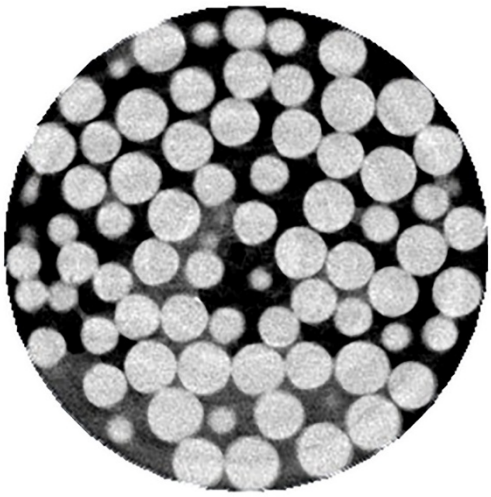

Figure 49. First cropped image slice of a partially saturated glass bead specimen.

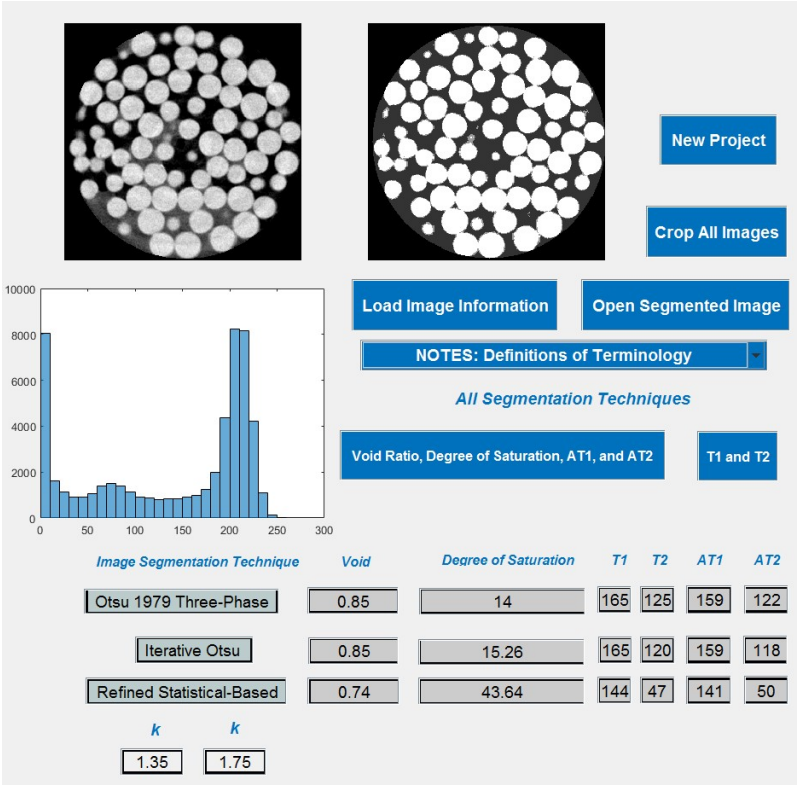

Figure 50. Partially saturated glass bead results of thresholding techniques (displayed image slice segmented with the Otsu (1979) three-phase method).

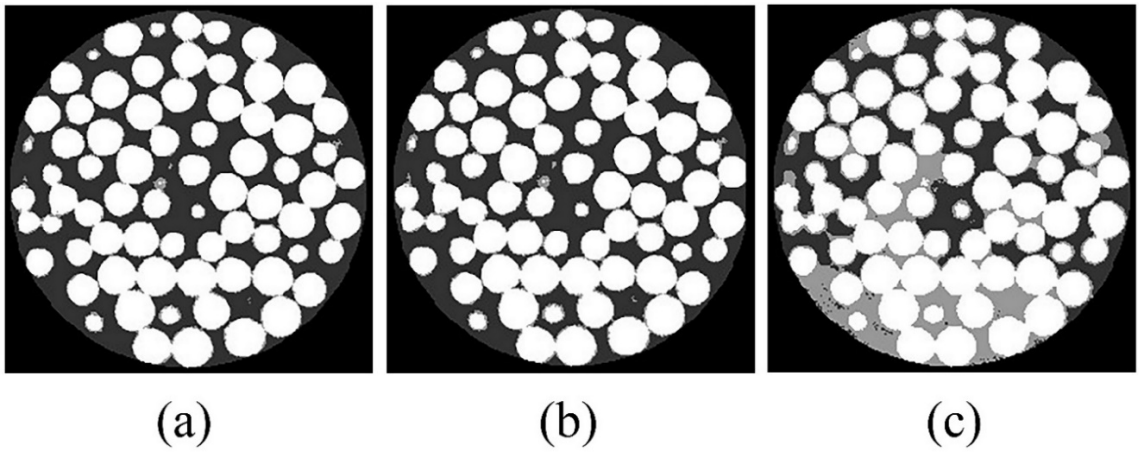

Figure 51. Segmented images of a partially saturated glass bead slice:
(a) Otsu (1979) three-phase method, (b) Iterative Otsu, (c) Refined statistical-based method.

Following the same algorithmic trend as observed with the results of the silica sand specimen, the Otsu (1979) three-phase method and the Iterative Otsu method yielded the same void ratio of 0.85 and average threshold one value of 159. Respectively, the degree of saturations were 14% and 15.26%. By analyzing Figure 51a-51b, the segmentations failed to capture the majority of the water pixels seen in the original slice. This is due to the methods determining threshold two values that were too high. A higher threshold two values results in a segmented image containing less water pixels and more air pixels, hence resulting in a lower degree of saturation. Even though the Iterative Otsu method did a slightly better job at capturing the water pixels than the Otsu (1979) three-phase method, the degree of saturation was still approximately three times smaller than the Image-Pro© value. The void ratio and degree of saturation for the Refined statistical-based method, with k1=1.35 and k2=1.75, were 0.74 and 43.64%, respectively. For this method, the average values of threshold one and threshold two were 141 and 50, respectively.

The proposed method's average threshold two value is more than two times smaller than that of the two Otsu-based techniques. This decrease led to significantly more water pixels being captured, as visible in Figure 51c. Quantitatively, the degree of saturation of the proposed method was very accurately determined, in comparison to the Image-Pro© value, with approximately 0.34% error. As with the silica sand specimen, the superiority of the Refined statistical-based method to the Arora et al. (2008) method was evaluated. Once again, the processing time for the proposed method was much faster (Table 16) with the segmentation of the glass bead specimen remaining the same between the Arora et al. (2008) and proposed methods (Figure 52). Over all, the segmentation of the glass beads

specimen was best captured by the Refined statistical-based method.

Table 16. Processing time comparison for the glass bead specimen.

Method	Processing Time (s)	Percent Decrease (Arora to Refined statistical-based)	Percent Decrease (Otsu to Refined statistical-based)
Arora et al. (k1=0.19, k2=4.55)	6.21	84.54%	80.49%
Otsu's (1979) three-phase	4.92		
Refined statistical-based (k1=1.35, k2=1.75)	0.96		

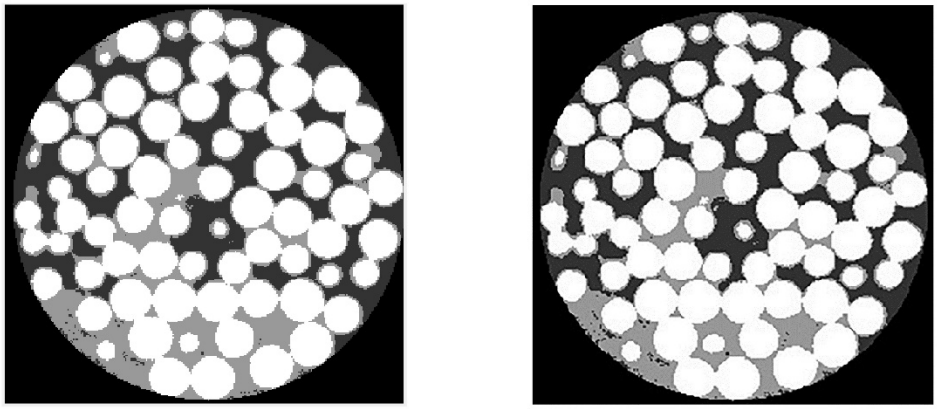

Figure 52. Segmented images of a partially saturated glass bead slice:
(a) Arora et al. (2008) method, (b) Refined statistical-based method.

5.2.3 Summary of Three-Phase Image Segmentation Results

Table 17 lists the final results of the Refined statistical-based method for the partially saturated granular media. Tables 18-21 provide statistical results for the three methods applied to each geomaterial. The proposed method's results for the silica sand

specimen showed that there were moderately low percent errors of 1.52 and 4.35 for void ratio and degree of saturation, respectively, in comparison to the values provided by Image-Pro©. The results of the glass bead specimen interestingly showed that the percent error associated with void ratio was a high 15.63 and the percent error associated with degree of saturation was a very low 0.34.

As indicated previously, threshold one separated solid pixels from water and air pixels and threshold two separated water pixels from air pixels. The percent errors for the silica sand specimen suggested that the fitting parameters, k1 and k2, for the Refined statistical-based method had a greater influence on threshold two and had a low effect on threshold one. On the other hand, the chosen fitting parameters for the glass bead specimen had a great influence on threshold one and not so much for threshold two; threshold two adjusted accordingly to the threshold one value.

These findings led to the hypothesis that the proposed Refined statistical-based method was more accurate at determining one geomaterial property rather than multiple properties (e.g., void ratio and degree of saturation). More images of partially saturated granular media would have to be analyzed to test the validity of this hypothesis. Regardless, the freedom and flexibility of choosing fitting parameters proved this method's superiority to the other two Otsu-based methods. Being able to apply a range of fitting parameters allows the proposed method to be adaptable to a wide range of images.

Table 17. Refined statistical-based method results for the partially saturated granular media.

Granular Media	Image Segmentation Technique	Void Ratio	Degree of Saturation	Percent Error (Void Ratio)	Percent Error (Degree of Saturation)
Silica Sand	Refined statistical-based method (k1=1.9, k2=2.5)	0.67	39.35%	1.52%	4.35%
Glass Bead	Refined statistical-based method (k1=1.35, k2=1.75)	0.74	43.64%	15.63%	0.34%

Table 18. Void ratio statistical results and comparisons for the silica sand specimen.

Geomaterial	Image Segmentation Technique	Void Ratio	Error	Standard Deviation	Coefficient of Variation (CV)
Silica Sand	Otsu's (1979) three-phase method	0.88	0.045	0.099	12.22%
	Iterative Otsu method	0.88	0.045		
	Refined statistical-based method (k1=1.9, k2=2.5)	0.67	0.0001		
	Image-Pro	0.66	--------		

Table 19. Void ratio statistical results and comparisons for the glass bead specimen.

Geomaterial	Image Segmentation Technique	Void Ratio	Error	Standard Deviation	Coefficient of Variation (CV)
Glass Bead	Otsu's (1979) three-phase method	0.85	0.044	0.052	6.38%
	Iterative Otsu method	0.85	0.044		
	Refined statistical-based method (k1=1.35, k2=1.75)	0.74	0.01		
	Image-Pro	0.64	--------		

Table 20. Degree of saturation statistical results and comparisons for the silica sand specimen.

Geomaterial	Image Segmentation Technique	Degree of Saturation	Error	Standard Deviation	Coefficient of Variation (CV)
Silica Sand	Otsu's (1979) three-phase method	16.94%	0.043	0.11	34.05%
	Iterative Otsu method	41.45%	0.0014		
	Refined statistical-based method (k1=1.9, k2=2.5)	39.35%	0.00027		
	Image-Pro	37.71%	--------		

Table 21. Degree of saturation statistical results and comparisons for the glass bead specimen.

Geomaterial	Image Segmentation Technique	Degree of Saturation	Error	Standard Deviation	Coefficient of Variation (CV)
Glass Bead	Otsu's (1979) three-phase method	14%	0.087	0.14	56.32%
	Iterative Otsu method	15.26%	0.080		
	Refined statistical-based method (k1=1.35, k2=1.75)	43.64%	0.0000023		
	Image-Pro	43.49%	--------		

Chapter 6

CONCLUSIONS AND FUTURE WORK

Five automatic thresholding techniques, namely the Otsu (1979), Pun (1980), Kapur et al. (1985), Johannsen and Bille (1982), and Kittler and Illingworth (1986) methods, were chosen for image segmentation of two-phase porous media. Since these techniques dealt with two-phase image segmentation, one threshold value, namely *threshold one*, was determined to separate the foreground/objects and background/air classes from each other. Pixels less than threshold one were referred to as void/air pixels and were colored black, and pixels greater than threshold one were referred to as solid pixels and were colored white. The algorithms for the five chosen thresholding techniques were coded in MATLAB© ultimately to determine material properties such as void ratio (e) for image slices obtained from an X-ray CT device.

The determination of an optimal threshold value varied from technique to technique due to differences in the execution of mathematical algorithms. The Otsu (1979) method found the optimal threshold value through the minimization of the within-class variance of the foreground and background classes of an image's gray-level histogram. The Pun (1980) method made the assumption that pixel information was statistically independent from one another. The Kapur et al. (1985) method chose the optimal threshold value by determining two probability distributions representing the foreground and background classes. The

Johannsen and Bille (1982) method determined the threshold value to be the gray-level pixel value resulting in the minimum interdependence between the foreground and background classes. Lastly, the Kittler and Illingworth (1986) method considered the gray-level histogram to be an estimation of the probability density function of the foreground and background classes. This method assumed that the pixels within these two classes were normally distributed and thus calculated the threshold value through utilization of pixel probability, mean, and standard deviation.

In order to analyze the effectiveness of the thresholding techniques, the five thresholding techniques, discussed above, were applied to pervious concrete, glass bead, and silica sand specimens. Three hundred and thirty-five image slices were analyzed for the pervious concrete specimen, and the cropped size of this specimen was approximately 68 mm in diameter. The method proposed by Kapur et al. (1985) yielded the best results qualitatively and quantitatively (e=0.28) to the laboratory-measured void ratio of 0.26 and the Image-Pro© void ratio of 0.30. Eleven image slices were utilized for the 10 mm in diameter glass bead specimen. Once again, the method proposed by Kapur et al. (1985) gave the best results with an average void ratio of 0.91, as compared to the Image-Pro© void ratio of 0.89. Ten image slices with a cropped diameter of 4.48 mm were used for the analysis of the silica sand specimen. For these set of images, the Otsu (1979) method was the most successful image segmentation technique, yielding a void ratio of 0.85 (Image-Pro© e=0.77).

Interestingly, the results of the applied image segmentation techniques for the three porous media specimens do not follow a specific type of trend. The analysis of the pervious

concrete specimen showed that the Kapur et al. (1985) method was the best technique, whereas the Pun (1980) method was the least accurate for this specimen. For the glass bead specimen, the Kapur et al. (1985) method proved most successful, and the Kittler and Illingworth (1986) method was clearly the least successful. Lastly, the segmentation of the silica sand specimen was best captured by the Otsu (1979) method and was least captured by the Kapur et al. (1985) method (in contrast with the pervious concrete and glass bead specimens). Therefore, it was difficult to make pre-analysis assumptions on which of the five techniques performed the best; the performance of the techniques varied based on the type of porous media analyzed.

The refined statistics-based method was proposed to effectively segment three-phase images of partially saturated granular geomaterials. Two other methods, the Otsu (1979) three-phase and the Iterative Otsu methods, were utilized for relative evaluation of the proposed algorithm. Since these three techniques dealt with three-phase (solids, water, and air) image segmentation, two threshold values, namely *threshold one* and *threshold two*, were required for proper image segmentation. Pixels greater than threshold one and less than threshold two were regarded as belonging to the solid and gas phases, respectively. Pixels between the two threshold-values were taken as belonging to the water phase. The algorithms for the three thresholding techniques were coded into MATLAB© ultimately to determine index properties of the analyzed media such as void ratio and degree of saturation.

The Otsu (1979) three-phase method was first evaluated as a thresholding option, because it is one of the oldest, simplest, and most successful methods for determining

automatic threshold values for different image types. An optimal threshold value was found by minimizing the within-class variance of the foreground and background classes of an image's gray-level histogram. For the two geomaterials analyzed, this method yielded reasonable values for threshold one but not so much for threshold two.

The Iterative Otsu method stemmed from the Otsu (1979) three-phase method in an attempt to more accurately calculate threshold two. The algorithm's loop terminated once the threshold value was within two gray-level pixels of the previous iteration.

A new thresholding technique, namely the refined statistics-based method was proposed in this work. This method was a refinement of the Arora et al. (2008) method which used recursive subrange identification. The proposed method was found to have a faster processing time without diminishing segmentation quality. The percent errors associated with the silica sand specimen showed that segmentation results were more heavily influenced by threshold two than threshold one. The reverse was found to be true for the glass bead specimen. These findings suggested that the proposed method was more accurate at determining one geomaterial property rather than multiple properties.

The refined statistics-based method proved to be a very flexible technique for image segmentation. The technique allowed for user input that enabled the adaptability of the algorithm for various images of interest. The fast execution and flexibility of this algorithm confirmed the superiority of the proposed method to the Otsu-based techniques.

Future work will entail testing the chosen two-phase and three-phase techniques for different soil mixtures, such as mixtures containing silica sand and biochar or soils amended with polymers. This would be done to understand the behavior of soils when

mixed with artificially or naturally produced materials. In addition, the segmented images can be used to investigate the fabric of a soil to extract microstructural parameters for use in constitutive models. In this regard, the role of fabric can be expanded to gain a better understanding of the relationship between different parameters, such as for the parameters used to create a soil-water characteristic curve (SWCC) in unsaturated soils. Regardless, the eight evaluated image segmentation techniques could very well open up more possibilities in the field of multi-phase image segmentation of porous media.

REFERENCES

Aach, T., Schiebel, U. and Spekowius, G. (1999). Digital image acquisition and processing in medical x-ray imaging. Journal of Electronic Imaging 8(1): 7-22.

Abdullah, S. L. S., Hambali, H. A. and Jamil, N. (2012). Segmentation of Natural Images Using an Improved Thresholding-Based Technique. Procedia Engineering 41: 938-944.

Alexander, K., Joly, H., Blond, L., D'Anjou, M. A., Nadeau, M. E., Olive, J. and Beauchamp, G. (2012). A comparison of computed tomography, computed radiography, and film-screen radiography for the detection of canine pulmonary nodules. Vet Radiol Ultrasound 53(3): 258-265.

Arifin, A. Z. and Asano, A. (2006). Image segmentation by histogram thresholding using hierarchical cluster analysis. Pattern Recognition Letters 27(13): 1515-1521.

Armbrust, L. J., Hoskinson, J. J., Biller, D. S., Mackenzie Ostmeyer, R., Milliken, G. A. and Choi, J. (2005). COMPARISON OF DIGITIZED AND DIRECT VIEWED (ANALOG) RADIOGRAPHIC IMAGES FOR DETECTION OF PULMONARY NODULES. Veterinary Radiology & Ultrasound 46(5): 361-367.

Arora, S., Acharya, J., Verma, A. and Panigrahi, P. K. (2008). Multilevel thresholding for image segmentation through a fast statistical recursive algorithm. Pattern Recognition Letters 29(2): 119-125.

Brice, C. R. and Fennema, C. L. (1970). Scene analysis using regions. Artificial Intelligence 1(3): 205-226.

Burk, R. L. and Feeney, D. A. (2003). Small animal radiology and ultrasonography: a diagnostic atlas and text. Philadelphia, Saunders.

Chen, Y. T., Lo, T. N., Chu, Y. S., Yi, J., Liu, C. J., Wang, J. Y., Wang, C. L., Chiu, C. W., Hua, T. E., Hwu, Y., Shen, Q., Yin, G. C., Liang, K. S., Lin, H. M., Je, J. H. and Margaritondo, G. (2008). Full-field hard x-ray microscopy below 30 nm: a challenging nanofabrication achievement. Nanotechnology 19(39): 395302.

Cheriet, M., Said., J. N. and Suen., C. Y. (1998). A recursive thresholding technique for image segmentation. IEEE Transactions on Image Processing 7(6): 918-921.

Christoph, R. and Neumann, H. (2011). X-ray Tomography in Industrial Metrology. Munich, Süddeutscher verlag onpact GmbH.

Chu, Y. S., Yi, J. M., De Carlo, F., Shen, Q., Lee, W.-K., Wu, H. J., Wang, C. L., Wang, J. Y., Liu, C. J., Wang, C. H., Wu, S. R., Chien, C. C., Hwu, Y., Tkachuk, A., Yun, W., Feser, M., Liang, K. S., Yang, C. S., Je, J. H. and Margaritondo, G. (2008). Hard-x-ray microscopy with Fresnel zone plates reaches 40nm Rayleigh resolution. Applied Physics Letters 92(10): 103119.

Cosmi, F. and Bernasconi, A. (2013). Micro-CT investigation on fatigue damage evolution in short fibre reinforced polymers. Composites Science and Technology 79: 70-76.

Cosslett, V. E. (1959). The comparative merits of different methods of microradiography. Applied Scientific Research, Section B 7(1): 338-343.

Da Silva, L., Trebes, J., Balhorn, R., Mrowka, S., Anderson, E., Attwood, D., Barbee, T., Brase, J., Corzett, M., Gray, J. and et, a. (1992). X-ray laser microscopy of rat sperm nuclei. Science 258(5080): 269-271.

De Chiffre, L., Carmignato, S., Kruth, J. P., Schmitt, R. and Weckenmann, A. (2014). Industrial applications of computed tomography. CIRP Annals - Manufacturing Technology 63(2): 655-677.

Ehsan, S., Michael, J. F., Edward, P. and William, R. E. (2003). Subtle Lung Nodules: Influence of Local Anatomic Variations on Detection. Radiology 228(1): 76-84.

Einstein, A. (1918). Bemerkung zu Ernst Gehrckes Notiz ‘Über den Äther’ (Comment on Ernst Gehrcke 'On the Aether'). Verhandlungen der Deutschen Physikalischen Gesellschaft.

Engström, A. (1946). Quantitative micro- and histochemical elementary analysis by Roentgen absoprtion spectrography. Acta Radiologica: 1-106.

Eric, J., Yuan, X. and Ian, M. (2012). Characterisation of voids in fibre reinforced composite materials. NDT & E International: 122-127.

Etaati, A., Wang, H., Pather, S., Yan, Z. and Abdanan Mehdizadeh, S. (2013). 3D X-ray microtomography study on fibre breakage in noil hemp fibre reinforced polypropylene composites. Composites Part B: Engineering 50: 239-246.

Feder, R., Banton, V., Sayre, D., Costa, J., Baldini, M. and Kim, B. (1985). Direct imaging of live human platelets by flash x-ray microscopy. Science 227(4682): 63-64.

Goebbels, J. and Zscherpel, Z. (2011). Proc. of Int. Symposium on Digital Industrial Radiology and Computed Tomography.

Grimm, R., Singh, H., Rachel, R., Typke, D., Zillig, W. and Baumeister, W. (1998). Electron tomography of ice-embedded prokaryotic cells. Biophysical Journal 74(2 Pt 1): 1031-1042.

Heinzl, C. (2009). Analysis and Visualization of Industrial CT Data. Institute of Computer Graphics and Algorithms. Vienna, Austria, Vienna University of Technology.

Holtz, R. D., Kovacs., W. D. and Sheahan., T. C. (2011). An Introduction to Geotechnical Engineering. Upper Saddle River, New Jersey, Pearson.

Horowitz, P. and Howell, J. A. (1972). A Scanning X-Ray Microscope Using Synchrotron Radiation. Science 178(4061): 608-611.

Hsieh, J. (2009). Computed Tomography Principles, Design, Artifacts, and Recent Advances. Hoboken, New Jersey, John Wiley & Sons.

Iassonov, P., Gebrenegus, T. and Tuller, M. (2009). Segmentation of X-ray computed tomography images of porous materials: A crucial step for characterization and quantitative analysis of pore structures. Water Resources Research 45(9): 1-12.

Jacobsen, C. (1992). Making Soft X-Ray Microscopy Harder: Considerations for Sub-0.1 μm Resolution Imaging at ~ 4 Å Wavelengths. X-Ray Microscopy III: Proceedings of the Third International Conference, London, September 3–7, 1990. A. G. Michette, G. R. Morrison and C. J. Buckley. Berlin, Heidelberg, Springer Berlin Heidelberg: 274-277.

Jacobsen, C., Medenwaldt, R. and Williams, S. (1998). A Perspective on Biological X-Ray and Electron Microscopy. X-Ray Microscopy and Spectromicroscopy: Status Report from the Fifth International Conference, Würzburg, August 19–23, 1996. J. Thieme, G. Schmahl, D. Rudolph and E. Umbach. Berlin, Heidelberg, Springer Berlin Heidelberg: 197-206.

Johannsen, G. and Bille, J. (1982). A threshold selection method using information measures. Int. Conf. Pattern Recognition, Munich, Germany.

Kang, H. C., Maser, J., Stephenson, G. B., Liu, C., Conley, R., Macrander, A. T. and Vogt, S. (2006). Nanometer Linear Focusing of Hard X Rays by a Multilayer Laue Lens. Physical Review Letters 96(12): 127401.

Kapur, J. N., Sahoo, P. K. and Wong, A. K. C. (1985). A new method for gray-level picture thresholding using the entropy of the histogram. Computer Vision, Graphics, and Image Processing 29(3): 273-285.

Kastner, J. (2011). X-ray Computed Tomography for the Development of Materials and Components. Vienna, Austria, Vienna University of Technology.

Kastner, J. (2012). Proc. of Int. Conf. on Industrial Computed Tomography, Wels, Austria.

Kirz, J. and Jacobsen, C. (2009). The history and future of X-ray microscopy. Journal of Physics: Conference Series 186: 012001.

Kittler, J. and Illingworth, J. (1986). Minimum error thresholding. Pattern Recognition 19(1): 41-47.

Kohler, R. (1981). A segmentation system based on thresholding. Computer Graphics and Image Processing 15(4): 319-338.

Kruth, J. P., Bartscher, M., Carmignato, S., Schmitt, R., De Chiffre, L. and Weckenmann, A. (2011). Computed tomography for dimensional metrology. CIRP Annals - Manufacturing Technology 60(2): 821-842.

Kurita, Y., Tsuboi, R., Ueki, R., Rifkin, D. B. and Ogawa, H. (1992). Immunohistochemical localization of basic fibroblast growth factor in wound healing sites of mouse skin. Archives of Dermatological Research 284(4): 193-197.

Leedham, G., Yan, C., Takru, K., Joie and Mian, L. (2003). Comparison of some thresholding algorithms for text/background segmentation in difficult document images.

Liao, P., Chen, T. and Chung, P. (2001). A Fast Algorithm for Multilevel Thresholding. J. Inf. Sci. Eng. 17(5): 713-727.

Lu, N. and Likos, W. J. (2004). Unsaturated Soil Mechanics. Hoboken, New Jersey, John Wiley & Sons.

Madra, A., Hajj, N. E. and Benzeggagh, M. (2014). X-ray microtomography applications for quantitative and qualitative analysis of porosity in woven glass fiber reinforced thermoplastic. Composites Science and Technology 95: 50-58.

Maire, E., Babout, L., Buffiere, J. Y. and Fougeres, R. (2001). Recent results on 3D characterisation of microstructure and damage of metal matrix composites and a metallic foam using X-ray tomography. Materials Science and Engineering: A 319–321: 216-219.

Maire, E., Colombo, P., Adrien, J., Babout, L. and Biasetto, L. (2007). Characterization of the morphology of cellular ceramics by 3D image processing of X-ray tomography. Journal of the European Ceramic Society 27(4): 1973-1981.

Malsch, F. (1939). Erzeugung stark vergrößerter Röntgen-Schattenbilder (Generation of highly enlarged x-ray shadows). Naturwissenschaften: 854-855.

Manahiloh, K. N. (2013). Microstructural analysis of unsaturated granular soils using X-ray Computed Tomography. Civil nd Environmental Engineering. Pullman, Washington State University. Ph.D.: 156.

Manahiloh, K. N., Muhunthan, B., Kayhanian, M. and Gebremariam, S. Y. (2012). X-Ray Computed Tomography and Nondestructive Evaluation of Clogging in Porous Concrete Field Samples. Journal of Materials in Civil Engineering 24(8): 1103-1109.

Månsson, L. G., Kheddache, S., Lanhede, B. and Tylén, U. (1999). Image quality for five modern chest radiography techniques: a modified FROC study with an anthropomorphic chest phantom. European Radiology 9(9): 1826-1834.

Maser, J., Stephenson, G. B., Vogt, S., Yun, W., Macrander, A., Kang, H. C., Liu, C. and Conley, R. (2004). Multilayer Laue lenses as high-resolution x-ray optics. Design and Microfabrication of Novel X-Ray Optics II.

Mathworks (2015). MATLAB. Natick, Massachusetts, Mathworks Inc.

Mayr, G., Plank, B., Sekelja, J. and Hendorfer, G. (2011). Active thermography as a quantitative method for non-destructive evaluation of porous carbon fiber reinforced polymers. NDT & E International 44(7): 537-543.

Morrison, G. R., Bridgwater, S., Browne, M. T., Burge, R. E., Cave, R. C., Charalambous, P. S., Foster, G. F., Hare, A. R., Michette, A. G., Morris, D., Taguchi, T. and Duke, P. J. (1989). Development of x-ray imaging at the Daresbury SRS. Review of Scientific Instruments 60(7): 2464-2467.

Müller, P. (2013). Coordinate Metrology by Traceable Computed Tomography, Technical University of Denmark.

Nemanic, S., London, C. A. and Wisner, E. R. (2006). Comparison of Thoracic Radiographs and Single Breath-Hold Helical CT for Detection of Pulmonary Nodules in Dogs with Metastatic Neoplasia. Journal of Veterinary Internal Medicine 20(3): 508-515.

Otsu, N. (1979). Threshold Selection Method from Gray-Level Histograms. IEEE Trans. Syst., Man Cybernetics SMC-9(1): 62-66.

Pal, N. R. and Pal, S. K. (1993). A review on image segmentation techniques. Pattern Recognition 26(9): 1277-1294.

Pun, T. (1980). A new method for grey-level picture thresholding using the entropy of the histogram. Signal Processing 2(3): 223-237.

Razavi, M. R. (2006). Experimental and numerical investigation of shear localization in granular material. Civil and Environmental Engineering. Pullman, Washington State University. PhD Dissertation.

Requena, G., Cloetens, P., Altendorfer, W., Poletti, C., Tolnai, D., Warchomicka, F. and Degischer, H. P. (2009). Sub-micrometer synchrotron tomography of multiphase metals using Kirkpatrick–Baez optics. Scripta Materialia 61(7): 760-763.

Röntgen, W. (1896). Ueber eine neue Art von Strahlen (On a new kind of rays). Conference reports of Würzburg Physik.-medic. Society., Würzburg, Germany.

Russ, J. C. (2015). Forensic Uses of Digital Imaging, Second Edition, CRC Press.

Sahoo, P. K., Soltani, S. and Wong, A. K. C. (1988). A survey of thresholding techniques. Computer Vision, Graphics, and Image Processing 41(2): 233-260.

Salvo, L., Cloetens, P., Maire, E., Zabler, S., Blandin, J. J., Buffière, J. Y., Ludwig, W., Boller, E., Bellet, D. and Josserond, C. (2003). X-ray micro-tomography an attractive characterisation technique in materials science. Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms 200: 273-286.

Sayre, D., Kirz, J., Feder, R., Kim, D. M. and Spiller, E. (1976). Transmission microscopy of unmodified biological materials. Comparative radiation dosages with electrons and ultrasoft X-ray photons. Ultramicroscopy 2: 337-349.

Schindelin, J., Arganda-Carreras, I., Frise, E., Kaynig, V., Longair, M., Pietzsch, T., Preibisch, S., Rueden, C., Saalfeld, S., Schmid, B., Tinevez, J.-Y., White, D. J., Hartenstein, V., Eliceiri, K., Tomancak, P. and Cardona, A. (2012). Fiji: an open-source platform for biological-image analysis. Nat Meth 9(7): 676-682.

Schmahl, G. and Rudolph, D. (1984). Introduction. X-Ray Microscopy: Proceedings of the International Symposium, Göttingen, Fed. Rep. of Germany, September 14–16, 1983. G. Schmahl and D. Rudolph. Berlin, Heidelberg, Springer Berlin Heidelberg: 1-2.

Scott, A. E., Mavrogordato, M., Wright, P., Sinclair, I. and Spearing, S. M. (2011). In situ fibre fracture measurement in carbon–epoxy laminates using high resolution computed tomography. Composites Science and Technology 71(12): 1471-1477.

Seltzer, R., González, C., Muñoz, R., Llorca, J. and Blanco-Varela, T. (2013). X-ray microtomography analysis of the damage micromechanisms in 3D woven composites

under low-velocity impact. Composites Part A: Applied Science and Manufacturing 45: 49-60.

Singh, T. R., Roy, S., Singh, O. I., Sinam, T. and Singh, K. M. (2011). A new local adaptive Thresholding technique in Binarization. International Journal of Computer Science Issues 8(2): 271-277.

Skinner, C. H., DiCicco, D. S., Kim, D., Rosser, R. J., Suckewer, S., Gupta, A. P. and Hirschberg, J. G. (1990). Contact microscopy with a soft X-ray laser. Journal of Microscopy 159(1): 51-60.

Sørensen, T. (2012). CT Scanning in the Medical Device Industry. Industrial Applications of CT Scanning - Possibilities & Challenges in the Manufacturing Industry. Kongens Lyngby, Denmark.

Sund, R. and Eilertsen, K. (2003). An algorithm for fast adaptive image binarization with applications in radiotherapy imaging. IEEE Transactions on Medical Imaging 22: 22-28.

Trebes, J. E., Brown, S. B., Campbell, E. M., Matthews, D. L., Nilson, D. G., Stone, G. F. and Whelan, D. A. (1987). Demonstration of X-ray Holography with an X-ray Laser. Science 238(4826): 517-519.

Tsai, D. (1995). A fast thresholding selection procedure for multimodal and unimodal histograms. Pattern Recognition Letters 16(6): 653-666.

Weckenmann, A., Krämer, P. and Potschies, B. (2008). Artefact for 3D CT measurements of electronic assemblies. Industrielle Computertomografie.

Wen, C. Y. and Chen, J. K. (2004). Multi-resolution image fusion technique and its application to forensic science. Forensic Science International 140(2–3): 217-232.

Woodard, P. K., Slone, R. M., Sagel, S. S., Fleishman, M. J., Gutierrez, F. R., Reiker, G. G., Pilgram, T. K. and Jost, R. G. (1998). Detection of CT-proved pulmonary nodules: comparison of selenium-based digital and conventional screen-film chest radiographs. Radiology 209(3): 705-709.

Yang, Q. and Kang, W. (2009). General Research on Image Segmentation Algorithms International Journal of Image, Graphics and Signal Processing 1: 1-8.

Yin, G., Tang, M.-T., Song, Y.-F., Chen, F.-R., Liang, K. S., Duewer, F. W., Yun, W., Ko, C.-H. and Shieh, H.-P. D. (2006). Energy-tunable transmission x-ray microscope for differential contrast imaging with near 60nm resolution tomography. Applied Physics Letters 88(24): 241115.

Appendix A

DIRECTIONS FOR PROPER GUI USAGE

Note: The same directions apply to both the two-phase and multi-phase GUIs, since the interfaces were created in the same manner.

1. Click “Rename Images.” This button renames all of the images in the folder of interest. Make sure that the original images are saved to a folder.
2. Click “Crop All Images.” This button crops the renamed images and saves the cropped images to a folder of choice. In order to apply cropping, please double click within the drawn ellipse.
3. Click “Load Image.” This button loads the first cropped image and its corresponding histogram. If necessary, the contrast and brightness of the image can be adjusted as well, for visual purposes. The slider allows for the viewing of all of the remaining cropped images and their histograms.
4. After choosing a segmentation method, click “Run.” This button allows for the selection of all of the cropped images. For the two-phase GUI, the void ratio (e) and average threshold value for all the cropped images are displayed. For the multi-phase GUI, the void ratio (e), degree of saturation, and the two average threshold values for all of the

cropped images are displayed. The segmentation of the first cropped image is shown and the slider allows for the viewing of the remaining cropped images with the chosen segmentation method applied.

5. The “Export Data to Excel” button saves the void (e) and average threshold values for all images, per segmentation technique, to an Excel file for the two-phase GUI. In regards to the multi-phase GUI, the void ratio (e), degree of saturation, and the two average threshold values are exported.

Appendix B

TWO-PHASE IMAGE SEGMENTATION GUI CODE

```
      function varargout = TwoPhaseImageSegmentation(varargin)

      gui_Singleton = 1;
      gui_State = struct('gui_Name',       mfilename, ...
                         'gui_Singleton',  gui_Singleton, ...
                         'gui_OpeningFcn',
@TwoPhaseImageSegmentation_OpeningFcn, ...
                         'gui_OutputFcn',
@TwoPhaseImageSegmentation_OutputFcn, ...
                         'gui_LayoutFcn',  [], ...
                         'gui_Callback',   []);
      if nargin && ischar(varargin{1})
         gui_State.gui_Callback = str2func(varargin{1});
      end

      if nargout
          [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
      else
          gui_mainfcn(gui_State, varargin{:});
      end

      % --- Executes just before TwoPhaseImageSegmentation GUI is
visible.
      function TwoPhaseImageSegmentation_OpeningFcn(hObject, eventdata,
handles, varargin)
      set(gcf, 'units', 'normalized', 'position', [.5 .5 .4 .55])

      % Choose default command line output for TwoPhaseImageSegmentation
      handles.output = hObject;

      % Update handles structure
      guidata(hObject, handles);

      % UIWAIT makes TwoPhaseImageSegmentation wait for user response
(see UIRESUME)

      set(handles.axes1,'DataAspectRatio',[2 3 3]);
      set(handles.axes4,'DataAspectRatio',[2 3 3]);
      set(handles.axes2,'DataAspectRatio',[2 3 3]);
      set(handles.axes3,'DataAspectRatio',[2 3 3]);
```

```
      % --- Outputs from this function are returned to the command line.
      function varargout = TwoPhaseImageSegmentation_OutputFcn(hObject,
eventdata, handles)

      % Get default command line output from handles structure
      varargout{1} = handles.output;

      % --- Executes on button press in RenameImages.
      function RenameImages_Callback(hObject, eventdata, handles)
      a = uigetdir;
      A =dir( fullfile(a, '*.bmp') );
      fileNames = { A.name };
      for iFile = 1 : numel( A )
        newName = fullfile(a, sprintf( '1.%02d.bmp', iFile ) );
        movefile( fullfile(a, fileNames{ iFile }), newName );
      end

      AA =dir( fullfile(a, '*.tif') );
      fileNames = { AA.name };
      for iFile = 1 : numel( AA )
        newName = fullfile(a, sprintf( '1.%02d.bmp', iFile ) );
        movefile( fullfile(a, fileNames{ iFile }), newName );
      end

      AA =dir( fullfile(a, '*.jpg') );
      fileNames = { AA.name };
      for iFile = 1 : numel( AA )
        newName = fullfile(a, sprintf( '1.%02d.bmp', iFile ) );
        movefile( fullfile(a, fileNames{ iFile }), newName );
      end

      AA =dir( fullfile(a, '*.png') );
      fileNames = { AA.name };
      for iFile = 1 : numel( AA )
        newName = fullfile(a, sprintf( '1.%02d.bmp', iFile ) );
        movefile( fullfile(a, fileNames{ iFile }), newName );
      end

      AA =dir( fullfile(a, '*.gif') );
      fileNames = { AA.name };
      for iFile = 1 : numel( AA )
        newName = fullfile(a, sprintf( '1.%02d.bmp', iFile ) );
        movefile( fullfile(a, fileNames{ iFile }), newName );
      end

      function numImages_Callback(hObject, eventdata, handles)
      updateSlider(handles);

      function updateSlider(handles)
      % This function updates the slider to have the correct min, max,
value, and
      % step size

      % Get the current slice number which were stored in the figure
```

```
axes
      sliceNum = getappdata(handles.axes1,'sliceNum');
      if(isempty(sliceNum))%may be empty if the figure has not been
initialized
          sliceNum = 1;    %set it to a default
      end

      % Get the number written in the text box which is the maximum
number of
      % images to be viewed
      NumImageslice = str2double(get(handles.numImages,'String'));

      % There are only NumImageslice - 1 images total, because we start
at 1
      step = 1/(NumImageslice - 1);

      % Set values for the slider bar
      set(handles.imageSlider, 'Max', NumImageslice);
      set(handles.imageSlider, 'Min', 1);
      set(handles.imageSlider, 'SliderStep', [step step]);

      % Set current value to the slice we are viewing
      set(handles.imageSlider, 'Value', sliceNum);

      % --- Executes during object creation, after setting all
properties.
      function numImages_CreateFcn(hObject, eventdata, handles)
      % hObject    handle to numImages (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    empty - handles not created until after all
CreateFcns called

      % Hint: edit controls usually have a white background on Windows.
      %       See ISPC and COMPUTER.
      if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
          set(hObject,'BackgroundColor','white');
      end

      function CropButton_Callback(hObject, eventdata, handles)
      axes(handles.axes1);
      AA = uigetdir('','Please Select the Folder to Save the Cropped
Images to');
      cd (AA);
      [FileName,PathName] =
uigetfile({'*.jpg;*.tif;*.bmp;*.png;*.gif','All Image Files'},'Please
Select First Renamed Image');

      imagefiles1 = dir([PathName '*.jpg']);
      imagefiles2 = dir([PathName '*.tif']);
      imagefiles3 = dir([PathName '*.bmp']);
      imagefiles4 = dir([PathName '*.png']);
      imagefiles5 = dir([PathName '*.gif']);
```

```
      imagefiles = [imagefiles1; imagefiles2; imagefiles3; imagefiles4;
imagefiles5];
      nfiles = length(imagefiles);    % Number of files found

      image = imread([PathName FileName]);

      % Following lines to change image matrix into a square sized
matrix

      image=im2uint8(image);
      SIZI=size(image);
      if numel(SIZI)>2
          if SIZI(1,3)>=3
              image=image(:,:,1:3);
              image=rgb2gray(image);
          end
      end

          % Reducing the image matrix into a square sized matrix

          if SIZI(1,1)~=SIZI(1,2)
              SIZim=size(image);
              Divs=round((max(size(image))-min(size(image)))/2);
              if SIZim(1,1)== max(size(image))
                  image=image(Divs+1:SIZim(1,1)-Divs+1,:);
              else
                  image=image(:,Divs+1:SIZim(1,2)-Divs+1);
              end
          end

      imshow(image) %needed to use imellipse
      user_defined_ellipse = imellipse(gca, []); % creates user defined
ellipse object.

      MASK = double(user_defined_ellipse.createMask());
      addNewPositionCallback(user_defined_ellipse,@(p)
title(mat2str(p,3)));

      fcn =
makeConstrainToRectFcn('imellipse',get(gca,'XLim'),get(gca,'YLim'));
      accepted_pos=wait(user_defined_ellipse); % You need to click twice
to continue.
      MASK = double(user_defined_ellipse.createMask());

      for i=1:nfiles

         currentfilename = imagefiles(i).name;
         currentimage = imread([PathName currentfilename]);

         currentimage=im2uint8(currentimage);
          SIZI=size(currentimage);
          if numel(SIZI)>2
              if SIZI(1,3)>=3
                  currentimage=currentimage(:,:,1:3);
```

```
                currentimage=rgb2gray(currentimage);
            end
        end

        if SIZI(1,1)~=SIZI(1,2)
            SIZim=size(currentimage);
            Divs=round((max(size(currentimage))-
min(size(currentimage)))/2);
            if SIZim(1,1)== max(size(currentimage))
                currentimage=currentimage(Divs+1:SIZim(1,1)-Divs+1,:);
            else
                currentimage=currentimage(:,Divs+1:SIZim(1,2)-Divs+1);
            end
        end
       new_image_name = [PathName 'CroppedImage_' currentfilename];
       [pathstr,name,ext] = fileparts(new_image_name);
       new_image_name=fullfile(name);
       new_image_name = [new_image_name '.png']; % making the image
.png so it can be transparent
       imwrite(currentimage, new_image_name,'png','Alpha',MASK);
    end

    msg = msgbox('Congratulations! All images have been cropped!');
    waitfor(msg);
    % --- Executes on slider movement.
    function imageSlider_Callback(hObject, eventdata, handles)
    % hObject    handle to imageSlider (see GCBO)
    % eventdata  reserved - to be defined in a future version of
MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Hints: get(hObject,'Value') returns position of slider
    %        get(hObject,'Min') and get(hObject,'Max') to determine
range of slider
    %get the current value on the slider
    imageSlider_value = get(hObject,'Value');

    % Get the current max value from the slider
    numImages = get(handles.imageSlider, 'Max');

    % Calculate the image number to display
    imageNum = floor(imageSlider_value)...
             + (sign(imageSlider_value)...
             * ( abs(imageSlider_value) - floor(imageSlider_value)
)...
             * numImages);

    %Create a string from the number which always has 2 digits
    str = sprintf('%02.0f',imageNum);

    %Retrieve filename data from app data
    ptNum    = getappdata(handles.axes1, 'ptNum');
    imageType = getappdata(handles.axes1, 'imageType');
    filepath = getappdata(handles.axes1, 'filePath');
```

```
%Create the filename as a cell string
filename = strcat(ptNum, '.', str, '.', imageType);

%Create full path to the image
imageStr = [filepath, filename{1}];

%Read in image data
image = imread(imageStr);

%Bring current axes in focus and show image
axes(handles.axes1);
imshow(image, []); %[] = [Imin Imax]

axes(handles.axes4);
imshow(image,[]);

%Store image data and slice number in axes
setappdata(handles.axes1, 'image',     image);
setappdata(handles.axes1, 'sliceNum',  imageSlider_value);

axes(handles.axes2);
sizeofmatrix = size(image,1);
[Row,Col,r]=MaskPortion(image);
c=[];
c1=zeros(sizeofmatrix,sizeofmatrix);
c1=uint8(c1);
e=0;

for i=1:sizeofmatrix
    for j=1:sizeofmatrix
        d=(j - (Row))^2 + (i - (Col))^2;
        if d <= (r)^2
            e=e+1;
            c(e)=image(i,j);
            c1(i,j)=image(i,j);
        end
    end
end
histogram(c);

set(handles.CurrentName, 'String', filename)

 contents = get(handles.Methods,'String');
  MethodName = contents{get(handles.Methods,'Value')};

 if strcmp(MethodName, 'Otsu ')
          axes(handles.axes3);
          sizeofmatrix = size(image,1);
          [Row,Col,r]=MaskPortion(image);
          c=[];
          c1=zeros(sizeofmatrix,sizeofmatrix);
          c1=uint8(c1);
          e=0;
```

```
            for i=1:sizeofmatrix
                for j=1:sizeofmatrix
                    d=(j - (Row))^2 + (i - (Col))^2;
                    if d <= (r)^2
                        e=e+1;
                        c(e)=image(i,j);
                        c1(i,j)=image(i,j);
                    end
                end
            end

    e=uint8(c);
    p=imhist(e);
    mean=0;
    for i=1:256
        mean=mean+(i*p(i,1));
    end
    mean=mean/256;
    sumt=zeros(256,1);
    for i=1:256
        meanli=0;
        for j=1:i-1
            meanli=meanli+(j*p(j,1));
        end
        meanli=meanli/256;
        meangi=0;
        for j=i:256
            meangi=meangi+(j*p(j,1));
        end
        meangi=meangi/256;
        sumli=0;
        sumgi=0;
        for ii=1:i-1
            sumli=sumli+p(ii,1)*((meanli-mean)^2);
        end

        for ii=i:256
            sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
        end

        sumt(i,1)=sumgi+sumli;

    end

    Threshold1=find(sumt==max(sumt));

    % Portion of Code that Calculates the Porosity and Void Ratio of
the Image

    c2=zeros(sizeofmatrix,sizeofmatrix);
    c2=uint8(c2);
    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
            d=(j - (Row))^2 + (i - (Col))^2;
```

```
        if d <= (r)^2
            if c1(i,j)<Threshold1
                c2(i,j)=20;
            else
                c2(i,j)=100;
            end

        end
    end
end

imshow(c2,[]);
 end

%%%

      if strcmp(MethodName, 'Pun ')
          axes(handles.axes3);
          sizeofmatrix = size(image,1);
          [Row,Col,r]=MaskPortion(image);
          c=[];
          c1=zeros(sizeofmatrix,sizeofmatrix);
          c1=uint8(c1);
          e=0;
          for i=1:sizeofmatrix
              for j=1:sizeofmatrix
                  d=(j - (Row))^2 + (i - (Col))^2;
                  if d <= (r)^2
                      e=e+1;
                      c(e)=image(i,j);
                      c1(i,j)=image(i,j);
                  end
              end
          end

% Automatic Threshold using Pun's Method

e=uint8(c);
p=imhist(e);
n=sum(p);
pi=zeros(256,1);

for i=1:256
      pi(i)=p(i)/n;
end

for i=1:256
      Sumpi=sum(pi(1:i));
      if Sumpi>=0.5
          m=i;
          break
      end
end
```

```
        numerator = pi(1:m).*log(pi(1:m));
        numerator = nansum(numerator);

        for i=1:256
        denominator = pi(1:i).*log(pi(1:i));
        end

        denominator = nansum(denominator);

        alpha = numerator/denominator;

        if alpha <=0.5
            correctedalpha = 1-alpha;
        else
            correctedalpha = alpha;
        end


        for i=1:256
          Summationpi=sum(pi(1:i));
          if Summationpi>=correctedalpha
              Threshold = i;
              break
          end
        end

        Threshold1 = Threshold;

    % Portion of Code that Calculates the Porosity and Void Ratio of
the Image

    c2=zeros(sizeofmatrix,sizeofmatrix);
    c2=uint8(c2);
    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
            d=(j - (Row))^2 + (i - (Col))^2;
            if d <= (r)^2
                if c1(i,j)<Threshold1
                    c2(i,j)=20;
                else
                    c2(i,j)=100;
                end

            end
        end
    end
    imshow(c2,[]);

          end

          %%%

          if strcmp(MethodName, 'Kapur, Sahoo, and Wong')
              axes(handles.axes3);
```

```
          sizeofmatrix = size(image,1);
          [Row,Col,r]=MaskPortion(image);
          c=[];
          c1=zeros(sizeofmatrix,sizeofmatrix);
          c1=uint8(c1);
          e=0;

          for i=1:sizeofmatrix
              for j=1:sizeofmatrix
                  d=(j - (Row))^2 + (i - (Col))^2;
                  if d <= (r)^2
                      e=e+1;
                      c1(i,j)=image(i,j);
                  end
              end
          end

% Automatic Threshold using Kapur, Sahoo, and Wong Method

e=uint8(c);
p=imhist(e);
n=sum(p);
pi=zeros(256,1);
for i=1:256
    pi(i)=p(i)/n;
end

pi(1:256,2)=linspace(1,256,256);
N=0;
for q=1:256
    if pi(q,1)~=0
        N=N+1;
        pi1(N,1)=pi(q,1);
        pi1(N,2)=pi(q,2);
    end
end

pi1= pi1(pi1(:,1)>0.0001,:);

size(pi1,1);
c=size(pi1,1);

sumt = zeros(c,1);
Ht= zeros(c,1);
Pt=zeros(c,1);
Ha=zeros(c,1);
Hb=zeros(c,1);

for t = 1:c

      Hi= -sum(pi1(1:c).*log(pi1(1:c)));

     for i = 1:c
         Pt(i)=0;
```

```
            Ht(t)=0;
            Pt(i) = sum(pi1(1:i))+Pt(i);
            Ht(t) = -sum(pi1(1:t).*log(pi1(1:t)))+Ht(t);
            Ha(i) = log(Pt(i))+(Ht(i)/Pt(i));
            Hb(i) = log(1-Pt(i))+((Hi-Ht(i))/(1-Pt(i)));
            Hb(i)=real(Hb(i));
        end

        sumt=Ha+Hb;
        sumt(:,2)=pi1(:,2);
       maxrow=max(sumt(:,1));
       Threshold1=max(sumt(sumt(:,1)==maxrow,2));

    end

    % Portion of Code that Calculates the Porosity and Void Ratio of
the Image

    c2=zeros(sizeofmatrix,sizeofmatrix);
    c2=uint8(c2);
    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
            d=(j - (Row))^2 + (i - (Col))^2;
            if d <= (r)^2
                if c1(i,j)<Threshold1
                    c2(i,j)=20;
                else
                    c2(i,j)=100;
                end

            end
        end
    end
    imshow(c2,[]);

        end

    %%%

          if strcmp(MethodName, 'Johannsen and Bille')
              axes(handles.axes3);
              sizeofmatrix = size(image,1);
              [Row,Col,r]=MaskPortion(image);
              c=[];
              c1=zeros(sizeofmatrix,sizeofmatrix);
              c1=uint8(c1);
              e=0;
              for i=1:sizeofmatrix
                  for j=1:sizeofmatrix
                      d=(j - (Row))^2 + (i - (Col))^2;
                      if d <= (r)^2
                          e=e+1;
                          c(e)=image(i,j);
                          c1(i,j)=image(i,j);
```

```
                    end
                end
            end

    % Automatic Threshold using Johannsen and Bille Method

    e=uint8(c);
    p=imhist(e);
    n=sum(p);
    pi=zeros(256,1);
    for i=1:256
        pi(i)=p(i)/n;
    end

    pi(1:256,2)=linspace(1,256,256);
    N=0;
    for q=1:256
        if pi(q,1)~=0
            N=N+1;
            pi1(N,1)=pi(q,1);
            pi1(N,2)=pi(q,2);
        end
    end

    pi1= pi1(pi1(:,1)>0.0001,:);

    size(pi1,1);
    c=size(pi1,1);

       sumt = zeros(c,1);
       Pn=zeros(c,1);
       Pt=zeros(c,1);
       Pi=zeros(c,1);
       Pr=zeros(c,1);
       Sn=zeros(c,1);
       Sn_bar=zeros(c,1);

    for t=1:c
        n=t+1;

        Pn(t)=sum(pi1(1:n));
        Pt(t)=sum(pi1(1:n-1));

        Sn(t)=(log(Pn(t)))-
(1/Pn(t)).*(((pi1(n)).*log(pi1(n)))+(Pt(t).*log(Pt(t))));
        Sn(t)=real(Sn(t));

        Pi(t)=sum(pi1(n:c));
        Pr(t)=sum(pi1(n+1:c));

        Sn_bar(t)=(log(Pi(t)))-
(1/Pi(t)).*(((pi1(n)).*log(pi1(n)))+(Pr(t).*log(Pr(t))));
        Sn_bar(t)=real(Sn_bar(t));
```

```
        sumt=Sn+Sn_bar;
        sumt(:,2)=pi1(:,2);
        minrow=min(sumt(:,1));
        Threshold1=min(sumt(sumt(:,1)==minrow,2));

    end

    % Portion of Code that Calculates the Porosity and Void Ratio of
the Image

    c2=zeros(sizeofmatrix,sizeofmatrix);
    c2=uint8(c2);
    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
            d=(j - (Row))^2 + (i - (Col))^2;
            if d <= (r)^2
                if c1(i,j)<Threshold1
                    c2(i,j)=20;
                else
                    c2(i,j)=100;
                end

            end
        end
    end
    imshow(c2,[]);
           end

    %%%

           if strcmp(MethodName, 'Kittler and Illingworth')
               axes(handles.axes3);
               sizeofmatrix = size(image,1);
               [Row,Col,r]=MaskPortion(image);
               c=[];
               c1=zeros(sizeofmatrix,sizeofmatrix);
               c1=uint8(c1);
               e=0;
               for i=1:sizeofmatrix
                   for j=1:sizeofmatrix
                       d=(j - (Row))^2 + (i - (Col))^2;
                       if d <= (r)^2
                           e=e+1;
                           c(e)=image(i,j);
                           c1(i,j)=image(i,j);
                       end
                   end
               end

    % Automatic Threshold using Kittler and Illingworth Method

    e=uint8(c);
    p=imhist(e);
    n=sum(p);
```

```
pi=zeros(256,1);
for i=1:256
    pi(i)=p(i)/n;
end

pi(1:256,2)=linspace(1,256,256);
N=0;
for q=1:256
    if pi(q,1)~=0
        N=N+1;
        pi1(N,1)=pi(q,1);
        pi1(N,2)=pi(q,2);
    end
end

size(pi1,1);
c=size(pi1,1);

   P1=zeros(c,1);
   P2=zeros(c,1);
   u1=zeros(c,1);
   u2=zeros(c,1);
   s1=zeros(c,1);
   s1_d=zeros(c,1);
   s1_final=zeros(c,1);
   s2_final=zeros(c-2,1);
   J=zeros(c-2,1);

   for i=1:c
       g=1:i;
       g1=i+1:c-1;
       P1(i)=sum(pi1(1:i));
       P2(i)=sum(pi1(i+1:c-1));
       u1(i)=sum(pi1(1:i).*g)/P1(i);
       u2(i)=sum(pi1(i+1:c-1).*g1)/P2(i);
       u2=u2(isfinite(u2));
       u2=u2(u2~=0);
       u2_final=u2';
       s1(i)=0;

       for ii=1:i
       s1(i)=((ii-u1(i))^2*pi1(ii));
       s1_n=cumsum(s1);
       s1_d(i)=P1(ii);
       s1_dd=s1_d;
       s1_final=s1_n./s1_dd;

       end

           G3=([2:c-1]);
           G4=G3';
           A=u2_final(:,1);

   end
```

```
         B=pi1([2:c-1]);
         Pi1_adj=B';
        s2_n=Pi1_adj.*(G4-A).^2;

        for i = 1:c
            for ii = i:c-2
            s2_final(ii)=sum(s2_n(i+1:c-2))/P2(ii);

            end

        end

        s1_final=sqrt(s1_final);
        s2_final=sqrt(s2_final);

       P1=P1([1:c-2]);
       P2=P2([1:c-2]);
       s1_final=s1_final([1:c-2]);
      J=1+2.*((P1.*log(s1_final))+(P2.*log(s2_final)))-
2.*((P1.*log(P1))+(P2.*log(P2)));
      J=J(~isinf(J));

      minrow=max(J);
      Threshold1=max(find(J>=minrow));

      % Portion of Code that Calculates the Porosity and Void Ratio of
the Image

      c2=zeros(sizeofmatrix,sizeofmatrix);
      c2=uint8(c2);
      for i=1:sizeofmatrix
          for j=1:sizeofmatrix
              d=(j - (Row))^2 + (i - (Col))^2;
              if d <= (r)^2
                  if c1(i,j)<Threshold1
                      c2(i,j)=20;
                  else
                      c2(i,j)=100;
                  end

              end
          end
      end
      imshow(c2,[]);
             end

      % --- Executes during object creation, after setting all
properties.
      function imageSlider_CreateFcn(hObject, eventdata, handles)
      % hObject    handle to imageSlider (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    empty - handles not created until after all
```

```
CreateFcns called

      % Hint: slider controls usually have a light gray background.
      if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
          set(hObject,'BackgroundColor',[.9 .9 .9]);
      end

      function CurrentName_Callback(hObject, eventdata, handles)
      % hObject    handle to CurrentName (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)

      % Hints: get(hObject,'String') returns contents of CurrentName as
text
      %        str2double(get(hObject,'String')) returns contents of
CurrentName as a double

      set(handles.CurrentName, 'String', [pathName filename])

      % --- Executes during object creation, after setting all
properties.
      function CurrentName_CreateFcn(hObject, eventdata, handles)
      % hObject    handle to CurrentName (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    empty - handles not created until after all
CreateFcns called

      % Hint: edit controls usually have a white background on Windows.
      %       See ISPC and COMPUTER.
      if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
          set(hObject,'BackgroundColor','white');
      end

      % --- Executes on button press in AdjustContrastandBrightness.
      function AdjustContrastandBrightness_Callback(hObject, eventdata,
handles)
      % hObject    handle to AdjustContrastandBrightness (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)
      %
set(handles.imageSlider,'Value',get(handles.imageSlider,'Min'));
      %
      axes(handles.axes4);
      imcontrast(gcf);

      % --- Executes on button press in Run.
      function Run_Callback(hObject, eventdata, handles)
       contents = get(handles.Methods,'String');
       MethodName = contents{get(handles.Methods,'Value')};
```

```
       getappdata(handles.axes1,'fileName');

      number_Of_Air_T=0;
      number_Of_Solid_T=0;

      contents = get(handles.Methods,'String');
      MethodName = contents{get(handles.Methods,'Value')};
      getappdata(handles.axes1,'fileName');

      %%%

      if strcmp(MethodName, 'Otsu ')
       set(handles.imageSlider,'Value',get(handles.imageSlider,'Min'));
       d = uigetdir('','Please Select the Folder Containing the Cropped
Images');
       cd(d);
       q=dir('*.png');
       b = numel(q);

       I=imread('CroppedImage_1.01.png');
       [Row,Col,r]=MaskPortion(I);
       sizeofmatrix = size(I,1);

      c=[];
      c1=zeros(sizeofmatrix,sizeofmatrix);
      c1=uint8(c1);
      e=0;
      for i=1:sizeofmatrix
          for j=1:sizeofmatrix
              d=(j - (Row))^2 + (i - (Col))^2;
              if d <= (r)^2
                  e=e+1;
                  c(e)=I(i,j);
                  c1(i,j)=I(i,j);
              end
          end
      end

      % Automatic Threshold using Otsu's Method

      e=uint8(c);
      p=imhist(e);
      mean=0;
      for i=1:256
          mean=mean+(i*p(i,1));
      end
      mean=mean/256;
      sumt=zeros(256,1);
      for i=1:256
          meanli=0;
          for j=1:i-1
              meanli=meanli+(j*p(j,1));
          end
          meanli=meanli/256;
```

```
        meangi=0;
        for j=i:256
            meangi=meangi+(j*p(j,1));
        end
        meangi=meangi/256;
        sumli=0;
        sumgi=0;
        for ii=1:i-1
            sumli=sumli+p(ii,1)*((meanli-mean)^2);
        end

        for ii=i:256
            sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
        end

        sumt(i,1)=sumgi+sumli;

    end

    Threshold1=find(sumt==max(sumt));

    % Portion of Code that Calculates the Porosity and Void Ratio of
the Image

    c2=zeros(sizeofmatrix,sizeofmatrix);
    c2=uint8(c2);
    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
            d=(j - (Row))^2 + (i - (Col))^2;
            if d <= (r)^2
                if c1(i,j)<Threshold1
                    c2(i,j)=20;
                else
                    c2(i,j)=100;
                end

            end
        end
    end

    axes(handles.axes3);
    imshow(c2,[]);

    for ka = 1: b
       I = sprintf('CroppedImage_1.%02d.png', ka);
       I=imread(I);
       [Row,Col,r]=MaskPortion(I);
    sizeofmatrix = size(I,1);

    c=[];
    c1=zeros(sizeofmatrix,sizeofmatrix);
    c1=uint8(c1);
    e=0;
    for i=1:sizeofmatrix
```

```
        for j=1:sizeofmatrix
            d=(j - (Row))^2 + (i - (Col))^2;
            if d <= (r)^2
                e=e+1;
                c(e)=I(i,j);
                c1(i,j)=I(i,j);
            end
        end
    end

    % Automatic Threshold using Otsu's Method

    e=uint8(c);
    p=imhist(e);
    mean=0;
    for i=1:256
        mean=mean+(i*p(i,1));
    end
    mean=mean/256;
    sumt=zeros(256,1);
    for i=1:256
        meanli=0;
        for j=1:i-1
            meanli=meanli+(j*p(j,1));
        end
        meanli=meanli/256;
        meangi=0;
        for j=i:256
            meangi=meangi+(j*p(j,1));
        end
        meangi=meangi/256;
        sumli=0;
        sumgi=0;
        for ii=1:i-1
            sumli=sumli+p(ii,1)*((meanli-mean)^2);
        end

        for ii=i:256
            sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
        end

        sumt(i,1)=sumgi+sumli;

    end

    Threshold1=find(sumt==max(sumt));

    % Portion of Code that Calculates the Porosity and Void Ratio of
the Image

    c2=zeros(sizeofmatrix,sizeofmatrix);
    c2=uint8(c2);
    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
```

```
            d=(j - (Row))^2 + (i - (Col))^2;
            if d <= (r)^2
                if c1(i,j)<Threshold1
                    c2(i,j)=20;
                else
                    c2(i,j)=100;
                end

            end
        end
    end

    number_Of_Air_Pixels=sum(sum(c2==20));
    number_Of_Solid_Pixels=sum(sum(c2==100));

    number_Of_Air_T=number_Of_Air_T+number_Of_Air_Pixels;
    number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;

    end

    void_ratio = number_Of_Air_T/number_Of_Solid_T;


    v=void_ratio;
    v=num2str(v);
    set(handles.VoidText, 'String', v);
    end

    %%%

    number_Of_Air_T=0;
    number_Of_Solid_T=0;

           if strcmp(MethodName, 'Pun ')

set(handles.imageSlider,'Value',get(handles.imageSlider,'Min'));
                d = uigetdir('','Please Select the Folder Containing
the Cropped Images');
                cd(d);
                q=dir('*.png');
                b = numel(q);

        I=imread('CroppedImage_1.01.png');
        [Row,Col,r]=MaskPortion(I);
        sizeofmatrix = size(I,1);

    c=[];
    c1=zeros(sizeofmatrix,sizeofmatrix);
    c1=uint8(c1);
    e=0;
    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
            d=(j - (Row))^2 + (i - (Col))^2;
            if d <= (r)^2
```

```
            e=e+1;
            c(e)=I(i,j);
            c1(i,j)=I(i,j);
        end
    end
end

% Automatic Threshold using Pun's Method

e=uint8(c);
p=imhist(e);
n=sum(p);
pi=zeros(256,1);

for i=1:256
      pi(i)=p(i)/n;
end

for i=1:256
      Sumpi=sum(pi(1:i));
      if Sumpi>=0.5
          m=i;
          break
      end
end

    numerator = pi(1:m).*log(pi(1:m));
    numerator = nansum(numerator);

    for i=1:256
    denominator = pi(1:i).*log(pi(1:i));
    end

    denominator = nansum(denominator);

    alpha = numerator/denominator;

    if alpha <=0.5
        correctedalpha = 1-alpha;
    else
        correctedalpha = alpha;
    end


    for i=1:256
      Summationpi=sum(pi(1:i));
      if Summationpi>=correctedalpha
          Threshold = i;
          break
      end
    end

    Threshold1 = Threshold;
```

```
      % Portion of Code that Calculates the Porosity and Void Ratio of
the Image

      c2=zeros(sizeofmatrix,sizeofmatrix);
      c2=uint8(c2);
      for i=1:sizeofmatrix
          for j=1:sizeofmatrix
              d=(j - (Row))^2 + (i - (Col))^2;
              if d <= (r)^2
                  if c1(i,j)<Threshold1
                      c2(i,j)=20;
                  else
                      c2(i,j)=100;
                  end

              end
          end
      end

      axes(handles.axes3);
      imshow(c2,[]);

                 for ka = 1: b
                 I = sprintf('CroppedImage_1.%02d.png', ka);
         I=imread(I);
         [Row,Col,r]=MaskPortion(I);
      sizeofmatrix = size(I,1);

      c=[];
      c1=zeros(sizeofmatrix,sizeofmatrix);
      c1=uint8(c1);
      e=0;
      for i=1:sizeofmatrix
          for j=1:sizeofmatrix
              d=(j - (Row))^2 + (i - (Col))^2;
              if d <= (r)^2
                  e=e+1;
                  c(e)=I(i,j);
                  c1(i,j)=I(i,j);
              end
          end
      end

      % Automatic Threshold using Pun's Method

      e=uint8(c);
      p=imhist(e);
      n=sum(p);
      pi=zeros(256,1);

      for i=1:256
            pi(i)=p(i)/n;
      end
```

```
      for i=1:256
            Sumpi=sum(pi(1:i));
            if Sumpi>=0.5
                m=i;
                break
            end
      end

          numerator = pi(1:m).*log(pi(1:m));
          numerator = nansum(numerator);

          for i=1:256
          denominator = pi(1:i).*log(pi(1:i));
          end

          denominator = nansum(denominator);

          alpha = numerator/denominator;

          if alpha <=0.5
              correctedalpha = 1-alpha;
          else
              correctedalpha = alpha;
          end


          for i=1:256
            Summationpi=sum(pi(1:i));
            if Summationpi>=correctedalpha
                Threshold = i;
                break
            end
          end

          Threshold1 = Threshold;

      % Portion of Code that Calculates the Porosity and Void Ratio of
the Image

      c2=zeros(sizeofmatrix,sizeofmatrix);
      c2=uint8(c2);
      for i=1:sizeofmatrix
          for j=1:sizeofmatrix
              d=(j - (Row))^2 + (i - (Col))^2;
              if d <= (r)^2
                  if c1(i,j)<Threshold1
                      c2(i,j)=20;
                  else
                      c2(i,j)=100;
                  end

              end
          end
      end
```

```
      number_Of_Air_Pixels=sum(sum(c2==20));
      number_Of_Solid_Pixels=sum(sum(c2==100));

      number_Of_Air_T=number_Of_Air_T+number_Of_Air_Pixels;
      number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;

      end

      void_ratio = number_Of_Air_T/number_Of_Solid_T;

      v=void_ratio;
      v=num2str(v);
      set(handles.VoidText, 'String', v);
             end

      %%%

       if strcmp(MethodName, 'Kapur, Sahoo, and Wong')
        set(handles.imageSlider,'Value',get(handles.imageSlider,'Min'));
        d = uigetdir('','Please Select the Folder Containing the Cropped
Images');
        cd(d);
        q=dir('*.png');
        b = numel(q);

      %%%

         I=imread('CroppedImage_1.01.png');
         [Row,Col,r]=MaskPortion(I);
         sizeofmatrix = size(I,1);

      c=[];
      c1=zeros(sizeofmatrix,sizeofmatrix);
      c1=uint8(c1);
      e=0;
      for i=1:sizeofmatrix
          for j=1:sizeofmatrix
              d=(j - (Row))^2 + (i - (Col))^2;
              if d <= (r)^2
                  e=e+1;
                  c(e)=I(i,j);
                  c1(i,j)=I(i,j);
              end
          end
      end

      % Automatic Threshold using Kapur, Sahoo, and Wong Method

      e=uint8(c);
      p=imhist(e);
      n=sum(p);
      pi=zeros(256,1);
      for i=1:256
```

```
        pi(i)=p(i)/n;
    end

    pi(1:256,2)=linspace(1,256,256);
    N=0;
    for q=1:256
        if pi(q,1)~=0
            N=N+1;
            pi1(N,1)=pi(q,1);
            pi1(N,2)=pi(q,2);
        end
    end

    pi1= pi1(pi1(:,1)>0.0001,:);

    size(pi1,1);
    c=size(pi1,1);

    sumt = zeros(c,1);
    Ht= zeros(c,1);
    Pt=zeros(c,1);
    Ha=zeros(c,1);
    Hb=zeros(c,1);

    for t = 1:c

          Hi= -sum(pi1(1:c).*log(pi1(1:c)));

         for i = 1:c
             Pt(i)=0;
             Ht(t)=0;
             Pt(i) = sum(pi1(1:i))+Pt(i);
             Ht(t) = -sum(pi1(1:t).*log(pi1(1:t)))+Ht(t);
             Ha(i) = log(Pt(i))+(Ht(i)/Pt(i));
             Hb(i) = log(1-Pt(i))+((Hi-Ht(i))/(1-Pt(i)));
             Hb(i)=real(Hb(i));
         end

         sumt=Ha+Hb;
         sumt(:,2)=pi1(:,2);
        maxrow=max(sumt(:,1));
        Threshold1=max(sumt(sumt(:,1)==maxrow,2));

    end

    % Portion of Code that Calculates the Porosity and Void Ratio of
the Image

    c2=zeros(sizeofmatrix,sizeofmatrix);
    c2=uint8(c2);
    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
            d=(j - (Row))^2 + (i - (Col))^2;
            if d <= (r)^2
```

```
            if c1(i,j)<Threshold1
                c2(i,j)=20;
            else
                c2(i,j)=100;
            end

        end
    end
end

axes(handles.axes3);
imshow(c2,[]);
clear Threshold1;

%%%
        if strcmp(MethodName, 'Kapur, Sahoo, and Wong')
   for ka = 1:b
   I = sprintf('CroppedImage_1.%02d.png', ka);
   I=imread(I);
    [Row,Col,r]=MaskPortion(I);
sizeofmatrix = size(I,1);

c=[];
c1=zeros(sizeofmatrix,sizeofmatrix);
c1=uint8(c1);
e=0;

for i=1:sizeofmatrix
    for j=1:sizeofmatrix
        d=(j - (Row))^2 + (i - (Col))^2;
        if d <= (r)^2
            e=e+1;
            c(e)=I(i,j);
            c1(i,j)=I(i,j);
        end
    end
end

% Automatic Threshold using Kapur, Sahoo, and Wong Method

e=uint8(c);
p=imhist(e);
n=sum(p);
pi=zeros(256,1);
for i=1:256
    pi(i)=p(i)/n;
end

pi(1:256,2)=linspace(1,256,256);
N=0;
for q=1:256
    if pi(q,1)~=0
        N=N+1;
        pi1(N,1)=pi(q,1);
```

```
            pi1(N,2)=pi(q,2);
        end
    end

    pi1= pi1(pi1(:,1)>0.0001,:);

    size(pi1,1);
    c=size(pi1,1);

    sumt = zeros(c,1);
    Ht= zeros(c,1);
    Pt=zeros(c,1);
    Ha=zeros(c,1);
    Hb=zeros(c,1);

    for t = 1:c
          Hi= -sum(pi1(1:c).*log(pi1(1:c)));

         for i = 1:c
             Pt(i)=0;
             Ht(t)=0;
             Pt(i) = sum(pi1(1:i))+Pt(i);
             Ht(t) = -sum(pi1(1:t).*log(pi1(1:t)))+Ht(t);
             Ha(i) = log(Pt(i))+(Ht(i)/Pt(i));
             Hb(i) = log(1-Pt(i))+((Hi-Ht(i))/(1-Pt(i)));
             Hb(i)=real(Hb(i));
         end

         sumt=Ha+Hb;
         sumt(:,2)=pi1(:,2);
         maxrow=max(sumt(:,1));

      Threshold1a=max(sumt(sumt(:,1)==maxrow,2));
        Threshold1(ka,1)=max(sumt(sumt(:,1)==maxrow,2));

    end

       end

    Threshold1=sum(Threshold1(1:b));
    Threshold1=Threshold1/(b);
    Threshold1=round(Threshold1);
    AT1=Threshold1;
    AT1=num2str(AT1);
    set(handles.AT1Box, 'String', AT1);

    % Portion of Code that Calculates the Porosity and Void Ratio of
the Image

    c2=zeros(sizeofmatrix,sizeofmatrix);
    c2=uint8(c2);
    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
            d=(j - (Row))^2 + (i - (Col))^2;
```

```
            if d <= (r)^2
                if c1(i,j)<Threshold1a
                    c2(i,j)=20;
                else
                    c2(i,j)=100;
                end

            end
        end
    end

    number_Of_Air_Pixels=sum(sum(c2==20));
    number_Of_Solid_Pixels=sum(sum(c2==100));

    number_Of_Air_T=number_Of_Air_T+number_Of_Air_Pixels;
    number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;

    end

    void_ratio = number_Of_Air_T/number_Of_Solid_T

    v=void_ratio;
    v=num2str(v);
    set(handles.VoidText, 'String', v);
     end

    if strcmp(MethodName, 'Johannsen and Bille')

set(handles.imageSlider,'Value',get(handles.imageSlider,'Min'));
        d = uigetdir('','Please Select the Folder Containing the
Cropped Images');
        cd(d);
        q=dir('*.png');
        b = numel(q);

        I=imread('CroppedImage_1.01.png');
        [Row,Col,r]=MaskPortion(I);
        sizeofmatrix = size(I,1);

    c=[];
    c1=zeros(sizeofmatrix,sizeofmatrix);
    c1=uint8(c1);
    e=0;
    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
            d=(j - (Row))^2 + (i - (Col))^2;
            if d <= (r)^2
                e=e+1;
                c(e)=I(i,j);
                c1(i,j)=I(i,j);
            end
        end
    end
```

```
% Automatic Threshold using Johannsen and Bille Method

e=uint8(c);
p=imhist(e);
n=sum(p);
pi=zeros(256,1);
for i=1:256
    pi(i)=p(i)/n;
end

pi(1:256,2)=linspace(1,256,256);
N=0;
for q=1:256
    if pi(q,1)~=0
        N=N+1;
        pi1(N,1)=pi(q,1);
        pi1(N,2)=pi(q,2);
    end
end

size(pi1,1);
c=size(pi1,1);

   sumt = zeros(c,1);
   Pn=zeros(c,1);
   Pt=zeros(c,1);
   Pi=zeros(c,1);
   Pr=zeros(c,1);
   Sn=zeros(c,1);
   Sn_bar=zeros(c,1);

for t=1:c
    n=t+1;

    Pn(t)=sum(pi1(1:n));
    Pt(t)=sum(pi1(1:n-1));

    Sn(t)=(log(Pn(t)))-
(1/Pn(t)).*(((pi1(n)).*log(pi1(n)))+(Pt(t).*log(Pt(t))));
    Sn(t)=real(Sn(t));

    Pi(t)=sum(pi1(n:c));
    Pr(t)=sum(pi1(n+1:c));

    Sn_bar(t)=(log(Pi(t)))-
(1/Pi(t)).*(((pi1(n)).*log(pi1(n)))+(Pr(t).*log(Pr(t))));
    Sn_bar(t)=real(Sn_bar(t));

    sumt=Sn+Sn_bar;
    pi1_final=pi1(15:end,:);
    sumt=sumt(15:end);
    sumt(:,2)=pi1_final(:,2);

    minrow=min(sumt(:,1));
```

```
        Threshold1=min(sumt(sumt(:,1)==minrow,2))

    end


    % Portion of Code that Calculates the Porosity and Void Ratio of
the Image

    c2=zeros(sizeofmatrix,sizeofmatrix);
    c2=uint8(c2);
    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
            d=(j - (Row))^2 + (i - (Col))^2;
            if d <= (r)^2
                if c1(i,j)<Threshold1
                    c2(i,j)=20;
                else
                    c2(i,j)=100;
                end

            end
        end
    end

    axes(handles.axes3);
    imshow(c2,[]);

        for ka = 1: b
               I = sprintf('CroppedImage_1.%02d.png', ka);
       I=imread(I);
       [Row,Col,r]=MaskPortion(I);
    sizeofmatrix = size(I,1);

    c=[];
    c1=zeros(sizeofmatrix,sizeofmatrix);
    c1=uint8(c1);
    e=0;
    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
            d=(j - (Row))^2 + (i - (Col))^2;
            if d <= (r)^2
                e=e+1;
                c(e)=I(i,j);
                c1(i,j)=I(i,j);
            end
        end
    end

    % Automatic Threshold using Johannsen and Bille Method

    e=uint8(c);
    p=imhist(e);
    n=sum(p);
    pi=zeros(256,1);
```

```
for i=1:256
    pi(i)=p(i)/n;
end

pi(1:256,2)=linspace(1,256,256);
N=0;
for q=1:256
    if pi(q,1)~=0
        N=N+1;
        pi1(N,1)=pi(q,1);
        pi1(N,2)=pi(q,2);
    end
end

size(pi1,1);
c=size(pi1,1);

   sumt = zeros(c,1);
   Pn=zeros(c,1);
   Pt=zeros(c,1);
   Pi=zeros(c,1);
   Pr=zeros(c,1);
   Sn=zeros(c,1);
   Sn_bar=zeros(c,1);

for t=1:c
    n=t+1;

    Pn(t)=sum(pi1(1:n));
    Pt(t)=sum(pi1(1:n-1));

    Sn(t)=(log(Pn(t)))-
(1/Pn(t)).*(((pi1(n)).*log(pi1(n)))+(Pt(t).*log(Pt(t))));
    Sn(t)=real(Sn(t));

    Pi(t)=sum(pi1(n:c));
    Pr(t)=sum(pi1(n+1:c));

    Sn_bar(t)=(log(Pi(t)))-
(1/Pi(t)).*(((pi1(n)).*log(pi1(n)))+(Pr(t).*log(Pr(t))));
    Sn_bar(t)=real(Sn_bar(t));


    sumt=Sn+Sn_bar;
    pi1_final=pi1(15:end,:);
    sumt=sumt(15:end);
    sumt(:,2)=pi1_final(:,2);
    minrow=min(sumt(:,1));
    Threshold1=min(sumt(sumt(:,1)==minrow,2))

end


% Portion of Code that Calculates the Porosity and Void Ratio of
```

```
the Image

      c2=zeros(sizeofmatrix,sizeofmatrix);
      c2=uint8(c2);
      for i=1:sizeofmatrix
          for j=1:sizeofmatrix
              d=(j - (Row))^2 + (i - (Col))^2;
              if d <= (r)^2
                  if c1(i,j)<Threshold1
                      c2(i,j)=20;
                  else
                      c2(i,j)=100;
                  end

              end
          end
      end

      number_Of_Air_Pixels=sum(sum(c2==20));
      number_Of_Solid_Pixels=sum(sum(c2==100));

      number_Of_Air_T=number_Of_Air_T+number_Of_Air_Pixels;
      number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;

      end

      void_ratio = number_Of_Air_T/number_Of_Solid_T;

      v=void_ratio;
      v=num2str(v);
      set(handles.VoidText, 'String', v);
      end

      %%%

       if strcmp(MethodName, 'Kittler and Illingworth')

set(handles.imageSlider,'Value',get(handles.imageSlider,'Min'));
           d = uigetdir('','Please Select the Folder Containing the
Cropped Images');
           cd(d);
           q=dir('*.png');
           b = numel(q);

         I=imread('CroppedImage_1.01.png');
         [Row,Col,r]=MaskPortion(I);
         sizeofmatrix = size(I,1);

      c=[];
      c1=zeros(sizeofmatrix,sizeofmatrix);
      c1=uint8(c1);
      e=0;
      for i=1:sizeofmatrix
          for j=1:sizeofmatrix
```

```
        d=(j - (Row))^2 + (i - (Col))^2;
        if d <= (r)^2
            e=e+1;
            c(e)=I(i,j);
            c1(i,j)=I(i,j);
        end
    end
end

% Automatic Threshold using Kittler and Illingworth Method

e=uint8(c);
p=imhist(e);
n=sum(p);
pi=zeros(256,1);
for i=1:256
    pi(i)=p(i)/n;
end

pi(1:256,2)=linspace(1,256,256);
N=0;
for q=1:256
    if pi(q,1)~=0
        N=N+1;
        pi1(N,1)=pi(q,1);
        pi1(N,2)=pi(q,2);
    end
end

size(pi1,1);
c=size(pi1,1);


   P1=zeros(c,1);
   P2=zeros(c,1);
   u1=zeros(c,1);
   u2=zeros(c,1);
   s1=zeros(c,1);
   s1_d=zeros(c,1);
   s1_final=zeros(c,1);
   s2_final=zeros(c-2,1);
   J=zeros(c-2,1);

   for i=1:c
       g=1:i;
       g1=i+1:c-1;
       P1(i)=sum(pi1(1:i));
       P2(i)=sum(pi1(i+1:c-1));
       u1(i)=sum(pi1(1:i).*g)/P1(i);
       u2(i)=sum(pi1(i+1:c-1).*g1)/P2(i);
       u2=u2(isfinite(u2));
       u2=u2(u2~=0);
       u2_final=u2';
       s1(i)=0;
```

```
            for ii=1:i
            s1(i)=((ii-u1(i))^2*pi1(ii));
            s1_n=cumsum(s1);
            s1_d(i)=P1(ii);
            s1_dd=s1_d;
            s1_final=s1_n./s1_dd;

            end

                G3=([2:c-1]);
                G4=G3';
                A=u2_final(:,1);

         end

         B=pi1([2:c-1]);
         Pi1_adj=B';
        s2_n=Pi1_adj.*(G4-A).^2;


        for i = 1:c
            for ii = i:c-2
            s2_final(ii)=sum(s2_n(i+1:c-2))/P2(ii);

            end

        end

        s1_final=sqrt(s1_final);
        s2_final=sqrt(s2_final);

       P1=P1([1:c-2]);
       P2=P2([1:c-2]);
       s1_final=s1_final([1:c-2]);
      J=1+2.*((P1.*log(s1_final))+(P2.*log(s2_final)))-
2.*((P1.*log(P1))+(P2.*log(P2)));
      J=J(~isinf(J));

      minrow=max(J);
      Threshold1=max(find(J>=minrow));

      % Portion of Code that Calculates the Porosity and Void Ratio of
the Image

      c2=zeros(sizeofmatrix,sizeofmatrix);
      c2=uint8(c2);
      for i=1:sizeofmatrix
          for j=1:sizeofmatrix
              d=(j - (Row))^2 + (i - (Col))^2;
              if d <= (r)^2
                  if c1(i,j)<Threshold1
                      c2(i,j)=20;
                  else
```

```
                c2(i,j)=100;
            end

        end
    end
end

axes(handles.axes3);
imshow(c2,[]);

     for ka = 1: b
         I = sprintf('CroppedImage_1.%02d.png', ka);
   I=imread(I);
   [Row,Col,r]=MaskPortion(I);
sizeofmatrix = size(I,1);

c=[];
c1=zeros(sizeofmatrix,sizeofmatrix);
c1=uint8(c1);
e=0;
for i=1:sizeofmatrix
    for j=1:sizeofmatrix
        d=(j - (Row))^2 + (i - (Col))^2;
        if d <= (r)^2
            e=e+1;
            c(e)=I(i,j);
            c1(i,j)=I(i,j);
        end
    end
end

% Automatic Threshold using Kittler and Illingworth Method

e=uint8(c);
p=imhist(e);
n=sum(p);
pi=zeros(256,1);
for i=1:256
    pi(i)=p(i)/n;
end

pi(1:256,2)=linspace(1,256,256);
N=0;
for q=1:256
    if pi(q,1)~=0
        N=N+1;
        pi1(N,1)=pi(q,1);
        pi1(N,2)=pi(q,2);
    end
end

size(pi1,1);
c=size(pi1,1);
```

```
 P1=zeros(c,1);
 P2=zeros(c,1);
 u1=zeros(c,1);
 u2=zeros(c,1);
 s1=zeros(c,1);
 s1_d=zeros(c,1);
 s1_final=zeros(c,1);
 s2_final=zeros(c-2,1);
 J=zeros(c-2,1);

 for i=1:c
     g=1:i;
     g1=i+1:c-1;
     P1(i)=sum(pi1(1:i));
     P2(i)=sum(pi1(i+1:c-1));
     u1(i)=sum(pi1(1:i).*g)/P1(i);
     u2(i)=sum(pi1(i+1:c-1).*g1)/P2(i);
     u2=u2(isfinite(u2));
     u2=u2(u2~=0);
     u2_final=u2';
     s1(i)=0;

     for ii=1:i
     s1(i)=((ii-u1(i))^2*pi1(ii));
     s1_n=cumsum(s1);
     s1_d(i)=P1(ii);
     s1_dd=s1_d;
     s1_final=s1_n./s1_dd;

     end

         G3=([2:c-1]);
         G4=G3';
         A=u2_final(:,1);

 end

 B=pi1([2:c-1]);
 Pi1_adj=B';
s2_n=Pi1_adj.*(G4-A).^2;


for i = 1:c
    for ii = i:c-2
    s2_final(ii)=sum(s2_n(i+1:c-2))/P2(ii);

    end

end

s1_final=sqrt(s1_final);
s2_final=sqrt(s2_final);

P1=P1([1:c-2]);
```

```
       P2=P2([1:c-2]);
       s1_final=s1_final([1:c-2]);
      J=1+2.*((P1.*log(s1_final))+(P2.*log(s2_final)))-
2.*((P1.*log(P1))+(P2.*log(P2)));
      J=J(~isinf(J));

      minrow=max(J);
      Threshold1=max(find(J>=minrow));
      Threshold1a(ka,1)=max(find(J>=minrow));

      % Portion of Code that Calculates the Porosity and Void Ratio of
the Image

      c2=zeros(sizeofmatrix,sizeofmatrix);
      c2=uint8(c2);
      for i=1:sizeofmatrix
          for j=1:sizeofmatrix
              d=(j - (Row))^2 + (i - (Col))^2;
              if d <= (r)^2
                  if c1(i,j)<Threshold1
                      c2(i,j)=20;
                  else
                      c2(i,j)=100;
                  end

              end
          end
      end

      number_Of_Air_Pixels=sum(sum(c2==20));
      number_Of_Solid_Pixels=sum(sum(c2==100));

      number_Of_Air_T=number_Of_Air_T+number_Of_Air_Pixels;
      number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;

      end

      void_ratio = number_Of_Air_T/number_Of_Solid_T;

      v=void_ratio;
      v=num2str(v);
      set(handles.VoidText, 'String', v);

      Threshold1a=sum(Threshold1a(1:b));
      Threshold1a=Threshold1a/(b);
      Threshold1a=round(Threshold1a);
      AT1=Threshold1a;
      AT1=num2str(AT1);
      set(handles.AT1Box, 'String', AT1);
         end

      %%%

      number_Of_Air_T=0;
```

```
number_Of_Solid_T=0;

 contents = get(handles.Methods,'String');
 MethodName = contents{get(handles.Methods,'Value')};
 getappdata(handles.axes1,'fileName');

      if strcmp(MethodName, 'Otsu ')
          for ka = 1: b
              I = sprintf('CroppedImage_1.%02d.png', ka);
              I=imread(I);
              [Row,Col,r]=MaskPortion(I);
              sizeofmatrix = size(I,1);
              c=[];
              c1=zeros(sizeofmatrix,sizeofmatrix);
              c1=uint8(c1);
              e=0;
              for i=1:sizeofmatrix
                  for j=1:sizeofmatrix
                      d=(j - (Row))^2 + (i - (Col))^2;
                      if d <= (r)^2
                          e=e+1;
                          c(e)=I(i,j);
                          c1(i,j)=I(i,j);
                      end
                  end
              end

% Automatic Threshold using Otsu's Method

e=uint8(c);
p=imhist(e);
mean=0;
for i=1:256
    mean=mean+(i*p(i,1));
end
mean=mean/256;
sumt=zeros(256,1);
for i=1:256
    meanli=0;
    for j=1:i-1
        meanli=meanli+(j*p(j,1));
    end
    meanli=meanli/256;
    meangi=0;
    for j=i:256
        meangi=meangi+(j*p(j,1));
    end
    meangi=meangi/256;
    sumli=0;
    sumgi=0;
    for ii=1:i-1
        sumli=sumli+p(ii,1)*((meanli-mean)^2);
    end
```

```
    for ii=i:256
        sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
    end

    sumt(i,1)=sumgi+sumli;

end

Threshold1(ka,1)=find(sumt==max(sumt));
end
Threshold1=sum(Threshold1(1:b));
Threshold1=Threshold1/(b);
Threshold1=round(Threshold1);
AT1=Threshold1;
AT1=num2str(AT1);
set(handles.AT1Box, 'String', AT1);
      end
%%%

 if strcmp(MethodName, 'Pun ')
     for ka = 1: b
         I = sprintf('CroppedImage_1.%02d.png', ka);
         I=imread(I);
         [Row,Col,r]=MaskPortion(I);
         sizeofmatrix = size(I,1);
         c=[];
         c1=zeros(sizeofmatrix,sizeofmatrix);
         c1=uint8(c1);
         e=0;
         for i=1:sizeofmatrix
             for j=1:sizeofmatrix
                 d=(j - (Row))^2 + (i - (Col))^2;
                 if d <= (r)^2
                     e=e+1;
                     c(e)=I(i,j);
                     c1(i,j)=I(i,j);
                 end
             end
         end

% Pun
e=uint8(c);
p=imhist(e);
n=sum(p);
pi=zeros(256,1);

for i=1:256
      pi(i)=p(i)/n;
end

for i=1:256
      Sumpi=sum(pi(1:i));
      if Sumpi>=0.5
          m=i;
```

```
          break
      end
end

    numerator = pi(1:m).*log(pi(1:m));
    numerator = nansum(numerator);

    for i=1:256
    denominator = pi(1:i).*log(pi(1:i));
    end

    denominator = nansum(denominator);

    alpha = numerator/denominator;

    if alpha <=0.5
        correctedalpha = 1-alpha;
    else
        correctedalpha = alpha;
    end

    for i=1:256
      Summationpi=sum(pi(1:i));
      if Summationpi>=correctedalpha
          Threshold = i;
          break
      end
    end

    Threshold1(ka,1) = Threshold;
end

Threshold1=sum(Threshold1(1:b));
Threshold1=Threshold1/(b);
Threshold1=round(Threshold1);
AT1=Threshold1;
AT1=num2str(AT1);
set(handles.AT1Box, 'String', AT1);
 end

 %%%

  if strcmp(MethodName, 'Johannsen and Bille')
      for ka = 1: b
          I = sprintf('CroppedImage_1.%02d.png', ka);
          I=imread(I);
          [Row,Col,r]=MaskPortion(I);
          sizeofmatrix = size(I,1);
          c=[];
          c1=zeros(sizeofmatrix,sizeofmatrix);
          c1=uint8(c1);
          e=0;
          for i=1:sizeofmatrix
              for j=1:sizeofmatrix
```

```
                    d=(j - (Row))^2 + (i - (Col))^2;
                    if d <= (r)^2
                        e=e+1;
                        c(e)=I(i,j);
                        c1(i,j)=I(i,j);
                    end
                end
            end

% Automatic Threshold using Johannsen and Bille Method

e=uint8(c);
p=imhist(e);
n=sum(p);
pi=zeros(256,1);
for i=1:256
    pi(i)=p(i)/n;
end

pi(1:256,2)=linspace(1,256,256);
N=0;
for q=1:256
    if pi(q,1)~=0
        N=N+1;
        pi1(N,1)=pi(q,1);
        pi1(N,2)=pi(q,2);
    end
end

size(pi1,1);
c=size(pi1,1);

   sumt = zeros(c,1);
   Pn=zeros(c,1);
   Pt=zeros(c,1);
   Pi=zeros(c,1);
   Pr=zeros(c,1);
   Sn=zeros(c,1);
   Sn_bar=zeros(c,1);

for t=1:c
    n=t+1;

    Pn(t)=sum(pi1(1:n));
    Pt(t)=sum(pi1(1:n-1));

    Sn(t)=(log(Pn(t)))-
(1/Pn(t)).*(((pi1(n)).*log(pi1(n)))+(Pt(t).*log(Pt(t))));
    Sn(t)=real(Sn(t));

    Pi(t)=sum(pi1(n:c));
    Pr(t)=sum(pi1(n+1:c));

    Sn_bar(t)=(log(Pi(t)))-
```

```
(1/Pi(t)).*(((pi1(n)).*log(pi1(n)))+(Pr(t).*log(Pr(t))));
          Sn_bar(t)=real(Sn_bar(t));

          sumt=Sn+Sn_bar;
          pi1_final=pi1(15:end,:);
          sumt=sumt(15:end);
          sumt(:,2)=pi1_final(:,2);
          minrow=min(sumt(:,1));
          Threshold1(ka,1)=min(sumt(sumt(:,1)==minrow,2));

      end

      end
      Threshold1=sum(Threshold1(1:b));
      Threshold1=Threshold1/(b);
      Threshold1=round(Threshold1);
      AT1=Threshold1;
      AT1=num2str(AT1);
      set(handles.AT1Box, 'String', AT1);
        end

        function VoidText_Callback(hObject, eventdata, handles)
      % hObject    handle to VoidText (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)

      % Hints: get(hObject,'String') returns contents of VoidText as
text
      %        str2double(get(hObject,'String')) returns contents of
VoidText as a double


      % --- Executes during object creation, after setting all
properties.
      function VoidText_CreateFcn(hObject, eventdata, handles)
      % hObject    handle to VoidText (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    empty - handles not created until after all
CreateFcns called

      % Hint: edit controls usually have a white background on Windows.
      %       See ISPC and COMPUTER.
      if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
          set(hObject,'BackgroundColor','white');
      end

      function AT1Box_Callback(hObject, eventdata, handles)
      % hObject    handle to AT1Box (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)
```

```
      % Hints: get(hObject,'String') returns contents of AT1Box as text
      %        str2double(get(hObject,'String')) returns contents of
AT1Box as a double

      % --- Executes during object creation, after setting all
properties.
      function AT1Box_CreateFcn(hObject, eventdata, handles)
      % hObject    handle to AT1Box (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    empty - handles not created until after all
CreateFcns called

      % Hint: edit controls usually have a white background on Windows.
      %       See ISPC and COMPUTER.
      if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
          set(hObject,'BackgroundColor','white');
      end
      % --- Executes on button press in SingleThreshold.
      function SingleThreshold_Callback(hObject, eventdata, handles)
      number_Of_Air_T=0;
      number_Of_Solid_T=0;
      contents = get(handles.Methods,'String');
      MethodName = contents{get(handles.Methods,'Value')};
      getappdata(handles.axes1,'fileName');

            if strcmp(MethodName, 'Otsu ')
                A = uigetfile({'*.jpg;*.tif;*.bmp;*.png;*.gif','All
Image Files'},'Please Select A Cropped Image from Folder that
Appears');
                I=imread(A);
                set(handles.Name, 'String', A);
                [Row,Col,r]=MaskPortion(I);
                sizeofmatrix = size(I,1);
                c=[];
                c1=zeros(sizeofmatrix,sizeofmatrix);
                c1=uint8(c1);
                e=0;
                for i=1:sizeofmatrix
                    for j=1:sizeofmatrix
                        d=(j - (Row))^2 + (i - (Col))^2;
                        if d <= (r)^2
                            e=e+1
                            c(e)=I(i,j);
                            c1(i,j)=I(i,j);
                        end
                    end
                end

      % Automatic Threshold using Otsu's Method

      e=uint8(c);
```

```
      p=imhist(e);
      mean=0;
      for i=1:256
          mean=mean+(i*p(i,1));
      end
      mean=mean/256;
      sumt=zeros(256,1);
      for i=1:256
          meanli=0;
          for j=1:i-1
              meanli=meanli+(j*p(j,1));
          end
          meanli=meanli/256;
          meangi=0;
          for j=i:256
              meangi=meangi+(j*p(j,1));
          end
          meangi=meangi/256;
          sumli=0;
          sumgi=0;
          for ii=1:i-1
              sumli=sumli+p(ii,1)*((meanli-mean)^2);
          end

          for ii=i:256
              sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
          end

          sumt(i,1)=sumgi+sumli;

      end

      Threshold1=find(sumt==max(sumt));


      T1=Threshold1;
      T1=num2str(T1);
      set(handles.T1Box, 'String', T1);
            end

      %%%

      if strcmp(MethodName, 'Pun ')

              A = uigetfile({'*.jpg;*.tif;*.bmp;*.png;*.gif','All Image
Files'},'Please Select An Image');
              I=imread(A);
              set(handles.Name, 'String', A);
              [Row,Col,r]=MaskPortion(I);
              sizeofmatrix = size(I,1);
              c=[];
              c1=zeros(sizeofmatrix,sizeofmatrix);
              c1=uint8(c1);
              e=0;
```

```
        for i=1:sizeofmatrix
            for j=1:sizeofmatrix
                d=(j - (Row))^2 + (i - (Col))^2;
                if d <= (r)^2
                    e=e+1;
                    c(e)=I(i,j);
                    c1(i,j)=I(i,j);
                end
            end
        end

% Pun

e=uint8(c);
p=imhist(e);
n=sum(p);
pi=zeros(256,1);

for i=1:256
      pi(i)=p(i)/n;
end

for i=1:256
      Sumpi=sum(pi(1:i));
      if Sumpi>=0.5
          m=i;
          break
      end
end

    numerator = pi(1:m).*log(pi(1:m));
    numerator = nansum(numerator);

    for i=1:256
    denominator = pi(1:i).*log(pi(1:i));
    end

    denominator = nansum(denominator);

    alpha = numerator/denominator;

    if alpha <=0.5
        correctedalpha = 1-alpha;
    else
        correctedalpha = alpha;
    end


    for i=1:256
      Summationpi=sum(pi(1:i));
      if Summationpi>=correctedalpha
          Threshold = i;
          break
      end
```

```
        end

    Threshold1 = Threshold;
    T1=Threshold1;
    T1=num2str(T1);
    set(handles.T1Box, 'String', T1);
    end

    %%%

    if strcmp(MethodName, 'Kapur, Sahoo, and Wong')
        A = uigetfile({'*.jpg;*.tif;*.bmp;*.png;*.gif','All Image
Files'},'Please Select An Image');
        I=imread(A);
        set(handles.Name, 'String', A);
        [Row,Col,r]=MaskPortion(I);
        sizeofmatrix = size(I,1);
        c=[];
        c1=zeros(sizeofmatrix,sizeofmatrix);
        c1=uint8(c1);
        e=0;
        for i=1:sizeofmatrix
            for j=1:sizeofmatrix
                d=(j - (Row))^2 + (i - (Col))^2;
                if d <= (r)^2
                    e=e+1;
                    c(e)=I(i,j);
                    c1(i,j)=I(i,j);
                end
            end
        end

    % Automatic Threshold using Kapur, Sahoo, and Wong Method

    e=uint8(c);
    p=imhist(e);
    n=sum(p);
    pi=zeros(256,1);
    for i=1:256
        pi(i)=p(i)/n;
    end

    pi(1:256,2)=linspace(1,256,256);
    N=0;
    for q=1:256
        if pi(q,1)~=0
            N=N+1;
            pi1(N,1)=pi(q,1);
            pi1(N,2)=pi(q,2);
        end
    end

    pi1= pi1(pi1(:,1)>0.0001,:);
```

```
size(pi1,1);
c=size(pi1,1);

sumt = zeros(c,1);
Ht= zeros(c,1);
Pt=zeros(c,1);
Ha=zeros(c,1);
Hb=zeros(c,1);

for t = 1:c

      Hi= -sum(pi1(1:c).*log(pi1(1:c)));

     for i = 1:c
         Pt(i)=0;
         Ht(t)=0;
         Pt(i) = sum(pi1(1:i))+Pt(i);
         Ht(t) = -sum(pi1(1:t).*log(pi1(1:t)))+Ht(t);
         Ha(i) = log(Pt(i))+(Ht(i)/Pt(i));
         Hb(i) = log(1-Pt(i))+((Hi-Ht(i))/(1-Pt(i)));
         Hb(i)=real(Hb(i));
     end

     sumt=Ha+Hb;
     sumt(:,2)=pi1(:,2);
    maxrow=max(sumt(:,1));
    Threshold1=max(sumt(sumt(:,1)==maxrow,2))

end
T1=Threshold1;
T1=num2str(T1);
set(handles.T1Box, 'String', T1);
end

%%%

if strcmp(MethodName, 'Johannsen and Bille')
    A = uigetfile({'*.jpg;*.tif;*.bmp;*.png;*.gif','All Image
Files'},'Please Select An Image');
    I=imread(A);
    set(handles.Name, 'String', A);
    [Row,Col,r]=MaskPortion(I);
    sizeofmatrix = size(I,1);
    c=[];
    c1=zeros(sizeofmatrix,sizeofmatrix);
    c1=uint8(c1);
    e=0;
    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
            d=(j - (Row))^2 + (i - (Col))^2;
            if d <= (r)^2
                e=e+1;
                c(e)=I(i,j);
                c1(i,j)=I(i,j);
```

```
            end
        end
    end

% Automatic Threshold using Johannsen and Bille Method

e=uint8(c);
p=imhist(e);
n=sum(p);
pi=zeros(256,1);
for i=1:256
    pi(i)=p(i)/n;
end

pi(1:256,2)=linspace(1,256,256);
N=0;
for q=1:256
    if pi(q,1)~=0
        N=N+1;
        pi1(N,1)=pi(q,1);
        pi1(N,2)=pi(q,2);
    end
end

size(pi1,1);
c=size(pi1,1);

   sumt = zeros(c,1);
   Pn=zeros(c,1);
   Pt=zeros(c,1);
   Pi=zeros(c,1);
   Pr=zeros(c,1);
   Sn=zeros(c,1);
   Sn_bar=zeros(c,1);

for t=1:c
    n=t+1;

    Pn(t)=sum(pi1(1:n));
    Pt(t)=sum(pi1(1:n-1));

    Sn(t)=(log(Pn(t)))-
(1/Pn(t)).*(((pi1(n)).*log(pi1(n)))+(Pt(t).*log(Pt(t))));
    Sn(t)=real(Sn(t));

    Pi(t)=sum(pi1(n:c));
    Pr(t)=sum(pi1(n+1:c));

    Sn_bar(t)=(log(Pi(t)))-
(1/Pi(t)).*(((pi1(n)).*log(pi1(n)))+(Pr(t).*log(Pr(t))));
    Sn_bar(t)=real(Sn_bar(t));

    sumt=Sn+Sn_bar;
    pi1_final=pi1(15:end,:);
```

```
        sumt=sumt(15:end);
        sumt(:,2)=pi1_final(:,2);
        minrow=min(sumt(:,1));
        Threshold1=min(sumt(sumt(:,1)==minrow,2));

    end
    T1=Threshold1;
    T1=num2str(T1);
    set(handles.T1Box, 'String', T1);
    end

    %%%

     if strcmp(MethodName, 'Kittler and Illingworth')
         A = uigetfile({'*.jpg;*.tif;*.bmp;*.png;*.gif','All Image
Files'},'Please Select An Image');
         I=imread(A);
         set(handles.Name, 'String', A);
         [Row,Col,r]=MaskPortion(I);
         sizeofmatrix = size(I,1);
         c=[];
         c1=zeros(sizeofmatrix,sizeofmatrix);
         c1=uint8(c1);
         e=0;
         for i=1:sizeofmatrix
             for j=1:sizeofmatrix
                 d=(j - (Row))^2 + (i - (Col))^2;
                 if d <= (r)^2
                     e=e+1;
                     c(e)=I(i,j);
                     c1(i,j)=I(i,j);
                 end
             end
         end

    % Automatic Threshold using Kittler and Illingworth Method

    e=uint8(c);
    p=imhist(e);
    n=sum(p);
    pi=zeros(256,1);
    for i=1:256
        pi(i)=p(i)/n;
    end

    pi(1:256,2)=linspace(1,256,256);
    N=0;
    for q=1:256
        if pi(q,1)~=0
            N=N+1;
            pi1(N,1)=pi(q,1);
            pi1(N,2)=pi(q,2);
        end
    end
```

```
size(pi1,1);
c=size(pi1,1);

   P1=zeros(c,1);
   P2=zeros(c,1);
   u1=zeros(c,1);
   u2=zeros(c,1);
   s1=zeros(c,1);
   s1_d=zeros(c,1);
   s1_final=zeros(c,1);
   s2_final=zeros(c-2,1);
   J=zeros(c-2,1);

   for i=1:c
       g=1:i;
       g1=i+1:c-1;
       P1(i)=sum(pi1(1:i));
       P2(i)=sum(pi1(i+1:c-1));
       u1(i)=sum(pi1(1:i).*g)/P1(i);
       u2(i)=sum(pi1(i+1:c-1).*g1)/P2(i);
       u2=u2(isfinite(u2));
       u2=u2(u2~=0);
       u2_final=u2';
       s1(i)=0;

       for ii=1:i
       s1(i)=((ii-u1(i))^2*pi1(ii));
       s1_n=cumsum(s1);
       s1_d(i)=P1(ii);
       s1_dd=s1_d;
       s1_final=s1_n./s1_dd;

       end

           G3=([2:c-1]);
           G4=G3';
           A=u2_final(:,1);

   end

   B=pi1([2:c-1]);
   Pi1_adj=B';
  s2_n=Pi1_adj.*(G4-A).^2;


  for i = 1:c
      for ii = i:c-2
      s2_final(ii)=sum(s2_n(i+1:c-2))/P2(ii);

      end

  end
```

```
        s1_final=sqrt(s1_final);
        s2_final=sqrt(s2_final);

       P1=P1([1:c-2]);
       P2=P2([1:c-2]);
       s1_final=s1_final([1:c-2]);
      J=1+2.*((P1.*log(s1_final))+(P2.*log(s2_final)))-
2.*((P1.*log(P1))+(P2.*log(P2)));
      J=J(~isinf(J));

      minrow=max(J);
      Threshold1=max(find(J>=minrow));

      T1=Threshold1;
      T1=num2str(T1);
      set(handles.T1Box, 'String', T1);
       end

       function T1Box_Callback(hObject, eventdata, handles)
      % hObject    handle to T1Box (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)

      % Hints: get(hObject,'String') returns contents of T1Box as text
      %        str2double(get(hObject,'String')) returns contents of
T1Box as a double

      % --- Executes during object creation, after setting all
properties.
      function T1Box_CreateFcn(hObject, eventdata, handles)
      % hObject    handle to T1Box (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    empty - handles not created until after all
CreateFcns called

      % Hint: edit controls usually have a white background on Windows.
      %       See ISPC and COMPUTER.
      if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
          set(hObject,'BackgroundColor','white');
      end

      function Name_Callback(hObject, eventdata, handles)
      % hObject    handle to Name (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)

      % Hints: get(hObject,'String') returns contents of Name as text
      %        str2double(get(hObject,'String')) returns contents of
Name as a double
```

```
      % --- Executes during object creation, after setting all
properties.
      function Name_CreateFcn(hObject, eventdata, handles)
      % hObject    handle to Name (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    empty - handles not created until after all
CreateFcns called

      % Hint: edit controls usually have a white background on Windows.
      %       See ISPC and COMPUTER.
      if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
          set(hObject,'BackgroundColor','white');
      end

      % --- Executes during object creation, after setting all
properties.
      function Excel_CreateFcn(hObject, eventdata, handles)
      % hObject    handle to Excel (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    empty - handles not created until after all
CreateFcns called

      % --- Executes on button press in Excel.
      function Excel_Callback(hObject, eventdata, handles)
      str1='Segmentation Method';
      O='Otsu';
      P= 'Pun';
      KSW= 'Kapur, Sahoo, and Wong';
      J = 'Johannsen and Bille';
      K='Kittler and Illingworth';
      str2='Void Ratio';
      str3='Average Threshold Value for all Images';

      d = uigetdir('*.png');
      cd(d);
      q=dir('*.png');
      b = numel(q);

      k = warndlg('Exporting of data can take a couple of minutes
depending on the number of images being analyzed. Please do not click
anything until data exporting is completed.','Warning');
      waitfor(k);

      %%% OTSU

      number_Of_Air_T=0;
      number_Of_Solid_T=0;

      for ka = 1: b
         I = sprintf('CroppedImage_1.%02d.png', ka);
         I=imread(I);
```

```
   [Row,Col,r]=MaskPortion(I);
   sizeofmatrix = size(I,1);
   c=[];
   c1=zeros(sizeofmatrix,sizeofmatrix);
   c1=uint8(c1);
   e=0;
   for i=1:sizeofmatrix
       for j=1:sizeofmatrix
           d=(j - (Row))^2 + (i - (Col))^2;
           if d <= (r)^2
               e=e+1;
               c(e)=I(i,j);
               c1(i,j)=I(i,j);
           end
       end
   end

% Automatic Threshold using Otsu's Method

e=uint8(c);
p=imhist(e);
mean=0;

for i=1:256
    mean=mean+(i*p(i,1));
end
mean=mean/256;
sumt=zeros(256,1);
for i=1:256
    meanli=0;
    for j=1:i-1
        meanli=meanli+(j*p(j,1));
    end
    meanli=meanli/256;
    meangi=0;
    for j=i:256
        meangi=meangi+(j*p(j,1));
    end
    meangi=meangi/256;
    sumli=0;
    sumgi=0;
    for ii=1:i-1
        sumli=sumli+p(ii,1)*((meanli-mean)^2);
    end
    for ii=i:256
        sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
    end
    sumt(i,1)=sumgi+sumli;
end

Threshold1=find(sumt==max(sumt));
Threshold1a(ka,1)=find(sumt==max(sumt));

% Portion of Code that Calculates the Porosity and Void Ratio of
```

```
the Image

      c2=zeros(sizeofmatrix,sizeofmatrix);
      c2=uint8(c2);
      for i=1:sizeofmatrix
          for j=1:sizeofmatrix
              d=(j - (Row))^2 + (i - (Col))^2;
              if d <= (r)^2
                  if c1(i,j)<Threshold1
                      c2(i,j)=20;
                  else
                      c2(i,j)=100;
                  end

              end
          end
      end

      number_Of_Air_Pixels=sum(sum(c2==20));
      number_Of_Solid_Pixels=sum(sum(c2==100));

      number_Of_Air_T=number_Of_Air_T+number_Of_Air_Pixels;
      number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;

      end

      Threshold1a=sum(Threshold1a(1:b));
      Threshold1a=Threshold1a/(b);
      Threshold1a=round(Threshold1a);
      AT1=Threshold1a;
      AT1=num2str(AT1);
      str4 = AT1;
      void_ratio = number_Of_Air_T/number_Of_Solid_T;

      v=void_ratio;
      v=num2str(v);
      str9=v;

      AH=cell(10,1);
      AH{1,1}='Otsu segmentation method complete.';
      AH{2,1}='Four methods remain...';
      set(handles.listbox3, 'String', AH);
      drawnow()

      %%% PUN

      number_Of_Air_T=0;
      number_Of_Solid_T=0;

         for ka = 1: b
         I = sprintf('CroppedImage_1.%02d.png', ka);
         I=imread(I);
         [Row,Col,r]=MaskPortion(I);
         sizeofmatrix = size(I,1);
```

```
   c=[];
   c1=zeros(sizeofmatrix,sizeofmatrix);
   c1=uint8(c1);
   e=0;
   for i=1:sizeofmatrix
    for j=1:sizeofmatrix
        d=(j - (Row))^2 + (i - (Col))^2;
        if d <= (r)^2
            e=e+1;
            c(e)=I(i,j);
            c1(i,j)=I(i,j);
        end
    end
end

e=uint8(c);
p=imhist(e);
n=sum(p);
pi=zeros(256,1);

for i=1:256
      pi(i)=p(i)/n;
end

for i=1:256
      Sumpi=sum(pi(1:i));
      if Sumpi>=0.5
          m=i;
          break
      end
end
    numerator = pi(1:m).*log(pi(1:m));
    numerator = nansum(numerator);
    for i=1:256
    denominator = pi(1:i).*log(pi(1:i));
    end
    denominator = nansum(denominator);
    alpha = numerator/denominator;
    if alpha <=0.5
        correctedalpha = 1-alpha;
    else
        correctedalpha = alpha;
    end
    for i=1:256
      Summationpi=sum(pi(1:i));
      if Summationpi>=correctedalpha
          Threshold = i;
          break
      end
    end

    Threshold1=Threshold;
    Threshold1a(ka,1) = Threshold;
```

```
      % Portion of Code that Calculates the Porosity and Void Ratio of
the Image

      c2=zeros(sizeofmatrix,sizeofmatrix);
      c2=uint8(c2);
      for i=1:sizeofmatrix
          for j=1:sizeofmatrix
              d=(j - (Row))^2 + (i - (Col))^2;
              if d <= (r)^2
                  if c1(i,j)<Threshold1
                      c2(i,j)=20;
                  else
                      c2(i,j)=100;
                  end

              end
          end
      end

      number_Of_Air_Pixels=sum(sum(c2==20));
      number_Of_Solid_Pixels=sum(sum(c2==100));

      number_Of_Air_T=number_Of_Air_T+number_Of_Air_Pixels;
      number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;

      end

      Threshold1a=sum(Threshold1a(1:b));
      Threshold1a=Threshold1a/(b);
      Threshold1a=round(Threshold1a);
      AT1=Threshold1a;
      AT1=num2str(AT1);
      str5=AT1;
      void_ratio = number_Of_Air_T/number_Of_Solid_T;
      v=void_ratio;
      v=num2str(v);
      str10=v;

      AH{3,1}='Pun segmentation method complete.';
      AH{4,1}='Three methods remain...';
      set(handles.listbox3,'String',AH);
      drawnow()

      %%% KAPUR, SAHOO, AND WONG

      number_Of_Air_T=0;
      number_Of_Solid_T=0;

      for ka = 1:b
          I = sprintf('CroppedImage_1.%02d.png', ka);
          I=imread(I);
          [Row,Col,r]=MaskPortion(I);
          sizeofmatrix = size(I,1);
          c=[];
```

```
    c1=zeros(sizeofmatrix,sizeofmatrix);
    c1=uint8(c1);
    e=0;
    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
            d=(j - (Row))^2 + (i - (Col))^2;
            if d <= (r)^2
                e=e+1;
                c(e)=I(i,j);
                c1(i,j)=I(i,j);
            end
        end
    end

% Automatic Threshold using Kapur, Sahoo, and Wong Method

e=uint8(c);
p=imhist(e);
n=sum(p);
pi=zeros(256,1);
for i=1:256
    pi(i)=p(i)/n;
end

pi(1:256,2)=linspace(1,256,256);
N=0;
for q=1:256
    if pi(q,1)~=0
        N=N+1;
        pi1(N,1)=pi(q,1);
        pi1(N,2)=pi(q,2);
    end
end

pi1= pi1(pi1(:,1)>0.0001,:);
size(pi1,1);
c=size(pi1,1);

sumt = zeros(c,1);
Ht= zeros(c,1);
Pt=zeros(c,1);
Ha=zeros(c,1);
Hb=zeros(c,1);

for t = 1:c
      Hi= -sum(pi1(1:c).*log(pi1(1:c)));
     for i = 1:c
         Pt(i)=0;
         Ht(t)=0;
         Pt(i) = sum(pi1(1:i))+Pt(i);
         Ht(t) = -sum(pi1(1:t).*log(pi1(1:t)))+Ht(t);
         Ha(i) = log(Pt(i))+(Ht(i)/Pt(i));
         Hb(i) = log(1-Pt(i))+((Hi-Ht(i))/(1-Pt(i)));
         Hb(i)=real(Hb(i));
```

```
            end

            sumt=Ha+Hb;

            sumt(:,2)=pi1(:,2);
            maxrow=max(sumt(:,1));

            Threshold1a=max(sumt(sumt(:,1)==maxrow,2));
            Threshold1(ka,1)=max(sumt(sumt(:,1)==maxrow,2));
        end

        end

        Threshold1=sum(Threshold1(1:b));
        Threshold1=Threshold1/(b);
        Threshold1=round(Threshold1);
        AT1=Threshold1;
        AT1=num2str(AT1);

        str6=AT1;

        % Portion of Code that Calculates the Porosity and Void Ratio of
    the Image

        c2=zeros(sizeofmatrix,sizeofmatrix);
        c2=uint8(c2);
        for i=1:sizeofmatrix
            for j=1:sizeofmatrix
                d=(j - (Row))^2 + (i - (Col))^2;
                if d <= (r)^2
                    if c1(i,j)<Threshold1a
                        c2(i,j)=20;
                    else
                        c2(i,j)=100;
                    end

                end
            end
        end

        number_Of_Air_Pixels=sum(sum(c2==20));
        number_Of_Solid_Pixels=sum(sum(c2==100));

        number_Of_Air_T=number_Of_Air_T+number_Of_Air_Pixels;
        number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;

        void_ratio = number_Of_Air_T/number_Of_Solid_T;

        v=void_ratio
        v=num2str(v);
        str11=v;
        AH{5,1}='Kapur, Sahoo, and Wong segmentation method complete.';
        AH{6,1}='Two methods remain...';
        set(handles.listbox3,'String',AH);
```

```
drawnow()

% Automatic Threshold using Johannsen and Bille Method

number_Of_Air_T=0;
number_Of_Solid_T=0;

for ka = 1: b
   I = sprintf('CroppedImage_1.%02d.png', ka);
   I=imread(I);
   [Row,Col,r]=MaskPortion(I);
   sizeofmatrix = size(I,1);
   c=[];
   c1=zeros(sizeofmatrix,sizeofmatrix);
   c1=uint8(c1);
   e=0;
   for i=1:sizeofmatrix
       for j=1:sizeofmatrix
           d=(j - (Row))^2 + (i - (Col))^2;
           if d <= (r)^2
               e=e+1;
               c(e)=I(i,j);
               c1(i,j)=I(i,j);
           end
       end
   end

% Automatic Threshold using Johannsen and Bille Method

e=uint8(c);
p=imhist(e);
n=sum(p);
pi=zeros(256,1);
for i=1:256
    pi(i)=p(i)/n;
end

pi(1:256,2)=linspace(1,256,256);
N=0;
for q=1:256
    if pi(q,1)~=0
        N=N+1;
        pi1(N,1)=pi(q,1);
        pi1(N,2)=pi(q,2);
    end
end

size(pi1,1);
c=size(pi1,1);

sumt = zeros(c,1);
Pn=zeros(c,1);
Pt=zeros(c,1);
Pi=zeros(c,1);
```

```
      Pr=zeros(c,1);
      Sn=zeros(c,1);
      Sn_bar=zeros(c,1);

      for t=1:c
          n=t+1;

          Pn(t)=sum(pi1(1:n));
          Pt(t)=sum(pi1(1:n-1));

          Sn(t)=(log(Pn(t)))-
(1/Pn(t)).*(((pi1(n)).*log(pi1(n)))+(Pt(t).*log(Pt(t))));
          Sn(t)=real(Sn(t));

          Pi(t)=sum(pi1(n:c));
          Pr(t)=sum(pi1(n+1:c));

          Sn_bar(t)=(log(Pi(t)))-
(1/Pi(t)).*(((pi1(n)).*log(pi1(n)))+(Pr(t).*log(Pr(t))));
          Sn_bar(t)=real(Sn_bar(t));

          sumt=Sn+Sn_bar;
          pi1_final=pi1(15:end,:);
          sumt=sumt(15:end);
          sumt(:,2)=pi1_final(:,2);
          minrow=min(sumt(:,1));
          Threshold1a(ka,1)=min(sumt(sumt(:,1)==minrow,2));
          Threshold1=min(sumt(sumt(:,1)==minrow,2));
          end

      % Portion of Code that Calculates the Porosity and Void Ratio of
the Image

      c2=zeros(sizeofmatrix,sizeofmatrix);
      c2=uint8(c2);
      for i=1:sizeofmatrix
          for j=1:sizeofmatrix
              d=(j - (Row))^2 + (i - (Col))^2;
              if d <= (r)^2
                  if c1(i,j)<Threshold1
                      c2(i,j)=20;
                  else
                      c2(i,j)=100;
                  end

              end
          end
      end

      number_Of_Air_Pixels=sum(sum(c2==20));
      number_Of_Solid_Pixels=sum(sum(c2==100));

      number_Of_Air_T=number_Of_Air_T+number_Of_Air_Pixels;
      number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;
```

```
end

Threshold1a=sum(Threshold1a(1:b));
Threshold1a=Threshold1a/(b);
Threshold1a=round(Threshold1a);
AT1=Threshold1a;
AT1=num2str(AT1);
str7=AT1;

void_ratio = number_Of_Air_T/number_Of_Solid_T;
v=void_ratio;
v=num2str(v);
str12=v;

AH{7,1}='Johannsen and Bille segmentation method complete.';
AH{8,1}='One method remains...';
set(handles.listbox3,'String',AH);
drawnow()

% Automatic Threshold using Kittler and Illingworth Method

number_Of_Air_T=0;
number_Of_Solid_T=0;

for ka = 1: b
   I = sprintf('CroppedImage_1.%02d.png', ka);
   I=imread(I);
   [Row,Col,r]=MaskPortion(I);
   sizeofmatrix = size(I,1);
   c=[];
   c1=zeros(sizeofmatrix,sizeofmatrix);
   c1=uint8(c1);
   e=0;
   for i=1:sizeofmatrix
       for j=1:sizeofmatrix
           d=(j - (Row))^2 + (i - (Col))^2;
           if d <= (r)^2
               e=e+1;
               c(e)=I(i,j);
               c1(i,j)=I(i,j);
           end
       end
   end

e=uint8(c);
p=imhist(e);
n=sum(p);
pi=zeros(256,1);

for i=1:256
    pi(i)=p(i)/n;
end
pi(1:256,2)=linspace(1,256,256);
```

```
N=0;
for q=1:256
    if pi(q,1)~=0
        N=N+1;
        pi1(N,1)=pi(q,1);
        pi1(N,2)=pi(q,2);
    end
end

size(pi1,1);
c=size(pi1,1);

P1=zeros(c,1);
P2=zeros(c,1);
u1=zeros(c,1);
u2=zeros(c,1);
s1=zeros(c,1);
s1_d=zeros(c,1);
s1_final=zeros(c,1);
s2_final=zeros(c-2,1);
J=zeros(c-2,1);

for i=1:c
    g=1:i;
    g1=i+1:c-1;
    P1(i)=sum(pi1(1:i));
    P2(i)=sum(pi1(i+1:c-1));
    u1(i)=sum(pi1(1:i).*g)/P1(i);
    u2(i)=sum(pi1(i+1:c-1).*g1)/P2(i);
    u2=u2(isfinite(u2));
    u2=u2(u2~=0);
    u2_final=u2';
    s1(i)=0;

    for ii=1:i
        s1(i)=((ii-u1(i))^2*pi1(ii));
        s1_n=cumsum(s1);
        s1_d(i)=P1(ii);
        s1_dd=s1_d;
        s1_final=s1_n./s1_dd;
    end

    G3=([2:c-1]);
    G4=G3';
    A=u2_final(:,1);

   end

   B=pi1([2:c-1]);
   Pi1_adj=B';
   s2_n=Pi1_adj.*(G4-A).^2;

   for i = 1:c
      for ii = i:c-2
```

```
            s2_final(ii)=sum(s2_n(i+1:c-2))/P2(ii);
            end
        end

        s1_final=sqrt(s1_final);
        s2_final=sqrt(s2_final);
        P1=P1([1:c-2]);
        P2=P2([1:c-2]);
        s1_final=s1_final([1:c-2]);
        J=1+2.*((P1.*log(s1_final))+(P2.*log(s2_final)))-
2.*((P1.*log(P1))+(P2.*log(P2)));
        J=J(~isinf(J));

        minrow=max(J);
        Threshold1=max(find(J>=minrow));
        Threshold1a(ka,1)=max(find(J>=minrow));

    % Portion of Code that Calculates the Porosity and Void Ratio of
the Image

    c2=zeros(sizeofmatrix,sizeofmatrix);
    c2=uint8(c2);
    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
            d=(j - (Row))^2 + (i - (Col))^2;
            if d <= (r)^2
                if c1(i,j)<Threshold1
                    c2(i,j)=20;
                else
                    c2(i,j)=100;
                end

            end
        end
    end

    number_Of_Air_Pixels=sum(sum(c2==20));
    number_Of_Solid_Pixels=sum(sum(c2==100));

    number_Of_Air_T=number_Of_Air_T+number_Of_Air_Pixels;
    number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;

    end

    Threshold1a=sum(Threshold1a(1:b));
    Threshold1a=Threshold1a/(b);
    Threshold1a=round(Threshold1a);
    AT1=Threshold1a;
    AT1=num2str(AT1);
    str8=AT1;

    void_ratio = number_Of_Air_T/number_Of_Solid_T;

    v=void_ratio;
```

```
      v=num2str(v);
      str13=v;

      AH{9,1}='Kittler and Illingworth segmentation method complete.';
      AH{10,1}='All methods complete.';
      set(handles.listbox3,'String',AH);
      drawnow()

      f =figure;
      set(f,'visible','off');
      r = cell(5,3);
      t=uitable(r);

      ColumnName = {'Segmentation Method','Void Ratio','Average
Threshold'};
      d= {O,str9,str4;P,str10,str5;KSW,str11,str6;'Johannsen and
Bille',str12,str7;K,str13,str8};
      t.Data = d;
      t.Position = [20 200 400 150];

      A=[ColumnName ; d];
      filename='SegmentationData.xlsx';
      xlswrite(filename,A);

      msg = msgbox('Data is succesfully saved as
SegmentationData.xlsx');
      waitfor(msg);

      % --- Executes on selection change in listbox3.
      function listbox3_Callback(hObject, eventdata, handles)
      % hObject    handle to listbox3 (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)

      % Hints: contents = cellstr(get(hObject,'String')) returns
listbox3 contents as cell array
      %        contents{get(hObject,'Value')} returns selected item from
listbox3

      % --- Executes during object creation, after setting all
properties.
      function listbox3_CreateFcn(hObject, eventdata, handles)
      % hObject    handle to listbox3 (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    empty - handles not created until after all
CreateFcns called

      % Hint: listbox controls usually have a white background on
Windows.
      %       See ISPC and COMPUTER.
      if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
        set(hObject,'BackgroundColor','white');
    end

    function file_menu_Callback(hObject, eventdata, handles)
    % hObject    handle to file_menu (see GCBO)
    % eventdata  reserved - to be defined in a future version of
MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % ------------------------------------------------------------------
----
    function menu_file_open_Callback(hObject, eventdata, handles)
    d = uigetdir('*.png');
    cd(d);
    q=dir('*.png');
    n = numel(q);

    n=num2str(n);
    set(handles.numImages, 'String', n);

    [filename, filepath] = uigetfile('*.*', 'Please Select First
Cropped Image from Folder that Appears');
    image = imread([filepath filename]);

    %scan the filename and parse the data int a cell string array, C
    C = textscan(filename, '%s %d %s', 'delimiter', '.');

    %bring axes into focus anf show image
    axes(handles.axes1);
    imshow(image, []); %[] = [Imin Imax]

    axes(handles.axes4);
    imshow(image, []);

    %store relevant data in the axes
    setappdata(handles.axes1, 'fileName',  filename);
    setappdata(handles.axes1, 'filePath',  filepath);
    setappdata(handles.axes1, 'image',     image);
    setappdata(handles.axes1, 'ptNum',     C{1});
    setappdata(handles.axes1, 'sliceNum',  C{2});
    setappdata(handles.axes1, 'imageType', C{3});
    guidata(hObject,handles);

    %update slider
    updateSlider(handles);

    axes(handles.axes2);
    sizeofmatrix = size(image,1);
    [Row,Col,r]=MaskPortion(image);
    c=[];
    c1=zeros(sizeofmatrix,sizeofmatrix);
    c1=uint8(c1);
    e=0;
    for i=1:sizeofmatrix
```

```
          for j=1:sizeofmatrix
              d=(j - (Row))^2 + (i - (Col))^2;
              if d <= (r)^2
                  e=e+1;
                  c(e)=image(i,j);
                  c1(i,j)=image(i,j);
              end
          end
      end

      histogram(c);

      set(handles.CurrentName, 'String', filename);

      function Close_file_Callback(hObject, eventdata, handles)
      % hObject    handle to Close_file (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)

      close(gcf);

      % --------------------------------------------------------------------
----
      function Save_file_Callback(hObject, eventdata, handles)

      %%% OTSU

      number_Of_Void_T=0;
      number_Of_Solid_T=0;

      AA = uigetdir('','Please Select the Folder Containing the Cropped
Images');
      cd (AA);
      BB = mkdir('Otsu Segmented Images');
      CC = strcat(AA,'\Otsu Segmented Images')
      cd (AA);
      x = '*.png';
      b=dir(x);
      b=length(b);

      for ka = 1: b

         I = sprintf('CroppedImage_1.%02d.png', ka);
         I=imread(I);
         [Row,Col,r]=MaskPortion(I);
         sizeofmatrix = size(I,1);
         c=[];
         c1=zeros(sizeofmatrix,sizeofmatrix);
         c1=uint8(c1);
         e=0;
         for i=1:sizeofmatrix
             for j=1:sizeofmatrix
                 d=(j - (Row))^2 + (i - (Col))^2;
```

```
            if d <= (r)^2
                e=e+1;
                c(e)=I(i,j);
                c1(i,j)=I(i,j);
            end
        end
    end

e=uint8(c);
p=imhist(e);
mean=0;

for i=1:256
    mean=mean+(i*p(i,1));
end
mean=mean/256;
sumt=zeros(256,1);
for i=1:256
    meanli=0;
    for j=1:i-1
        meanli=meanli+(j*p(j,1));
    end
    meanli=meanli/256;
    meangi=0;
    for j=i:256
        meangi=meangi+(j*p(j,1));
    end
    meangi=meangi/256;
    sumli=0;
    sumgi=0;
    for ii=1:i-1
        sumli=sumli+p(ii,1)*((meanli-mean)^2);
    end
    for ii=i:256
        sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
    end
    sumt(i,1)=sumgi+sumli;
end

Threshold1=find(sumt==max(sumt));

% Portion of Code that Calculates the Porosity and Void Ratio of
the Image

c2=zeros(sizeofmatrix,sizeofmatrix);
c2=uint8(c2);
for i=1:sizeofmatrix
    for j=1:sizeofmatrix
        d=(j - (Row))^2 + (i - (Col))^2;
        if d <= (r)^2
            if c1(i,j)<Threshold1
                c2(i,j)=20;
            else
                c2(i,j)=100;
```

```
            end

        end
    end
end

number_Of_Void_Pixels=sum(sum(c2==20));
number_Of_Solid_Pixels=sum(sum(c2==100));

number_Of_Void_T=number_Of_Void_T+number_Of_Void_Pixels;
number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;

QQ = 'Otsu Segmented Image_1.0';

cd (CC);
imwrite(c2,[QQ,num2str(ka),'.png']);
cd (AA);
end

%%% PUN

number_Of_Void_T=0;
number_Of_Solid_T=0;

mkdir('Pun Segmented Images');
cd (AA);
BB = mkdir('Pun Segmented Images');
CC = strcat(AA,'\Pun Segmented Images')
cd (AA);
x = '*.png';
b=dir(x);
b=length(b);

for ka = 1: b
    I = sprintf('CroppedImage_1.%02d.png', ka);
    I=imread(I);
    [Row,Col,r]=MaskPortion(I);
    sizeofmatrix = size(I,1);
    c=[];
    c1=zeros(sizeofmatrix,sizeofmatrix);
    c1=uint8(c1);
    e=0;
    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
            d=(j - (Row))^2 + (i - (Col))^2;
            if d <= (r)^2
                e=e+1;
                c(e)=I(i,j);
            c1(i,j)=I(i,j);
        end
    end
    end

% Automatic Threshold using Pun's Method
```

```
e=uint8(c);
p=imhist(e);
n=sum(p);
pi=zeros(256,1);

for i=1:256
      pi(i)=p(i)/n;
end

for i=1:256
      Sumpi=sum(pi(1:i));
      if Sumpi>=0.5
          m=i;
          break
      end
end

    numerator = pi(1:m).*log(pi(1:m));
    numerator = nansum(numerator);

    for i=1:256
    denominator = pi(1:i).*log(pi(1:i));
    end

    denominator = nansum(denominator);

    alpha = numerator/denominator;

    if alpha <=0.5
        correctedalpha = 1-alpha;
    else
        correctedalpha = alpha;
    end

    for i=1:256
      Summationpi=sum(pi(1:i));
      if Summationpi>=correctedalpha
          Threshold = i;
          break
      end
    end

    Threshold1 = Threshold;



    % Portion of Code that Calculates the Porosity and Void Ratio of
the Image

    c2=zeros(sizeofmatrix,sizeofmatrix);
    c2=uint8(c2);
    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
```

```
        d=(j - (Row))^2 + (i - (Col))^2;
        if d <= (r)^2
            if c1(i,j)<Threshold1
                c2(i,j)=20;
            else
                c2(i,j)=100;
            end

        end
    end
end

number_Of_Void_Pixels=sum(sum(c2==20));
number_Of_Solid_Pixels=sum(sum(c2==100));

number_Of_Void_T=number_Of_Void_T+number_Of_Void_Pixels;
number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;

QT = 'Pun Segmented Image_1.0';

cd (CC);
imwrite(c2,[QT,num2str(ka),'.png']);
cd (AA);
end

%%% KAPUR, SAHOO, AND WONG

number_Of_Void_T=0;
number_Of_Solid_T=0;

cd (AA);
BB = mkdir('Kapur, Sahoo, and Wong Segmented Images');
CC = strcat(AA,'\Kapur, Sahoo, and Wong Segmented Images')
cd (AA);
x = '*.png';
b=dir(x);
b=length(b);

          for ka = 1: b
   I = sprintf('CroppedImage_1.%02d.png', ka);
   I=imread(I);
   [Row,Col,r]=MaskPortion(I);
   sizeofmatrix = size(I,1);
   c=[];
   c1=zeros(sizeofmatrix,sizeofmatrix);
   c1=uint8(c1);
   e=0;
   for i=1:sizeofmatrix
       for j=1:sizeofmatrix
           d=(j - (Row))^2 + (i - (Col))^2;
           if d <= (r)^2
               e=e+1;
               c(e)=I(i,j);
               c1(i,j)=I(i,j);
```

```
          end
       end
   end

% Automatic Threshold using Kapur, Sahoo, and Wong Method

e=uint8(c);
p=imhist(e);
n=sum(p);
pi=zeros(256,1);
for i=1:256
    pi(i)=p(i)/n;
end

pi(1:256,2)=linspace(1,256,256);
N=0;
for q=1:256
    if pi(q,1)~=0
        N=N+1;
        pi1(N,1)=pi(q,1);
        pi1(N,2)=pi(q,2);
    end
end

pi1= pi1(pi1(:,1)>0.0001,:);

size(pi1,1);
c=size(pi1,1);

sumt = zeros(c,1);
Ht= zeros(c,1);
Pt=zeros(c,1);
Ha=zeros(c,1);
Hb=zeros(c,1);

for t = 1:c
      Hi= -sum(pi1(1:c).*log(pi1(1:c)));
     for i = 1:c
         Pt(i)=0;
         Ht(t)=0;
         Pt(i) = sum(pi1(1:i))+Pt(i);
         Ht(t) = -sum(pi1(1:t).*log(pi1(1:t)))+Ht(t);
         Ha(i) = log(Pt(i))+(Ht(i)/Pt(i));
         Hb(i) = log(1-Pt(i))+((Hi-Ht(i))/(1-Pt(i)));
         Hb(i)=real(Hb(i));
     end

     sumt=Ha+Hb;
     sumt(:,2)=pi1(:,2);
     maxrow=max(sumt(:,1));
     Threshold1=max(sumt(sumt(:,1)==maxrow,2));
end

% Portion of Code that Calculates the Porosity and Void Ratio of
```

```
the Image

      c2=zeros(sizeofmatrix,sizeofmatrix);
      c2=uint8(c2);
      for i=1:sizeofmatrix
          for j=1:sizeofmatrix
              d=(j - (Row))^2 + (i - (Col))^2;
              if d <= (r)^2
                  if c1(i,j)<Threshold1
                      c2(i,j)=20;
                  else
                      c2(i,j)=100;
                  end

              end
          end
      end

      number_Of_Void_Pixels=sum(sum(c2==20));
      number_Of_Solid_Pixels=sum(sum(c2==100));

      number_Of_Void_T=number_Of_Void_T+number_Of_Void_Pixels;
      number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;

      QA = 'Kapur, Sahoo, and Wong Segmented Image_1.0';
      cd (CC);

      imwrite(c2,[QA,num2str(ka),'.png']);
      cd (AA);

                end

      %%% JOHANNSEN AND BILLE

      number_Of_Void_T=0;
      number_Of_Solid_T=0;

      cd (AA);
      BB = mkdir('Johannsen and Bille Segmented Images');
      CC = strcat(AA,'\Johannsen and Bille Segmented Images')
      cd (AA);
      x = '*.png';
      b=dir(x);
      b=length(b);

      for ka = 1: b
         I = sprintf('CroppedImage_1.%02d.png', ka);
         I=imread(I);
         [Row,Col,r]=MaskPortion(I);
         sizeofmatrix = size(I,1);
         c=[];
         c1=zeros(sizeofmatrix,sizeofmatrix);
         c1=uint8(c1);
         e=0;
```

```
   for i=1:sizeofmatrix
       for j=1:sizeofmatrix
           d=(j - (Row))^2 + (i - (Col))^2;
           if d <= (r)^2
               e=e+1;
               c(e)=I(i,j);
               c1(i,j)=I(i,j);
           end
       end
   end

% Automatic Threshold using Johannsen and Bille Method

e=uint8(c);
p=imhist(e);
n=sum(p);
pi=zeros(256,1);
for i=1:256
    pi(i)=p(i)/n;
end

pi(1:256,2)=linspace(1,256,256);
N=0;
for q=1:256
    if pi(q,1)~=0
        N=N+1;
        pi1(N,1)=pi(q,1);
        pi1(N,2)=pi(q,2);
    end
end

pi1= pi1(pi1(:,1)>0.0001,:);

size(pi1,1);
c=size(pi1,1);

   sumt = zeros(c,1);
   Pn=zeros(c,1);
   Pt=zeros(c,1);
   Pi=zeros(c,1);
   Pr=zeros(c,1);
   Sn=zeros(c,1);
   Sn_bar=zeros(c,1);

for t=1:c
    n=t+1;

    Pn(t)=sum(pi1(1:n));
    Pt(t)=sum(pi1(1:n-1));

    Sn(t)=(log(Pn(t)))-
(1/Pn(t)).*(((pi1(n)).*log(pi1(n)))+(Pt(t).*log(Pt(t))));
    Sn(t)=real(Sn(t));
```

```
          Pi(t)=sum(pi1(n:c));
          Pr(t)=sum(pi1(n+1:c));

          Sn_bar(t)=(log(Pi(t)))-
(1/Pi(t)).*(((pi1(n)).*log(pi1(n)))+(Pr(t).*log(Pr(t))));
          Sn_bar(t)=real(Sn_bar(t));

          sumt=Sn+Sn_bar;
          sumt(:,2)=pi1(:,2);
          minrow=min(sumt(:,1));
          Threshold1=min(sumt(sumt(:,1)==minrow,2));

      end

      % Portion of Code that Calculates the Porosity and Void Ratio of
the Image

      c2=zeros(sizeofmatrix,sizeofmatrix);
      c2=uint8(c2);
      for i=1:sizeofmatrix
          for j=1:sizeofmatrix
              d=(j - (Row))^2 + (i - (Col))^2;
              if d <= (r)^2
                  if c1(i,j)<Threshold1
                      c2(i,j)=20;
                  else
                      c2(i,j)=100;
                  end

              end
          end
      end

      number_Of_Void_Pixels=sum(sum(c2==20));
      number_Of_Solid_Pixels=sum(sum(c2==100));

      number_Of_Void_T=number_Of_Void_T+number_Of_Void_Pixels;
      number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;

      QY = 'Johannsen and Bille Segmented Image_1.0';
      cd (CC);

      imwrite(c2,[QY,num2str(ka),'.png']);
        cd (AA);

      end

      %%% KITTLER AND ILLINGWORTH

      number_Of_Void_T=0;
      number_Of_Solid_T=0;

      cd (AA);
      BB = mkdir('Kittler and Illingworth Segmented Images');
```

```
CC = strcat(AA,'\Kittler and Illingworth Segmented Images')
cd (AA);
x = '*.png';
b=dir(x);
b=length(b);

for ka = 1: b
   I = sprintf('CroppedImage_1.%02d.png', ka);
   I=imread(I);
   [Row,Col,r]=MaskPortion(I);
   sizeofmatrix = size(I,1);
   c=[];
   c1=zeros(sizeofmatrix,sizeofmatrix);
   c1=uint8(c1);
   e=0;
   for i=1:sizeofmatrix
       for j=1:sizeofmatrix
           d=(j - (Row))^2 + (i - (Col))^2;
           if d <= (r)^2
               e=e+1;
               c(e)=I(i,j);
               c1(i,j)=I(i,j);
           end
       end
   end

% Automatic Threshold using Kittler and Illingworth Method

e=uint8(c);
p=imhist(e);
n=sum(p);
pi=zeros(256,1);
for i=1:256
    pi(i)=p(i)/n;
end

pi(1:256,2)=linspace(1,256,256);
N=0;
for q=1:256
    if pi(q,1)~=0
        N=N+1;
        pi1(N,1)=pi(q,1);
        pi1(N,2)=pi(q,2);
    end
end

size(pi1,1);
c=size(pi1,1);

   P1=zeros(c,1);
   P2=zeros(c,1);
   u1=zeros(c,1);
   u2=zeros(c,1);
   s1=zeros(c,1);
```

```
        s1_d=zeros(c,1);
        s1_final=zeros(c,1);
        s2_final=zeros(c-2,1);
        J=zeros(c-2,1);

        for i=1:c
            g=1:i;
            g1=i+1:c-1;
            P1(i)=sum(pi1(1:i));
            P2(i)=sum(pi1(i+1:c-1));
            u1(i)=sum(pi1(1:i).*g)/P1(i);
            u2(i)=sum(pi1(i+1:c-1).*g1)/P2(i);
            u2=u2(isfinite(u2));
            u2=u2(u2~=0);
            u2_final=u2';
            s1(i)=0;

            for ii=1:i
            s1(i)=((ii-u1(i))^2*pi1(ii));
            s1_n=cumsum(s1);
            s1_d(i)=P1(ii);
            s1_dd=s1_d;
            s1_final=s1_n./s1_dd;

            end
                G3=([2:c-1]);
                G4=G3';
                A=u2_final(:,1);
        end

        B=pi1([2:c-1]);
        Pi1_adj=B';
        s2_n=Pi1_adj.*(G4-A).^2;

       for i = 1:c
           for ii = i:c-2
           s2_final(ii)=sum(s2_n(i+1:c-2))/P2(ii);
           end
       end

       s1_final=sqrt(s1_final);
       s2_final=sqrt(s2_final);

       P1=P1([1:c-2]);
       P2=P2([1:c-2]);
       s1_final=s1_final([1:c-2]);
       J=1+2.*((P1.*log(s1_final))+(P2.*log(s2_final)))-
2.*((P1.*log(P1))+(P2.*log(P2)));
       J=J(~isinf(J));

       minrow=max(J);
       Threshold1=max(find(J>=minrow));

     % Portion of Code that Calculates the Porosity and Void Ratio of
```

```
the Image

      c2=zeros(sizeofmatrix,sizeofmatrix);
      c2=uint8(c2);
      for i=1:sizeofmatrix
          for j=1:sizeofmatrix
              d=(j - (Row))^2 + (i - (Col))^2;
              if d <= (r)^2
                  if c1(i,j)<Threshold1
                      c2(i,j)=20;
                  else
                      c2(i,j)=100;
                  end

              end
          end
      end

      number_Of_Void_Pixels=sum(sum(c2==20));
      number_Of_Solid_Pixels=sum(sum(c2==100));

      number_Of_Void_T=number_Of_Void_T+number_Of_Void_Pixels;
      number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;

      QP = 'Kittler and Illingworth Segmented Image_1.0';

      cd (CC);

      imwrite(c2,[QP,num2str(ka),'.png']);
        cd (AA);

      end
                 h=msgbox('All Segmented Images Successfully Saved!');

      function NewProject_open_Callback(hObject, eventdata, handles)
      OrigDlgH = ancestor(hObject, 'figure');
      delete(OrigDlgH);
      TwoPhaseImageSegmentation;

      function Tools_Callback(hObject, eventdata, handles)
      % hObject    handle to Tools (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)

      % ------------------------------------------------------------------
----
      function CB_open_Callback(hObject, eventdata, handles)
      axes(handles.axes4);
      imcontrast(gcf);

      % ------------------------------------------------------------------
----
      function Export_Data_Callback(hObject, eventdata, handles)
```

```
str1='Segmentation Method';
O='Otsu';
P= 'Pun';
KSW= 'Kapur, Sahoo, and Wong';
J = 'Johannsen and Bille';
K='Kittler and Illingworth';

str2='Void Ratio';
str3='Average Threshold Value for all Images';

d = uigetdir('*.png');
cd(d);
q=dir('*.png');
b = numel(q);
k = warndlg('Exporting of data can take a couple of minutes
depending on the number of images being analyzed. Please do not click
anything until data exporting is completed.','Warning');
waitfor(k);

%%% OTSU

number_Of_Air_T=0;
number_Of_Solid_T=0;

for ka = 1: b
   I = sprintf('CroppedImage_1.%02d.png', ka);
   I=imread(I);
   [Row,Col,r]=MaskPortion(I);
   sizeofmatrix = size(I,1);
   c=[];
   c1=zeros(sizeofmatrix,sizeofmatrix);
   c1=uint8(c1);
   e=0;
   for i=1:sizeofmatrix
       for j=1:sizeofmatrix
           d=(j - (Row))^2 + (i - (Col))^2;
           if d <= (r)^2
               e=e+1;
               c(e)=I(i,j);
               c1(i,j)=I(i,j);
           end
       end
   end

% Automatic Threshold using Otsu's Method

e=uint8(c);
p=imhist(e);
mean=0;
for i=1:256
    mean=mean+(i*p(i,1));
end
mean=mean/256;
sumt=zeros(256,1);
```

```
    for i=1:256
        meanli=0;
        for j=1:i-1
            meanli=meanli+(j*p(j,1));
        end
        meanli=meanli/256;
        meangi=0;
        for j=i:256
            meangi=meangi+(j*p(j,1));
        end
        meangi=meangi/256;
        sumli=0;
        sumgi=0;
        for ii=1:i-1
            sumli=sumli+p(ii,1)*((meanli-mean)^2);
        end

        for ii=i:256
            sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
        end

        sumt(i,1)=sumgi+sumli;

    end

    Threshold1=find(sumt==max(sumt));
    Threshold1a(ka,1)=find(sumt==max(sumt));

    % Portion of Code that Calculates the Porosity and Void Ratio of
the Image

    c2=zeros(sizeofmatrix,sizeofmatrix);
    c2=uint8(c2);
    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
            d=(j - (Row))^2 + (i - (Col))^2;
            if d <= (r)^2
                if c1(i,j)<Threshold1
                    c2(i,j)=20;
                else
                    c2(i,j)=100;
                end

            end
        end
    end

    number_Of_Air_Pixels=sum(sum(c2==20));
    number_Of_Solid_Pixels=sum(sum(c2==100));

    number_Of_Air_T=number_Of_Air_T+number_Of_Air_Pixels;
    number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;

    end
```

```
Threshold1a=sum(Threshold1a(1:b));
Threshold1a=Threshold1a/(b);
Threshold1a=round(Threshold1a);
AT1=Threshold1a;
AT1=num2str(AT1);
str4 = AT1;
void_ratio = number_Of_Air_T/number_Of_Solid_T;

v=void_ratio;
v=num2str(v);
str9=v;

AH=cell(10,1);
AH{1,1}='Otsu segmentation method complete.';
AH{2,1}='Four methods remain...';
set(handles.listbox3, 'String', AH);
drawnow()

%%% PUN

number_Of_Air_T=0;
number_Of_Solid_T=0;

for ka = 1: b
    I = sprintf('CroppedImage_1.%02d.png', ka);
    I=imread(I);
    [Row,Col,r]=MaskPortion(I);
    sizeofmatrix = size(I,1);

c=[];
c1=zeros(sizeofmatrix,sizeofmatrix);
c1=uint8(c1);
e=0;

for i=1:sizeofmatrix
    for j=1:sizeofmatrix
        d=(j - (Row))^2 + (i - (Col))^2;
        if d <= (r)^2
            e=e+1;
            c(e)=I(i,j);
            c1(i,j)=I(i,j);
        end
    end
end

% Pun

e=uint8(c);
p=imhist(e);
n=sum(p);
pi=zeros(256,1);

for i=1:256
```

```
      pi(i)=p(i)/n;
end

for i=1:256
      Sumpi=sum(pi(1:i));
      if Sumpi>=0.5
          m=i;
          break
      end
end

    numerator = pi(1:m).*log(pi(1:m));
    numerator = nansum(numerator);

    for i=1:256
    denominator = pi(1:i).*log(pi(1:i));
    end

    denominator = nansum(denominator);

    alpha = numerator/denominator;

    if alpha <=0.5
        correctedalpha = 1-alpha;
    else
        correctedalpha = alpha;
    end

    for i=1:256
      Summationpi=sum(pi(1:i));
      if Summationpi>=correctedalpha
          Threshold = i;
          break
      end
    end

    Threshold1=Threshold;
    Threshold1a(ka,1) = Threshold;

% Portion of Code that Calculates the Porosity and Void Ratio of
the Image

c2=zeros(sizeofmatrix,sizeofmatrix);
c2=uint8(c2);
for i=1:sizeofmatrix
    for j=1:sizeofmatrix
        d=(j - (Row))^2 + (i - (Col))^2;
        if d <= (r)^2
            if c1(i,j)<Threshold1
                c2(i,j)=20;
            else
                c2(i,j)=100;
            end
```

```
        end
    end
end

number_Of_Air_Pixels=sum(sum(c2==20));
number_Of_Solid_Pixels=sum(sum(c2==100));

number_Of_Air_T=number_Of_Air_T+number_Of_Air_Pixels;
number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;

end

Threshold1a=sum(Threshold1a(1:b));
Threshold1a=Threshold1a/(b);
Threshold1a=round(Threshold1a);
AT1=Threshold1a;
AT1=num2str(AT1);

str5=AT1;

void_ratio = number_Of_Air_T/number_Of_Solid_T;

v=void_ratio;
v=num2str(v);
str10=v;


AH{3,1}='Pun segmentation method complete.';
AH{4,1}='Three methods remain...';
set(handles.listbox3,'String',AH);
drawnow()

%%% KAPUR, SAHOO, AND WONG

number_Of_Air_T=0;
number_Of_Solid_T=0;

for ka = 1: b
    I = sprintf('CroppedImage_1.%02d.png', ka);
    I=imread(I);
   [Row,Col,r]=MaskPortion(I);
    sizeofmatrix = size(I,1);
    c=[];
    c1=zeros(sizeofmatrix,sizeofmatrix);
    c1=uint8(c1);
    e=0;
    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
            d=(j - (Row))^2 + (i - (Col))^2;
            if d <= (r)^2
                e=e+1;
                c(e)=I(i,j);
                c1(i,j)=I(i,j);
        end
```

```
    end
end

% Automatic Threshold using Kapur, Sahoo, and Wong Method

e=uint8(c);
p=imhist(e);
n=sum(p);
pi=zeros(256,1);
for i=1:256
    pi(i)=p(i)/n;
end

pi(1:256,2)=linspace(1,256,256);
N=0;
for q=1:256
    if pi(q,1)~=0
        N=N+1;
        pi1(N,1)=pi(q,1);
        pi1(N,2)=pi(q,2);
    end
end

pi1= pi1(pi1(:,1)>0.0001,:);
size(pi1,1);
c=size(pi1,1);

sumt = zeros(c,1);
Ht= zeros(c,1);
Pt=zeros(c,1);
Ha=zeros(c,1);
Hb=zeros(c,1);

for t = 1:c
      Hi= -sum(pi1(1:c).*log(pi1(1:c)));
     for i = 1:c
         Pt(i)=0;
         Ht(t)=0;
         Pt(i) = sum(pi1(1:i))+Pt(i);
         Ht(t) = -sum(pi1(1:t).*log(pi1(1:t)))+Ht(t);
         Ha(i) = log(Pt(i))+(Ht(i)/Pt(i));
         Hb(i) = log(1-Pt(i))+((Hi-Ht(i))/(1-Pt(i)));
         Hb(i)=real(Hb(i));
     end

     sumt=Ha+Hb;

     sumt(:,2)=pi1(:,2);
     maxrow=max(sumt(:,1));

     Threshold1a=max(sumt(sumt(:,1)==maxrow,2));
     Threshold1(ka,1)=max(sumt(sumt(:,1)==maxrow,2));
end
```

```
    end

    Threshold1=sum(Threshold1(1:b));
    Threshold1=Threshold1/(b);
    Threshold1=round(Threshold1);
    AT1=Threshold1;
    AT1=num2str(AT1);

    % Portion of Code that Calculates the Porosity and Void Ratio of
the Image

    c2=zeros(sizeofmatrix,sizeofmatrix);
    c2=uint8(c2);
    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
            d=(j - (Row))^2 + (i - (Col))^2;
            if d <= (r)^2
                if c1(i,j)<Threshold1a
                    c2(i,j)=20;
                else
                    c2(i,j)=100;
                end

            end
        end
    end

    number_Of_Air_Pixels=sum(sum(c2==20));
    number_Of_Solid_Pixels=sum(sum(c2==100));

    number_Of_Air_T=number_Of_Air_T+number_Of_Air_Pixels;
    number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;

    str6=AT1;

    void_ratio = number_Of_Air_T/number_Of_Solid_T;

    v=void_ratio
    v=num2str(v);
    str11=v;
    AH{5,1}='Kapur, Sahoo, and Wong segmentation method complete.';
    AH{6,1}='Two methods remain...';
    set(handles.listbox3,'String',AH);
    drawnow()

    %%% JOHANNSEN AND BILLE

    number_Of_Air_T=0;
    number_Of_Solid_T=0;

    for ka = 1: b
       I = sprintf('CroppedImage_1.%02d.png', ka);
       I=imread(I);
       [Row,Col,r]=MaskPortion(I);
```

```
   sizeofmatrix = size(I,1);
   c=[];
   c1=zeros(sizeofmatrix,sizeofmatrix);
   c1=uint8(c1);
   e=0;
   for i=1:sizeofmatrix
       for j=1:sizeofmatrix
           d=(j - (Row))^2 + (i - (Col))^2;
           if d <= (r)^2
               e=e+1;
               c(e)=I(i,j);
               c1(i,j)=I(i,j);
           end
       end
   end

% Automatic Threshold using Johannsen and Bille Method

e=uint8(c);
p=imhist(e);
n=sum(p);
pi=zeros(256,1);
for i=1:256
    pi(i)=p(i)/n;
end

pi(1:256,2)=linspace(1,256,256);
N=0;
for q=1:256
    if pi(q,1)~=0
        N=N+1;
        pi1(N,1)=pi(q,1);
        pi1(N,2)=pi(q,2);
    end
end

size(pi1,1);
c=size(pi1,1);

sumt = zeros(c,1);
Pn=zeros(c,1);
Pt=zeros(c,1);
Pi=zeros(c,1);
Pr=zeros(c,1);
Sn=zeros(c,1);
Sn_bar=zeros(c,1);

for t=1:c
    n=t+1;

    Pn(t)=sum(pi1(1:n));
    Pt(t)=sum(pi1(1:n-1));

    Sn(t)=(log(Pn(t)))-
```

```
(1/Pn(t)).*(((pi1(n)).*log(pi1(n)))+(Pt(t).*log(Pt(t))));
          Sn(t)=real(Sn(t));

          Pi(t)=sum(pi1(n:c));
          Pr(t)=sum(pi1(n+1:c));

          Sn_bar(t)=(log(Pi(t)))-
(1/Pi(t)).*(((pi1(n)).*log(pi1(n)))+(Pr(t).*log(Pr(t))));
          Sn_bar(t)=real(Sn_bar(t));

          sumt=Sn+Sn_bar;
          pi1_final=pi1(15:end,:);
          sumt=sumt(15:end);
          sumt(:,2)=pi1_final(:,2);
          minrow=min(sumt(:,1));
          Threshold1a(ka,1)=min(sumt(sumt(:,1)==minrow,2));
           Threshold1=min(sumt(sumt(:,1)==minrow,2));
          end

      % Portion of Code that Calculates the Porosity and Void Ratio of
the Image

      c2=zeros(sizeofmatrix,sizeofmatrix);
      c2=uint8(c2);
      for i=1:sizeofmatrix
          for j=1:sizeofmatrix
              d=(j - (Row))^2 + (i - (Col))^2;
              if d <= (r)^2
                  if c1(i,j)<Threshold1
                      c2(i,j)=20;
                  else
                      c2(i,j)=100;
                  end

              end
          end
      end

      number_Of_Air_Pixels=sum(sum(c2==20));
      number_Of_Solid_Pixels=sum(sum(c2==100));

      number_Of_Air_T=number_Of_Air_T+number_Of_Air_Pixels;
      number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;

      end

      Threshold1a=sum(Threshold1a(1:b));
      Threshold1a=Threshold1a/(b);
      Threshold1a=round(Threshold1a);
      AT1=Threshold1a;
      AT1=num2str(AT1);
      str7=AT1;

      void_ratio = number_Of_Air_T/number_Of_Solid_T;
```

```
v=void_ratio;
v=num2str(v);
str12=v;

AH{7,1}='Johannsen and Bille segmentation method complete.';
AH{8,1}='One method remains...';
set(handles.listbox3,'String',AH);
drawnow()

% Automatic Threshold using Kittler and Illingworth Method

number_Of_Air_T=0;
number_Of_Solid_T=0;

for ka = 1: b
   I = sprintf('CroppedImage_1.%02d.png', ka);
   I=imread(I);
   [Row,Col,r]=MaskPortion(I);
   sizeofmatrix = size(I,1);
   c=[];
   c1=zeros(sizeofmatrix,sizeofmatrix);
   c1=uint8(c1);
   e=0;
   for i=1:sizeofmatrix
       for j=1:sizeofmatrix
           d=(j - (Row))^2 + (i - (Col))^2;
           if d <= (r)^2
               e=e+1;
               c(e)=I(i,j);
               c1(i,j)=I(i,j);
           end
       end
   end

e=uint8(c);
p=imhist(e);
n=sum(p);
pi=zeros(256,1);

for i=1:256
    pi(i)=p(i)/n;
end
pi(1:256,2)=linspace(1,256,256);
N=0;
for q=1:256
    if pi(q,1)~=0
        N=N+1;
        pi1(N,1)=pi(q,1);
        pi1(N,2)=pi(q,2);
    end
end

size(pi1,1);
```

```
      c=size(pi1,1);
      P1=zeros(c,1);
      P2=zeros(c,1);
      u1=zeros(c,1);
      u2=zeros(c,1);
      s1=zeros(c,1);
      s1_d=zeros(c,1);
      s1_final=zeros(c,1);
      s2_final=zeros(c-2,1);
      J=zeros(c-2,1);

      for i=1:c
          g=1:i;
          g1=i+1:c-1;
          P1(i)=sum(pi1(1:i));
          P2(i)=sum(pi1(i+1:c-1));
          u1(i)=sum(pi1(1:i).*g)/P1(i);
          u2(i)=sum(pi1(i+1:c-1).*g1)/P2(i);
          u2=u2(isfinite(u2));
          u2=u2(u2~=0);
          u2_final=u2';
          s1(i)=0;

          for ii=1:i
              s1(i)=((ii-u1(i))^2*pi1(ii));
              s1_n=cumsum(s1);
              s1_d(i)=P1(ii);
              s1_dd=s1_d;
              s1_final=s1_n./s1_dd;
          end
                 G3=([2:c-1]);
                 G4=G3';
                 A=u2_final(:,1);
         end

         B=pi1([2:c-1]);
         Pi1_adj=B';
         s2_n=Pi1_adj.*(G4-A).^2;

        for i = 1:c
            for ii = i:c-2
            s2_final(ii)=sum(s2_n(i+1:c-2))/P2(ii);
            end
        end

      s1_final=sqrt(s1_final);
      s2_final=sqrt(s2_final);

      P1=P1([1:c-2]);
      P2=P2([1:c-2]);
      s1_final=s1_final([1:c-2]);
      J=1+2.*((P1.*log(s1_final))+(P2.*log(s2_final)))-
2.*((P1.*log(P1))+(P2.*log(P2)));
      J=J(~isinf(J));
```

```
      minrow=max(J);
      Threshold1=max(find(J>=minrow));
      Threshold1a(ka,1)=max(find(J>=minrow));

      % Portion of Code that Calculates the Porosity and Void Ratio of
the Image

      c2=zeros(sizeofmatrix,sizeofmatrix);
      c2=uint8(c2);
      for i=1:sizeofmatrix
          for j=1:sizeofmatrix
              d=(j - (Row))^2 + (i - (Col))^2;
              if d <= (r)^2
                  if c1(i,j)<Threshold1
                      c2(i,j)=20;
                  else
                      c2(i,j)=100;
                  end

              end
          end
      end

      number_Of_Air_Pixels=sum(sum(c2==20));
      number_Of_Solid_Pixels=sum(sum(c2==100));

      number_Of_Air_T=number_Of_Air_T+number_Of_Air_Pixels;
      number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;
      Threshold1a;
      end

      Threshold1a=sum(Threshold1a(1:b));
      Threshold1a=Threshold1a/(b);
      Threshold1a=round(Threshold1a);
      AT1=Threshold1a;
      AT1=num2str(AT1);
      str8=AT1;

      void_ratio = number_Of_Air_T/number_Of_Solid_T;

      v=void_ratio;
      v=num2str(v);
      str13=v;

      AH{9,1}='Kittler and Illingworth segmentation method complete.';
      AH{10,1}='All methods complete.';
      set(handles.listbox3,'String',AH);
      drawnow()

      f =figure;
      set(f,'visible','off');
      r = cell(5,3);
      t=uitable(r);
```

```
      ColumnName = {'Segmentation Method','Void Ratio','Average
Threshold'};
      d= {O,str9,str4;P,str10,str5;KSW,str11,str6;'Johannsen and
Bille',str12,str7;K,str13,str8};
      t.Data = d;
      t.Position = [20 200 400 150];

      A=[ColumnName ; d];
      filename='SegmentationData.xlsx';
      xlswrite(filename,A);

      msg = msgbox('Data is succesfully saved as
SegmentationData.xlsx');
      waitfor(msg);

      function Help_Tag_Callback(hObject, eventdata, handles)
      % hObject    handle to Help_Tag (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)

      % ------------------------------------------------------------------
----
      function User_guide_Callback(hObject, eventdata, handles)
      % hObject    handle to User_guide (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)
      h = msgbox({'For proper program usage, the following steps must be
used in the displayed order:';'';'Step 1: Click "Rename Images." This
button renames all of the images in the folder of interest. Make sure
that the original images are saved to a folder.';'';'Step 2: Click
"Crop All Images." This button crops the renamed images and saves the
cropped images to a folder of choice. In order to apply cropping,
please double click within the drawn ellipse.';'';'Step 3: Click "Load
Image." This button loads the first cropped image and its corresponding
histogram. If necessary, the contrast and brightness of the image can
be adjusted as well. The slider allows for the viewing of all of the
remaining cropped images.';'';'Step 4: After choosing a segmentation
method, click "Run." This button allows for the selection of all of the
cropped images. The void ratio (e) and average threshold value for all
the cropped images are displayed. The segmentation of the first cropped
image is shown and the slider allows for the viewing of the remaining
cropped images with the chosen segmentation method applied.';'';'Note:
The "Export Data to Excel" button saves the void ratio (e) and average
threshold value for all images, per segmentation technique, to an Excel
file.' },'Program Instructions');

      function About_Us_Callback(hObject, eventdata, handles)
      % hObject    handle to About_Us (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)
```

```
      h = msgbox({'University of Delaware';'Geotechnical
Engineering';'Civil and Environmental Engineering';'301 DuPont
Hall';'Newark, DE 19716';'USA'});

      % --- Executes on selection change in Methods.
      function Methods_Callback(hObject, eventdata, handles)
      % hObject    handle to Methods (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)

      % Hints: contents = cellstr(get(hObject,'String')) returns Methods
contents as cell array
      %        contents{get(hObject,'Value')} returns selected item from
Methods

      % --- Executes during object creation, after setting all
properties.
      function Methods_CreateFcn(hObject, eventdata, handles)
      % hObject    handle to Methods (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    empty - handles not created until after all
CreateFcns called

      % Hint: popupmenu controls usually have a white background on
Windows.
      %       See ISPC and COMPUTER.
      if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
          set(hObject,'BackgroundColor','white');
      end

      % --- Executes during object creation, after setting all
properties.
      function axes1_CreateFcn(hObject, eventdata, handles)
      title('Load Image','FontSize',11.5);
      set(gca,'Xtick',[],'Ytick',[]);

      function axes2_CreateFcn(hObject, eventdata, handles)
      title('Image Histogram','FontSize',11.5);
      set(gca,'Xtick',[],'Ytick',[]);
      xlabel('Grayscale Pixel Intensity','FontSize',10);
      ylabel('Grayscale Pixel Frequency','FontSize',10);

      % --- Executes during object creation, after setting all
properties.
      function axes3_CreateFcn(hObject, eventdata, handles)
      title('Segmented Image','FontSize',11.5);
      set(gca,'Xtick',[],'Ytick',[]);

      % --- Executes during object creation, after setting all
properties.
      function axes4_CreateFcn(hObject, eventdata, handles)
```

```
      title('Image Brightness','FontSize',11.5);
      set(gca,'Xtick',[],'Ytick',[]);

      % --- Executes when figure1 is resized.
      function figure1_SizeChangedFcn(hObject, eventdata, handles)
      % hObject    handle to figure1 (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)
```

Appendix C

THREE-PHASE IMAGE SEGMENTATION GUI CODE

```
      function varargout = ThreePhaseImageSegmentation(varargin)

      gui_Singleton = 1;
      gui_State = struct('gui_Name',       mfilename, ...
                         'gui_Singleton',  gui_Singleton, ...
                         'gui_OpeningFcn',
@ThreePhaseImageSegmentation_OpeningFcn, ...
                         'gui_OutputFcn',
@ThreePhaseImageSegmentation_OutputFcn, ...
                         'gui_LayoutFcn',  [] , ...
                         'gui_Callback',   []);
      if nargin && ischar(varargin{1})
          gui_State.gui_Callback = str2func(varargin{1});
      end

      if nargout
          [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
      else
          gui_mainfcn(gui_State, varargin{:});
      end

      % --- Executes just before ThreePhaseImageSegmentation GUI is made
visible.
      function ThreePhaseImageSegmentation_OpeningFcn(hObject,
eventdata, handles, varargin)

      set(gcf, 'units', 'normalized', 'position', [.5 .5 .45 .6])
      set(handles.axes1,'DataAspectRatio',[2 3 3]);
      set(handles.axes4,'DataAspectRatio',[2 3 3]);
      set(handles.axes2,'DataAspectRatio',[2 3 3]);
      set(handles.axes3,'DataAspectRatio',[2 3 3]);

      % Choose default command line output for
ThreePhaseImageSegmentation
      handles.output = hObject;

      % Update handles structure
      guidata(hObject, handles);

      % --- Outputs from this function are returned to the command line.
      function varargout =
```

```
ThreePhaseImageSegmentation_OutputFcn(hObject, eventdata, handles)

      % Get default command line output from handles structure
      varargout{1} = handles.output;

      % ------------------------------------------------------------------
--



      function LoadImages_Callback(hObject, eventdata, handles)
      d = uigetdir('*.png');
      cd(d);
      q=dir('*.png');
      n = numel(q);

      n=num2str(n);
      set(handles.numImages, 'String', n);

      [filename, filepath] = uigetfile('*.*', 'Please Select First
Cropped Image from Folder that Appears');
      image = imread([filepath filename]);

      %scan the filename and parse the data int a cell string array, C
      C = textscan(filename, '%s %d %s', 'delimiter', '.');

      %bring axes into focus anf show image
      axes(handles.axes1);
      imshow(image, []); %[] = [Imin Imax]

      axes(handles.axes4);
      imshow(image, []);

      %store relevant data in the axes
      setappdata(handles.axes1, 'fileName',  filename);
      setappdata(handles.axes1, 'filePath',  filepath);
      setappdata(handles.axes1, 'image',     image);
      setappdata(handles.axes1, 'ptNum',     C{1});
      setappdata(handles.axes1, 'sliceNum',  C{2});
      setappdata(handles.axes1, 'imageType', C{3});
      guidata(hObject,handles);

      %update slider
      updateSlider(handles);

      axes(handles.axes2);
      sizeofmatrix = size(image,1);
      [Row,Col,r]=MaskPortion(image);
      c=[];
      c1=zeros(sizeofmatrix,sizeofmatrix);
      c1=uint8(c1);
      e=0;
```

```
for i=1:sizeofmatrix
    for j=1:sizeofmatrix
        d=(j - (Row))^2 + (i - (Col))^2;
        if d <= (r)^2
            e=e+1;
            c(e)=image(i,j);
            c1(i,j)=image(i,j);
        end
    end
end

histogram(c);

set(handles.CurrentName, 'String', filename);
% --- Executes on button press in RenameImages.
function RenameImages_Callback(hObject, eventdata, handles)
a = uigetdir;

AA =dir( fullfile(a, '*.bmp') );
fileNames = { AA.name };
for iFile = 1 : numel( AA )
  newName = fullfile(a, sprintf( '1.%02d.bmp', iFile ) );
  movefile( fullfile(a, fileNames{ iFile }), newName );
end

AA =dir( fullfile(a, '*.tif') );
fileNames = { AA.name };
for iFile = 1 : numel( AA )
  newName = fullfile(a, sprintf( '1.%02d.bmp', iFile ) );
  movefile( fullfile(a, fileNames{ iFile }), newName );
end

AA =dir( fullfile(a, '*.jpg') );
fileNames = { AA.name };
for iFile = 1 : numel( AA )
  newName = fullfile(a, sprintf( '1.%02d.bmp', iFile ) );
  movefile( fullfile(a, fileNames{ iFile }), newName );
end

AA =dir( fullfile(a, '*.png') );
fileNames = { AA.name };
for iFile = 1 : numel( AA )
  newName = fullfile(a, sprintf( '1.%02d.bmp', iFile ) );
  movefile( fullfile(a, fileNames{ iFile }), newName );
end

AA =dir( fullfile(a, '*.gif') );
fileNames = { AA.name };
for iFile = 1 : numel( AA )
  newName = fullfile(a, sprintf( '1.%02d.bmp', iFile ) );
  movefile( fullfile(a, fileNames{ iFile }), newName );
end

% --- Executes on button press in CropButton.
```

```
      function CropButton_Callback(hObject, eventdata, handles)
      axes(handles.axes1);

      AA = uigetdir('','Please Select the Folder to Save the Cropped
Images to');
      cd (AA);

      [FileName,PathName] =
uigetfile({'*.jpg;*.tif;*.bmp;*.png;*.gif','All Image Files'},'Please
Select First Renamed Image');

      imagefiles1 = dir([PathName '*.jpg']);
      imagefiles2 = dir([PathName '*.tif']);
      imagefiles3 = dir([PathName '*.bmp']);
      imagefiles4 = dir([PathName '*.png']);
      imagefiles5 = dir([PathName '*.gif']);
      imagefiles = [imagefiles1; imagefiles2; imagefiles3; imagefiles4;
imagefiles5];
      nfiles = length(imagefiles);    % Number of files found

      image = imread([PathName FileName]);

      %%% Following lines to change image matrix into a square sized
matrix

      image=im2uint8(image);
      SIZI=size(image);
      if numel(SIZI)>2
          if SIZI(1,3)>=3
              image=image(:,:,1:3);
              image=rgb2gray(image);
          end
      end

      % Reducing the image matrix into a square sized matrix
          if SIZI(1,1)~=SIZI(1,2)
              SIZim=size(image);
              Divs=round((max(size(image))-min(size(image)))/2);
              if SIZim(1,1)== max(size(image))
                  image=image(Divs+1:SIZim(1,1)-Divs+1,:);
              else
                  image=image(:,Divs+1:SIZim(1,2)-Divs+1);
              end
          end

      % Done

      imshow(image) %needed to use imellipse
      user_defined_ellipse = imellipse(gca, []); % creates user defined
ellipse object.

      MASK = double(user_defined_ellipse.createMask());

      addNewPositionCallback(user_defined_ellipse,@(p)
```

```
title(mat2str(p,3)));

      fcn =
makeConstrainToRectFcn('imellipse',get(gca,'XLim'),get(gca,'YLim'));

      accepted_pos=wait(user_defined_ellipse);% You need to click twice
to continue.
      MASK = double(user_defined_ellipse.createMask());

      for i=1:nfiles

         currentfilename = imagefiles(i).name;
         currentimage = imread([PathName currentfilename]);
         currentimage=im2uint8(currentimage);
         SIZI=size(currentimage);

          if numel(SIZI)>2
              if SIZI(1,3)>=3
                  currentimage=currentimage(:,:,1:3);
                  currentimage=rgb2gray(currentimage);
              end
          end

          if SIZI(1,1)~=SIZI(1,2)
              SIZim=size(currentimage);
              Divs=round((max(size(currentimage))-
min(size(currentimage)))/2);
              if SIZim(1,1)== max(size(currentimage))
                  currentimage=currentimage(Divs+1:SIZim(1,1)-Divs+1,:);
              else
                  currentimage=currentimage(:,Divs+1:SIZim(1,2)-Divs+1);
              end
          end

         new_image_name = [PathName 'CroppedImage_' currentfilename];
         [pathstr,name,ext] = fileparts(new_image_name);
         new_image_name=fullfile(name);
         new_image_name = [new_image_name '.png']; % making the image
.png so it can be transparent
         imwrite(currentimage, new_image_name,'png','Alpha',MASK);

      end

      msg = msgbox('Congratulations! All images have been cropped!');
      waitfor(msg);

      % --- Executes on selection change in Methods.
      function Methods_Callback(hObject, eventdata, handles)
      % hObject    handle to Methods (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)

      % Hints: contents = cellstr(get(hObject,'String')) returns Methods
```

```
contents as cell array
      %        contents{get(hObject,'Value')} returns selected item from
Methods


      % --- Executes during object creation, after setting all
properties.
      function Methods_CreateFcn(hObject, eventdata, handles)
      % hObject    handle to Methods (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    empty - handles not created until after all
CreateFcns called

      % Hint: popupmenu controls usually have a white background on
Windows.
      %       See ISPC and COMPUTER.
      if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
          set(hObject,'BackgroundColor','white');
      end

      % --- Executes on button press in k1Parameter.
      function k1Parameter_Callback(hObject, eventdata, handles)
      prompt = {'Enter k1 parameter value:'};
      dlg_title = 'Input';
      num_lines = 1;
      defaultans = {'0.3'};
      answer = inputdlg(prompt,dlg_title,num_lines,defaultans);

      k1=get(handles.k1Parameter,'String');
      set(handles.k1Parameter,'String',answer);
      k1=char(answer);
      handles.k1 = k1;
      guidata(hObject,handles);

      % --- Executes on button press in k2Parameter.
      function k2Parameter_Callback(hObject, eventdata, handles)
      prompt = {'Enter k2 parameter value:'};
      dlg_title = 'Input';
      num_lines = 1;
      defaultans = {'1.4'};
      answer = inputdlg(prompt,dlg_title,num_lines,defaultans);

      k2=get(handles.k2Parameter,'String');
      set(handles.k2Parameter,'String',answer);
      k2=char(answer);
      handles.k2 = k2;
      guidata(hObject,handles);

      % --- Executes on button press in Run.
      function Run_Callback(hObject, eventdata, handles)
      contents = get(handles.Methods,'String');
      MethodName = contents{get(handles.Methods,'Value')};
```

```
      getappdata(handles.axes1,'fileName');

      number_Of_Air_T=0;
      number_Of_Solid_T=0;
      number_Of_Water_T=0;

       contents = get(handles.Methods,'String');
       MethodName = contents{get(handles.Methods,'Value')};
       getappdata(handles.axes1,'fileName');

      %%% OTSU

      if strcmp(MethodName, 'Otsu ')

set(handles.imageSlider,'Value',get(handles.imageSlider,'Min'));
          d = uigetdir('','Please Select the Folder Containing the
Cropped Images');
          cd(d);
          q=dir('*.png')
          b = numel(q);

          I=imread('CroppedImage_1.01.png');
          [Row,Col,r]=MaskPortion(I);
          sizeofmatrix = size(I,1);
          c=[];
          c1=zeros(sizeofmatrix,sizeofmatrix);
          c1=uint8(c1);
          e=0;

          for i=1:sizeofmatrix
              for j=1:sizeofmatrix
                  d=(j - (Row))^2 + (i - (Col))^2;
                  if d <= (r)^2
                      e=e+1;
                      c(e)=I(i,j);
                      c1(i,j)=I(i,j);
                  end
              end
          end

      e=uint8(c);
      p=imhist(e);

      mean=0;
      for i=1:256
          mean=mean+(i*p(i,1));
      end
      mean=mean/256;
      sumt=zeros(256,1);
      for i=1:256
          meanli=0;
          for j=1:i-1
              meanli=meanli+(j*p(j,1));
          end
```

```
    meanli=meanli/256;
    meangi=0;
    for j=i:256
        meangi=meangi+(j*p(j,1));
    end
    meangi=meangi/256;
    sumli=0;
    sumgi=0;
    for ii=1:i-1
        sumli=sumli+p(ii,1)*((meanli-mean)^2);
    end
    for ii=i:256
        sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
    end

    sumt(i,1)=sumgi+sumli;

end

Threshold1=find(sumt==max(sumt));

% Following lines of code are used to determine Threshold 2

pmax = p(Threshold1,1);
p2 = p(1:Threshold1);

mean=0;
for i=1:Threshold1
    mean=mean+(i*p2(i,1));
end
mean=mean/256;
sumt=zeros(256,1);
for i=1:Threshold1
    meanli=0;
    for j=1:Threshold1-1
        meanli=meanli+(j*p2(j,1));
    end
    meanli=meanli/256;
    meangi=0;
    for j=i:Threshold1
        meangi=meangi+(j*p2(j,1));
    end
    meangi=meangi/256;
    sumli=0;
    sumgi=0;
    for ii=1:Threshold1-1
        sumli=sumli+p2(ii,1)*((meanli-mean)^2);
    end
    for ii=i:Threshold1
        sumgi=sumgi+p2(ii,1)*((meangi-mean)^2);
    end
    sumt(i,1)=sumgi+sumli;

end
```

```
Threshold2=find(sumt==max(sumt));

c2=zeros(sizeofmatrix,sizeofmatrix);
c2=uint8(c2);
for i=1:sizeofmatrix
    for j=1:sizeofmatrix
        d=(j - (Row))^2 + (i - (Col))^2;
        if d <= (r)^2
            if c1(i,j)>Threshold1
                c2(i,j)=100;
            elseif c1(i,j) < Threshold2
                c2(i,j)=20;
            else
                c2(i,j) = 60;
            end

        end
    end
end

axes(handles.axes3);
imshow(c2,[]);

for ka = 1: b
   I = sprintf('CroppedImage_1.%02d.png', ka);
   I=imread(I);
   [Row,Col,r]=MaskPortion(I);
sizeofmatrix = size(I,1);

c=[];
c1=zeros(sizeofmatrix,sizeofmatrix);
c1=uint8(c1);
e=0;
for i=1:sizeofmatrix
    for j=1:sizeofmatrix
        d=(j - (Row))^2 + (i - (Col))^2;
        if d <= (r)^2
            e=e+1;
            c(e)=I(i,j);
            c1(i,j)=I(i,j);
        end
    end
end

% Automatic Threshold using Otsu's Method

e=uint8(c);
p=imhist(e);

mean=0;
for i=1:256
    mean=mean+(i*p(i,1));
end
```

```
mean=mean/256;
sumt=zeros(256,1);
for i=1:256
    meanli=0;
    for j=1:i-1
        meanli=meanli+(j*p(j,1));
    end
    meanli=meanli/256;
    meangi=0;
    for j=i:256
        meangi=meangi+(j*p(j,1));
    end
    meangi=meangi/256;
    sumli=0;
    sumgi=0;
    for ii=1:i-1
        sumli=sumli+p(ii,1)*((meanli-mean)^2);
    end

    for ii=i:256
        sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
    end

    sumt(i,1)=sumgi+sumli;

end
Threshold1=find(sumt==max(sumt));

% Following lines of code are used to determine Threshold 2

pmax = p(Threshold1,1);
p2 = p(1:Threshold1);

mean=0;
for i=1:Threshold1
    mean=mean+(i*p2(i,1));
end
mean=mean/256;
sumt=zeros(256,1);
for i=1:Threshold1
    meanli=0;
    for j=1:Threshold1-1
        meanli=meanli+(j*p2(j,1));
    end
    meanli=meanli/256;
    meangi=0;
    for j=i:Threshold1
        meangi=meangi+(j*p2(j,1));
    end
    meangi=meangi/256;
    sumli=0;
    sumgi=0;
    for ii=1:Threshold1-1
        sumli=sumli+p2(ii,1)*((meanli-mean)^2);
```

```
        end

        for ii=i:Threshold1
            sumgi=sumgi+p2(ii,1)*((meangi-mean)^2);
        end

        sumt(i,1)=sumgi+sumli;

    end

    Threshold2=find(sumt==max(sumt));


    %%
    c2=zeros(sizeofmatrix,sizeofmatrix);
    c2=uint8(c2);
    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
            d=(j - (Row))^2 + (i - (Col))^2;
            if d <= (r)^2
                if c1(i,j)>Threshold1
                    c2(i,j)=100;
                elseif c1(i,j) < Threshold2
                    c2(i,j)=20;
                else
                    c2(i,j) = 60;
                end

            end
        end
    end


    number_Of_Air_Pixels=sum(sum(c2==20));
    number_Of_Solid_Pixels=sum(sum(c2==100));
    number_Of_Water_Pixels=sum(sum(c2==60));

    number_Of_Air_T=number_Of_Air_T+number_Of_Air_Pixels;
    number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;
    number_Of_Water_T=number_Of_Water_T+number_Of_Water_Pixels;


    end
    void_ratio =
(number_Of_Air_T+number_Of_Water_T)/number_Of_Solid_T;
    void_ratio=round(void_ratio,2);

    %%

    v=void_ratio;
    v=num2str(v);

    set(handles.VoidText, 'String', v);
          end
```

```
      %%% ITERATIVE OTSU

      number_Of_Air_T=0;
      number_Of_Solid_T=0;
      number_Of_Water_T=0;


      if strcmp(MethodName, 'Iterative Otsu')

set(handles.imageSlider,'Value',get(handles.imageSlider,'Min'));
          d = uigetdir('','Please Select the Folder Containing the
Cropped Images');
          cd(d);
          q=dir('*.png');
          b = numel(q);

          I=imread('CroppedImage_1.01.png');
          [Row,Col,r]=MaskPortion(I);
          sizeofmatrix = size(I,1);
          c=[];
          c1=zeros(sizeofmatrix,sizeofmatrix);
          c1=uint8(c1);
          e=0;

          for i=1:sizeofmatrix
              for j=1:sizeofmatrix
                  d=(j - (Row))^2 + (i - (Col))^2;
                  if d <= (r)^2
                      e=e+1;
                      c(e)=I(i,j);
                      c1(i,j)=I(i,j);
                  end
              end
          end

      e=uint8(c);
      p=imhist(e);
      mean=0;
      for i=1:256
          mean=mean+(i*p(i,1));
      end
      mean=mean/256;
      sumt=zeros(256,1);
      for i=1:256
          meanli=0;
          for j=1:i-1
              meanli=meanli+(j*p(j,1));
          end
          meanli=meanli/256;
          meangi=0;
          for j=i:256
              meangi=meangi+(j*p(j,1));
          end
```

```
            meangi=meangi/256;
            sumli=0;
            sumgi=0;
            for ii=1:i-1
                sumli=sumli+p(ii,1)*((meanli-mean)^2);
            end

            for ii=i:256
                sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
            end

            sumt(i,1)=sumgi+sumli;

        end

        Threshold1=find(sumt==max(sumt));

        [counts,N]=imhist(e);
        counts1=counts(1:Threshold1);
        mu1=cumsum(counts1);   % To get the total summation of intensity
values
        N1 = (0:Threshold1-1)';
        mean=(sum(N1.*counts1))/mu1(end);
        mean=round(mean);
        IntensityRegion1=mean;

        counts2=counts(Threshold1+1:end);
        mu2=cumsum(counts2);   % To get the total summation of intensity
values
        N2 = (Threshold1:255)';
        mean=(sum(N2.*counts2))/mu2(end);
        mean=round(mean);
        IntensityRegion2=mean;

        Threshold2 = (IntensityRegion1+IntensityRegion2)/2;
        Threshold2=round(Threshold2);

        % Iterative Process to Find the Ideal Value for Threshold2

        [counts,N]=imhist(e);
        counts3=counts(1:Threshold2);
        mu1=cumsum(counts3);   % To get the total summation of intensity
values
        N1 = (0:Threshold2-1)';
        mean=(sum(N1.*counts3))/mu1(end);
        mean=round(mean);
        IntensityRegion1=mean;

        counts4=counts(Threshold2+1:end);
        mu2=cumsum(counts4);   % To get the total summation of intensity
values
        N2 = (Threshold2:255)';
        mean=(sum(N2.*counts4))/mu2(end);
        mean=round(mean);
```

```
      IntensityRegion2=mean;

      Thresholdnew = (IntensityRegion1+IntensityRegion2)/2;
      Thresholdnew=round(Thresholdnew);

      % Check if Thresholdnew is an Acceptable Choice for Threshold2

      loopCounter=1;
      while Thresholdnew-5 < Threshold2 && Thresholdnew+5 < Threshold2

          Threshold2=Thresholdnew;

      [counts,N]=imhist(e);
      counts5=counts(1:Threshold2);
      mu1=cumsum(counts5);   % To get the total summation of intensity
values
      N1 = (0:Threshold2-1)';
      mean=(sum(N1.*counts5))/mu1(end);
      mean=round(mean);
      IntensityRegion1=mean;

      counts6=counts(Threshold2+1:end);
      mu2=cumsum(counts6);   % To get the total summation of intensity
values
      N2 = (Threshold2:255)';
      mean=(sum(N2.*counts6))/mu2(end);
      mean=round(mean);
      IntensityRegion2=mean;

      Thresholdnew = (IntensityRegion1+IntensityRegion2)/2;
      Thresholdnew=round(Thresholdnew);

      if loopCounter>=10
          break;
      end

      loopCounter=loopCounter+1;

      end

      Threshold2=Thresholdnew;

      c2=zeros(sizeofmatrix,sizeofmatrix);
      c2=uint8(c2);
      for i=1:sizeofmatrix
          for j=1:sizeofmatrix
              d=(j - (Row))^2 + (i - (Col))^2;
              if d <= (r)^2
                  if c1(i,j)>Threshold1
                      c2(i,j)=100;
                  elseif c1(i,j) < Threshold2
                      c2(i,j)=20;
                  else
                      c2(i,j) = 60;
```

```
            end

        end
    end
end
axes(handles.axes3);
imshow(c2,[]);

for ka = 1: b
    I = sprintf('CroppedImage_1.%02d.png', ka);
    I=imread(I);
    [Row,Col,r]=MaskPortion(I);
    sizeofmatrix = size(I,1);

    c=[];
    c1=zeros(sizeofmatrix,sizeofmatrix);
    c1=uint8(c1);
    e=0;

    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
            d=(j - (Row))^2 + (i - (Col))^2;
            if d <= (r)^2
                e=e+1;
                c(e)=I(i,j);
                c1(i,j)=I(i,j);
            end
        end
    end

e=uint8(c);
p=imhist(e);
mean=0;
for i=1:256
    mean=mean+(i*p(i,1));
end
mean=mean/256;
sumt=zeros(256,1);
for i=1:256
    meanli=0;
    for j=1:i-1
        meanli=meanli+(j*p(j,1));
    end
    meanli=meanli/256;
    meangi=0;
    for j=i:256
        meangi=meangi+(j*p(j,1));
    end
    meangi=meangi/256;
    sumli=0;
    sumgi=0;
    for ii=1:i-1
        sumli=sumli+p(ii,1)*((meanli-mean)^2);
    end
```

```
          for ii=i:256
              sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
          end

          sumt(i,1)=sumgi+sumli;

      end

      Threshold1=find(sumt==max(sumt));

      [counts,N]=imhist(e);
      counts1=counts(1:Threshold1);
      mu1=cumsum(counts1);   % To get the total summation of intensity
values
      N1 = (0:Threshold1-1)';
      mean=(sum(N1.*counts1))/mu1(end);
      mean=round(mean);
      IntensityRegion1=mean;

      counts2=counts(Threshold1+1:end);
      mu2=cumsum(counts2);   % To get the total summation of intensity
values
      N2 = (Threshold1:255)';
      mean=(sum(N2.*counts2))/mu2(end);
      mean=round(mean);
      IntensityRegion2=mean;

      Threshold2 = (IntensityRegion1+IntensityRegion2)/2;
      Threshold2=round(Threshold2);

      % Iterative Process to Find the Ideal Value for Threshold2

      [counts,N]=imhist(e);
      counts3=counts(1:Threshold2);
      mu1=cumsum(counts3);   % To get the total summation of intensity
values
      N1 = (0:Threshold2-1)';
      mean=(sum(N1.*counts3))/mu1(end);
      mean=round(mean);
      IntensityRegion1=mean;

      counts4=counts(Threshold2+1:end);
      mu2=cumsum(counts4);   % To get the total summation of intensity
values
      N2 = (Threshold2:255)';
      mean=(sum(N2.*counts4))/mu2(end);
      mean=round(mean);
      IntensityRegion2=mean;

      Thresholdnew = (IntensityRegion1+IntensityRegion2)/2;
      Thresholdnew=round(Thresholdnew);


      % Check if Thresholdnew is an Acceptable Choice for Threshold2
```

```
      loopCounter=1;

      while Thresholdnew-5 < Threshold2 && Thresholdnew+5 < Threshold2

          Threshold2=Thresholdnew;
          [counts,N]=imhist(e);
          counts5=counts(1:Threshold2);
          mu1=cumsum(counts5);   % To get the total summation of
intensity values
          N1 = (0:Threshold2-1)';
          mean=(sum(N1.*counts5))/mu1(end);
          mean=round(mean);
          IntensityRegion1=mean;

          counts6=counts(Threshold2+1:end);
          mu2=cumsum(counts6);   % To get the total summation of
intensity values
          N2 = (Threshold2:255)';
          mean=(sum(N2.*counts6))/mu2(end);
          mean=round(mean);

          IntensityRegion2=mean;

          Thresholdnew = (IntensityRegion1+IntensityRegion2)/2;
          Thresholdnew=round(Thresholdnew);

          if loopCounter>=10
              break;
          end

          loopCounter=loopCounter+1;

      end

      Threshold2=Thresholdnew;

      c2=zeros(sizeofmatrix,sizeofmatrix);
      c2=uint8(c2);
      for i=1:sizeofmatrix
          for j=1:sizeofmatrix
              d=(j - (Row))^2 + (i - (Col))^2;
              if d <= (r)^2
                  if c1(i,j)>Threshold1
                      c2(i,j)=100;
                  elseif c1(i,j) < Threshold2
                      c2(i,j)=20;
                  else
                      c2(i,j) = 60;
                  end

              end
          end
      end
```

```
      number_Of_Air_Pixels=sum(sum(c2==20));
      number_Of_Solid_Pixels=sum(sum(c2==100));
      number_Of_Water_Pixels=sum(sum(c2==60));

      number_Of_Air_T=number_Of_Air_T+number_Of_Air_Pixels;
      number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;
      number_Of_Water_T=number_Of_Water_T+number_Of_Water_Pixels;

      end

      void_ratio =
(number_Of_Air_T+number_Of_Water_T)/number_Of_Solid_T;
      void_ratio=round(void_ratio,2);

      v=void_ratio;
      v=num2str(v);

      set(handles.VoidText, 'String', v);
      end

      %%% REFINED STATISTICAL-BASED

      number_Of_Air_T=0;
      number_Of_Solid_T=0;
      number_Of_Water_T=0;

      if strcmp(MethodName, 'Refined Statistical-Based')
          k = warndlg('Make sure to choose values for free/fitting
parameters k1 and k2!','Warning');
          waitfor(k);
          k1= handles.k1;
          k1=str2double(k1);
          k2= handles.k2;
          k2=str2double(k2);

set(handles.imageSlider,'Value',get(handles.imageSlider,'Min'));

          d = uigetdir('','Please Select the Folder Containing the
Cropped Images');
          cd(d);
          q=dir('*.png');
          b = numel(q);

          I=imread('CroppedImage_1.01.png');
          [Row,Col,r]=MaskPortion(I);
          sizeofmatrix = size(I,1);
          c=[];
          c1=zeros(sizeofmatrix,sizeofmatrix);
          c1=uint8(c1);
          e=0;
          for i=1:sizeofmatrix
              for j=1:sizeofmatrix
                  d=(j - (Row))^2 + (i - (Col))^2;
```

```
                if d <= (r)^2
                    e=e+1;
                    c(e)=I(i,j);
                    c1(i,j)=I(i,j);
                end
            end
        end

        e=uint8(c);


    % Automatic Threshold using Refined Statistical-based Method

    [counts,N]=imhist(e);
    mu=cumsum(counts);   % To get the total summation of intensity
values
    mean=(sum(N.*counts))/mu(end);
    mean=round(mean);
    variance=sum((counts-mean).^2)/(mu-1);
    variance=sum(variance);
    standarddeviation=sqrt(variance);
    standarddeviation=round(standarddeviation);

    T1=mean-(k1*standarddeviation);
    T2=mean+(k2*standarddeviation);

    Threshold1=abs(T1+T2)/2;
    Threshold1=round(Threshold1);

    Threshold2=abs(T1-T2)/2;
    Threshold2=round(Threshold2);

    c2=zeros(sizeofmatrix,sizeofmatrix);
    c2=uint8(c2);
    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
            d=(j - (Row))^2 + (i - (Col))^2;
            if d <= (r)^2
                if c1(i,j)>Threshold1
                    c2(i,j)=100;
                elseif c1(i,j) < Threshold2
                    c2(i,j)=20;
                else
                    c2(i,j) = 60;
                end

            end
        end
    end

    axes(handles.axes3);
    imshow(c2,[]);

    for ka = 1: b
```

```
        I = sprintf('CroppedImage_1.%02d.png', ka);
        I=imread(I);
       [Row,Col,r]=MaskPortion(I);
        sizeofmatrix = size(I,1);

        c=[];
        c1=zeros(sizeofmatrix,sizeofmatrix);
        c1=uint8(c1);
        e=0;
        for i=1:sizeofmatrix
            for j=1:sizeofmatrix
                d=(j - (Row))^2 + (i - (Col))^2;
                if d <= (r)^2
                e=e+1;
                c(e)=I(i,j);
                c1(i,j)=I(i,j);
                end
            end
        end

        e=uint8(c);

    % Automatic Threshold using Refined Statistical-based method

    [counts,N]=imhist(e);
    mu=cumsum(counts);   % To get the total summation of intensity
values
    mean=(sum(N.*counts))/mu(end);
    mean=round(mean);
    variance=sum((counts-mean).^2)/(mu-1);
    variance=sum(variance);
    standarddeviation=sqrt(variance);
    standarddeviation=round(standarddeviation);

    T1=mean-(k1*standarddeviation);
    T2=mean+(k2*standarddeviation);

    Threshold1=abs(T1+T2)/2;
    Threshold1=round(Threshold1);

    Threshold2=abs(T1-T2)/2;
    Threshold2=round(Threshold2);

    c2=zeros(sizeofmatrix,sizeofmatrix);
    c2=uint8(c2);
    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
            d=(j - (Row))^2 + (i - (Col))^2;
            if d <= (r)^2
                if c1(i,j)>Threshold1
                    c2(i,j)=100;
                elseif c1(i,j) < Threshold2
                    c2(i,j)=20;
                else
```

```
                    c2(i,j) = 60;
                end

            end
        end
    end

    number_Of_Air_Pixels=sum(sum(c2==20));
    number_Of_Solid_Pixels=sum(sum(c2==100));
    number_Of_Water_Pixels=sum(sum(c2==60));

    number_Of_Air_T=number_Of_Air_T+number_Of_Air_Pixels;
    number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;
    number_Of_Water_T=number_Of_Water_T+number_Of_Water_Pixels;

    end

    void_ratio =
(number_Of_Air_T+number_Of_Water_T)/number_Of_Solid_T;
    void_ratio=round(void_ratio,2);

    v=void_ratio;
    v=num2str(v);

    set(handles.VoidText, 'String', v);
    end

    %%% OTSU

    number_Of_Air_T=0;
    number_Of_Solid_T=0;
    number_Of_Water_T=0;

    contents = get(handles.Methods,'String');
    MethodName = contents{get(handles.Methods,'Value')};

    getappdata(handles.axes1,'fileName');

    if strcmp(MethodName, 'Otsu ')
        for ka = 1: b
            I = sprintf('CroppedImage_1.%02d.png', ka);
            I=imread(I);
            [Row,Col,r]=MaskPortion(I);
            sizeofmatrix = size(I,1);
            c=[];
            c1=zeros(sizeofmatrix,sizeofmatrix);
            c1=uint8(c1);
            e=0;
            for i=1:sizeofmatrix
                for j=1:sizeofmatrix
                    d=(j - (Row))^2 + (i - (Col))^2;
                    if d <= (r)^2
                        e=e+1;
                        c(e)=I(i,j);
```

```
                    c1(i,j)=I(i,j);
                end
            end
        end

% Automatic Threshold using Otsu's Method

e=uint8(c);
p=imhist(e);

mean=0;
for i=1:256
    mean=mean+(i*p(i,1));
end
mean=mean/256;
sumt=zeros(256,1);

for i=1:256
    meanli=0;
    for j=1:i-1
        meanli=meanli+(j*p(j,1));
    end
    meanli=meanli/256;
    meangi=0;
    for j=i:256
        meangi=meangi+(j*p(j,1));
    end
    meangi=meangi/256;
    sumli=0;
    sumgi=0;
    for ii=1:i-1
        sumli=sumli+p(ii,1)*((meanli-mean)^2);
    end
    for ii=i:256
        sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
    end

    sumt(i,1)=sumgi+sumli;

end

Threshold1=find(sumt==max(sumt));

% Following lines of code are used to determine Threshold 2

pmax = p(Threshold1,1);
p2 = p(1:Threshold1);
mean=0;

for i=1:Threshold1
    mean=mean+(i*p2(i,1));
end

mean=mean/256;
```

```
sumt=zeros(256,1);

for i=1:Threshold1
    meanli=0;
    for j=1:Threshold1-1
        meanli=meanli+(j*p2(j,1));
    end
    meanli=meanli/256;
    meangi=0;
    for j=i:Threshold1
        meangi=meangi+(j*p2(j,1));
    end
    meangi=meangi/256;
    sumli=0;
    sumgi=0;
    for ii=1:Threshold1-1
        sumli=sumli+p2(ii,1)*((meanli-mean)^2);
    end
    for ii=i:Threshold1
        sumgi=sumgi+p2(ii,1)*((meangi-mean)^2);
    end

    sumt(i,1)=sumgi+sumli;

end

Threshold2=find(sumt==max(sumt));

c2=zeros(sizeofmatrix,sizeofmatrix);
c2=uint8(c2);
for i=1:sizeofmatrix
    for j=1:sizeofmatrix
        d=(j - (Row))^2 + (i - (Col))^2;
        if d <= (r)^2
            if c1(i,j)>Threshold1
                c2(i,j)=100;
            elseif c1(i,j) < Threshold2
                c2(i,j)=20;
            else
                c2(i,j) = 60;
            end

        end
    end
end

number_Of_Air_Pixels=sum(sum(c2==20));
number_Of_Solid_Pixels=sum(sum(c2==100));
number_Of_Water_Pixels=sum(sum(c2==60));

number_Of_Air_T=number_Of_Air_T+number_Of_Air_Pixels;
number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;
number_Of_Water_T=number_Of_Water_T+number_Of_Water_Pixels;
```

```
      degree_of_saturation =
(number_Of_Water_T/(number_Of_Air_T+number_Of_Water_T))*100;
          end

      p=degree_of_saturation;
      p=round(p,2);
      p=num2str(p);
      set(handles.SatText, 'String', p);
      end

      %%% ITERATIVE OTSU

      if strcmp(MethodName, 'Iterative Otsu')

set(handles.imageSlider,'Value',get(handles.imageSlider,'Min'));
          for ka = 1: b
              I = sprintf('CroppedImage_1.%02d.png', ka);
              I=imread(I);
              [Row,Col,r]=MaskPortion(I);
              sizeofmatrix = size(I,1);
              c=[];
              c1=zeros(sizeofmatrix,sizeofmatrix);
              c1=uint8(c1);
              e=0;

              for i=1:sizeofmatrix
                  for j=1:sizeofmatrix
                      d=(j - (Row))^2 + (i - (Col))^2;
                      if d <= (r)^2
                          e=e+1;
                          c(e)=I(i,j);
                          c1(i,j)=I(i,j);
                      end
                  end
              end

      e=uint8(c);
      p=imhist(e);
      mean=0;
      for i=1:256
          mean=mean+(i*p(i,1));
      end
      mean=mean/256;
      sumt=zeros(256,1);
      for i=1:256
          meanli=0;
          for j=1:i-1
              meanli=meanli+(j*p(j,1));
          end
          meanli=meanli/256;
          meangi=0;
          for j=i:256
              meangi=meangi+(j*p(j,1));
          end
```

```
          meangi=meangi/256;
          sumli=0;
          sumgi=0;
          for ii=1:i-1
              sumli=sumli+p(ii,1)*((meanli-mean)^2);
          end
          for ii=i:256
              sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
          end
          sumt(i,1)=sumgi+sumli;
      end

      Threshold1=find(sumt==max(sumt));

      [counts,N]=imhist(e);
      counts1=counts(1:Threshold1);
      mu1=cumsum(counts1);   % To get the total summation of intensity
values
      N1 = (0:Threshold1-1)';
      mean=(sum(N1.*counts1))/mu1(end);
      mean=round(mean);
      IntensityRegion1=mean;

      counts2=counts(Threshold1+1:end);
      mu2=cumsum(counts2);   % To get the total summation of intensity
values
      N2 = (Threshold1:255)';
      mean=(sum(N2.*counts2))/mu2(end);
      mean=round(mean);
      IntensityRegion2=mean;

      Threshold2 = (IntensityRegion1+IntensityRegion2)/2;
      Threshold2=round(Threshold2);

      % Iterative Process to Find the Ideal Value for Threshold2

      [counts,N]=imhist(e);
      counts3=counts(1:Threshold2);
      mu1=cumsum(counts3);   % To get the total summation of intensity
values
      N1 = (0:Threshold2-1)';
      mean=(sum(N1.*counts3))/mu1(end);
      mean=round(mean);
      IntensityRegion1=mean;

      counts4=counts(Threshold2+1:end);
      mu2=cumsum(counts4);   % To get the total summation of intensity
values
      N2 = (Threshold2:255)';
      mean=(sum(N2.*counts4))/mu2(end);
      mean=round(mean);
      IntensityRegion2=mean;

      Thresholdnew = (IntensityRegion1+IntensityRegion2)/2;
```

```
      Thresholdnew=round(Thresholdnew);


      % Check if Thresholdnew is an Acceptable Choice for Threshold2

      loopCounter=1;

      while Thresholdnew-5 < Threshold2 && Thresholdnew+5 < Threshold2
          Threshold2=Thresholdnew;
          [counts,N]=imhist(e);
          counts5=counts(1:Threshold2);
          mu1=cumsum(counts5);   % To get the total summation of
intensity values
          N1 = (0:Threshold2-1)';
          mean=(sum(N1.*counts5))/mu1(end);
          mean=round(mean);
          IntensityRegion1=mean;
          counts6=counts(Threshold2+1:end);
          mu2=cumsum(counts6);   % To get the total summation of
intensity values
          N2 = (Threshold2:255)';
          mean=(sum(N2.*counts6))/mu2(end);
          mean=round(mean);
          IntensityRegion2=mean;

          Thresholdnew = (IntensityRegion1+IntensityRegion2)/2;
          Thresholdnew=round(Thresholdnew);

          if loopCounter>=10
              break;
          end

          loopCounter=loopCounter+1;
      end

      Threshold2=Thresholdnew;

      c2=zeros(sizeofmatrix,sizeofmatrix);
      c2=uint8(c2);
      for i=1:sizeofmatrix
          for j=1:sizeofmatrix
              d=(j - (Row))^2 + (i - (Col))^2;
              if d <= (r)^2
                  if c1(i,j)>Threshold1
                      c2(i,j)=100;
                  elseif c1(i,j) < Threshold2
                      c2(i,j)=20;
                  else
                      c2(i,j) = 60;
                  end

              end
          end
      end
```

```
      number_Of_Air_Pixels=sum(sum(c2==20));
      number_Of_Solid_Pixels=sum(sum(c2==100));
      number_Of_Water_Pixels=sum(sum(c2==60));

      number_Of_Air_T=number_Of_Air_T+number_Of_Air_Pixels;
      number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;
      number_Of_Water_T=number_Of_Water_T+number_Of_Water_Pixels;

      degree_of_saturation =
(number_Of_Water_T/(number_Of_Air_T+number_Of_Water_T))*100;

          end

      p=degree_of_saturation;
      p=round(p,2);
      p=num2str(p);
      set(handles.SatText, 'String', p);
      end

      %%% REFINED STATISTICAL-BASED

      number_Of_Air_T=0;
      number_Of_Solid_T=0;
      number_Of_Water_T=0;

      if strcmp(MethodName, 'Refined Statistical-Based')
          k1= handles.k1;
          k1=str2double(k1);
          k2= handles.k2;
          k2=str2double(k2);

set(handles.imageSlider,'Value',get(handles.imageSlider,'Min'));

          for ka = 1: b
              I = sprintf('CroppedImage_1.%02d.png', ka);
              I=imread(I);
              [Row,Col,r]=MaskPortion(I);
              sizeofmatrix = size(I,1);
              c=[];
              c1=zeros(sizeofmatrix,sizeofmatrix);
              c1=uint8(c1);
              e=0;
              for i=1:sizeofmatrix
                  for j=1:sizeofmatrix
                      d=(j - (Row))^2 + (i - (Col))^2;
                      if d <= (r)^2
                          e=e+1;
                          c(e)=I(i,j);
                          c1(i,j)=I(i,j);
                      end
                  end
              end
```

```
            e=uint8(c);


      % Automatic Threshold using Refined Statistical-Based Method

      [counts,N]=imhist(e);
      mu=cumsum(counts);   % To get the total summation of intensity
values
      mean=(sum(N.*counts))/mu(end);
      mean=round(mean);
      variance=sum((counts-mean).^2)/(mu-1);
      variance=sum(variance);
      standarddeviation=sqrt(variance);
      standarddeviation=round(standarddeviation);

      T1=mean-(k1*standarddeviation);
      T2=mean+(k2*standarddeviation);

      Threshold1=abs(T1+T2)/2;
      Threshold1=round(Threshold1);

      Threshold2=abs(T1-T2)/2;
      Threshold2=round(Threshold2);

      c2=zeros(sizeofmatrix,sizeofmatrix);
      c2=uint8(c2);

      for i=1:sizeofmatrix
          for j=1:sizeofmatrix
              d=(j - (Row))^2 + (i - (Col))^2;
              if d <= (r)^2
                  if c1(i,j)>Threshold1
                      c2(i,j)=100;
                  elseif c1(i,j) < Threshold2
                      c2(i,j)=20;
                  else
                      c2(i,j) = 60;
                  end
              end
          end
      end

      number_Of_Air_Pixels=sum(sum(c2==20));
      number_Of_Solid_Pixels=sum(sum(c2==100));
      number_Of_Water_Pixels=sum(sum(c2==60));

      number_Of_Air_T=number_Of_Air_T+number_Of_Air_Pixels;
      number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;
      number_Of_Water_T=number_Of_Water_T+number_Of_Water_Pixels;

          end

      degree_of_saturation =
(number_Of_Water_T/(number_Of_Air_T+number_Of_Water_T))*100;
```

```
p=degree_of_saturation;
p=round(p,2);
p=num2str(p);
set(handles.SatText, 'String', p);
end

%%% OTSU

number_Of_Air_T=0;
number_Of_Solid_T=0;
number_Of_Water_T=0;

contents = get(handles.Methods,'String');
MethodName = contents{get(handles.Methods,'Value')};
getappdata(handles.axes1,'fileName');

if strcmp(MethodName, 'Otsu ')
    for ka = 1: b
        I = sprintf('CroppedImage_1.%02d.png', ka);
        I=imread(I);
        [Row,Col,r]=MaskPortion(I);
        sizeofmatrix = size(I,1);
        c=[];
        c1=zeros(sizeofmatrix,sizeofmatrix);
        c1=uint8(c1);
        e=0;

        for i=1:sizeofmatrix
            for j=1:sizeofmatrix
                d=(j - (Row))^2 + (i - (Col))^2;
                if d <= (r)^2
                    e=e+1;
                    c(e)=I(i,j);
                    c1(i,j)=I(i,j);
                end
            end
        end

        e=uint8(c);
        p=imhist(e);
        mean=0;

        for i=1:256
            mean=mean+(i*p(i,1));
        end

        mean=mean/256;
        sumt=zeros(256,1);

        for i=1:256
            meanli=0;
            for j=1:i-1
                meanli=meanli+(j*p(j,1));
```

```
            end
            meanli=meanli/256;
            meangi=0;
            for j=i:256
                meangi=meangi+(j*p(j,1));
            end
            meangi=meangi/256;
            sumli=0;
            sumgi=0;
            for ii=1:i-1
                sumli=sumli+p(ii,1)*((meanli-mean)^2);
            end

            for ii=i:256
                sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
            end

            sumt(i,1)=sumgi+sumli;
        end

Threshold1(ka,1)=find(sumt==max(sumt));

    end

Threshold1=sum(Threshold1(1:b));
Threshold1=Threshold1/b;
Threshold1=round(Threshold1);
AT1=Threshold1;
AT1=num2str(AT1);
set(handles.AT1Box, 'String', AT1);
end

%%% OTSU

number_Of_Air_T=0;
number_Of_Solid_T=0;
number_Of_Water_T=0;

contents = get(handles.Methods,'String');
MethodName = contents{get(handles.Methods,'Value')};
getappdata(handles.axes1,'fileName');

if strcmp(MethodName, 'Otsu ')
    for ka = 1: b
        I = sprintf('CroppedImage_1.%02d.png', ka);
        I=imread(I);
        [Row,Col,r]=MaskPortion(I);
        sizeofmatrix = size(I,1);
        c=[];
        c1=zeros(sizeofmatrix,sizeofmatrix);
        c1=uint8(c1);
        e=0;
        for i=1:sizeofmatrix
            for j=1:sizeofmatrix
```

```
                d=(j - (Row))^2 + (i - (Col))^2;
                if d <= (r)^2
                    e=e+1;
                    c(e)=I(i,j);
                    c1(i,j)=I(i,j);
                end
            end
        end

e=uint8(c);
p=imhist(e);

mean=0;

for i=1:256
    mean=mean+(i*p(i,1));
end
mean=mean/256;
sumt=zeros(256,1);
for i=1:256
    meanli=0;
    for j=1:i-1
        meanli=meanli+(j*p(j,1));
    end
    meanli=meanli/256;
    meangi=0;
    for j=i:256
        meangi=meangi+(j*p(j,1));
    end
    meangi=meangi/256;
    sumli=0;
    sumgi=0;
    for ii=1:i-1
        sumli=sumli+p(ii,1)*((meanli-mean)^2);
    end
    for ii=i:256
        sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
    end

    sumt(i,1)=sumgi+sumli;
end

Threshold1=find(sumt==max(sumt));

% Following lines of code are used to determine Threshold 2

pmax = p(Threshold1,1);
p2 = p(1:Threshold1);
mean=0;

for i=1:Threshold1
    mean=mean+(i*p2(i,1));
end
```

```
      mean=mean/256;
      sumt=zeros(256,1);

      for i=1:Threshold1
          meanli=0;
          for j=1:Threshold1-1
              meanli=meanli+(j*p2(j,1));
          end
          meanli=meanli/256;
          meangi=0;
          for j=i:Threshold1
              meangi=meangi+(j*p2(j,1));
          end
          meangi=meangi/256;
          sumli=0;
          sumgi=0;
          for ii=1:Threshold1-1
              sumli=sumli+p2(ii,1)*((meanli-mean)^2);
          end
          for ii=i:Threshold1
              sumgi=sumgi+p2(ii,1)*((meangi-mean)^2);
          end

          sumt(i,1)=sumgi+sumli;

      end

      Threshold2(ka,1)=find(sumt==max(sumt));

      end

      Threshold2=sum(Threshold2(1:b));
      Threshold2=Threshold2/b;
      Threshold2=round(Threshold2);
      AT2=Threshold2;
      AT2=num2str(AT2);
      set(handles.AT2Box, 'String', AT2);
      end

      %%% ITERATIVE OTSU

      number_Of_Air_T=0;
      number_Of_Solid_T=0;
      number_Of_Water_T=0;

      if strcmp(MethodName, 'Iterative Otsu')

set(handles.imageSlider,'Value',get(handles.imageSlider,'Min'));
          for ka = 1: b
              I = sprintf('CroppedImage_1.%02d.png', ka);
              I=imread(I);
              [Row,Col,r]=MaskPortion(I);
              sizeofmatrix = size(I,1);
              c=[];
```

```
        c1=zeros(sizeofmatrix,sizeofmatrix);
        c1=uint8(c1);
        e=0;

        for i=1:sizeofmatrix
            for j=1:sizeofmatrix
                d=(j - (Row))^2 + (i - (Col))^2;
                if d <= (r)^2
                    e=e+1;
                    c(e)=I(i,j);
                    c1(i,j)=I(i,j);
                end
            end
        end

e=uint8(c);
p=imhist(e);
mean=0;

for i=1:256
    mean=mean+(i*p(i,1));
end
mean=mean/256;
sumt=zeros(256,1);
for i=1:256
    meanli=0;
    for j=1:i-1
        meanli=meanli+(j*p(j,1));
    end
    meanli=meanli/256;
    meangi=0;
    for j=i:256
        meangi=meangi+(j*p(j,1));
    end
    meangi=meangi/256;
    sumli=0;
    sumgi=0;
    for ii=1:i-1
        sumli=sumli+p(ii,1)*((meanli-mean)^2);
    end
    for ii=i:256
        sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
    end
    sumt(i,1)=sumgi+sumli;

end

Threshold1(ka,1)=find(sumt==max(sumt));

    end

Threshold1=sum(Threshold1(1:b));
Threshold1=Threshold1/b;
Threshold1=round(Threshold1);
```

```
      AT1=Threshold1;
      AT1=num2str(AT1);
      set(handles.AT1Box, 'String', AT1);

      end

      %%% ITERATIVE OTSU

      number_Of_Air_T=0;
      number_Of_Solid_T=0;
      number_Of_Water_T=0;

      if strcmp(MethodName, 'Iterative Otsu')

set(handles.imageSlider,'Value',get(handles.imageSlider,'Min'));

          for ka = 1: b
              I = sprintf('CroppedImage_1.%02d.png', ka);
              I=imread(I);
              [Row,Col,r]=MaskPortion(I);
              sizeofmatrix = size(I,1);
              c=[];
              c1=zeros(sizeofmatrix,sizeofmatrix);
              c1=uint8(c1);
              e=0;
              for i=1:sizeofmatrix
                  for j=1:sizeofmatrix
                      d=(j - (Row))^2 + (i - (Col))^2;
                      if d <= (r)^2
                          e=e+1;
                          c(e)=I(i,j);
                          c1(i,j)=I(i,j);
                      end
                  end
              end

      e=uint8(c);
      p=imhist(e);
      mean=0;
      for i=1:256
          mean=mean+(i*p(i,1));
      end
      mean=mean/256;
      sumt=zeros(256,1);
      for i=1:256
          meanli=0;
          for j=1:i-1
              meanli=meanli+(j*p(j,1));
          end
          meanli=meanli/256;
          meangi=0;
          for j=i:256
              meangi=meangi+(j*p(j,1));
          end
```

```
          meangi=meangi/256;
          sumli=0;
          sumgi=0;
          for ii=1:i-1
              sumli=sumli+p(ii,1)*((meanli-mean)^2);
          end
          for ii=i:256
              sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
          end

          sumt(i,1)=sumgi+sumli;

      end

      Threshold1=find(sumt==max(sumt));

      [counts,N]=imhist(e);
      counts1=counts(1:Threshold1);
      mu1=cumsum(counts1);   % To get the total summation of intensity
values
      N1 = (0:Threshold1-1)';
      mean=(sum(N1.*counts1))/mu1(end);
      mean=round(mean);
      IntensityRegion1=mean;

      counts2=counts(Threshold1+1:end);
      mu2=cumsum(counts2);   % To get the total summation of intensity
values
      N2 = (Threshold1:255)';
      mean=(sum(N2.*counts2))/mu2(end);
      mean=round(mean);
      IntensityRegion2=mean;

      Threshold2(ka,1) = (IntensityRegion1+IntensityRegion2)/2;
      Threshold2(ka,1)=round(Threshold2(ka,1));

      % Iterative Process to Find the Ideal Value for Threshold2

      [counts,N]=imhist(e);
      counts3=counts(1:Threshold2(ka,1));
      mu1=cumsum(counts3);   % To get the total summation of intensity
values
      N1 = (0:Threshold2(ka,1)-1)';
      mean=(sum(N1.*counts3))/mu1(end);
      mean=round(mean);
      IntensityRegion1=mean;

      counts4=counts(Threshold2(ka,1)+1:end);
      mu2=cumsum(counts4);   % To get the total summation of intensity
values
      N2 = (Threshold2(ka,1):255)';
      mean=(sum(N2.*counts4))/mu2(end);
      mean=round(mean);
      IntensityRegion2=mean;
```

```
      Thresholdnew = (IntensityRegion1+IntensityRegion2)/2;
      Thresholdnew=round(Thresholdnew);


      % Check if Thresholdnew is an Acceptable Choice for Threshold2

      loopCounter=1;

      while Thresholdnew-5 < Threshold2(ka,1) && Thresholdnew+5 <
Threshold2(ka,1)
          Threshold2(ka,1)=Thresholdnew;
          [counts,N]=imhist(e);
          counts5=counts(1:Threshold2(ka,1));
          mu1=cumsum(counts5);   % To get the total summation of
intensity values
          N1 = (0:Threshold2(ka,1)-1)';
          mean=(sum(N1.*counts5))/mu1(end);
          mean=round(mean);
          IntensityRegion1=mean;

          counts6=counts(Threshold2(ka,1)+1:end);
          mu2=cumsum(counts6);   % To get the total summation of
intensity values
          N2 = (Threshold2(ka,1):255)';
          mean=(sum(N2.*counts6))/mu2(end);
          mean=round(mean);
          IntensityRegion2=mean;

          Thresholdnew = (IntensityRegion1+IntensityRegion2)/2;
          Thresholdnew=round(Thresholdnew);

          if loopCounter>=10
              break;
          end

          loopCounter=loopCounter+1;

      end

      Threshold2(ka,1)=Thresholdnew;

          end

      Threshold2=sum(Threshold2(1:b));
      Threshold2=Threshold2/b;
      Threshold2=round(Threshold2);
      AT2=Threshold2;
      AT2=num2str(AT2);
      set(handles.AT2Box, 'String', AT2);

      end

      %%% REFINED STATISTICAL-BASED
```

```
number_Of_Air_T=0;
number_Of_Solid_T=0;
number_Of_Water_T=0;

if strcmp(MethodName, 'Refined Statistical-Based')
k1= handles.k1;
k1=str2double(k1);
k2= handles.k2;
k2=str2double(k2);
set(handles.imageSlider,'Value',get(handles.imageSlider,'Min'));

for ka = 1: b
    I = sprintf('CroppedImage_1.%02d.png', ka);
    I=imread(I);
    [Row,Col,r]=MaskPortion(I);
    sizeofmatrix = size(I,1);
    c=[];
    c1=zeros(sizeofmatrix,sizeofmatrix);
    c1=uint8(c1);
    e=0;
    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
            d=(j - (Row))^2 + (i - (Col))^2;
            if d <= (r)^2
                e=e+1;
                c(e)=I(i,j);
                c1(i,j)=I(i,j);
            end
        end
    end

    e=uint8(c);
    [counts,N]=imhist(e);
    mu=cumsum(counts);   % To get the total summation of intensity
values
    mean=(sum(N.*counts))/mu(end);
    mean=round(mean);
    variance=sum((counts-mean).^2)/(mu-1);
    variance=sum(variance);
    standarddeviation=sqrt(variance);
    standarddeviation=round(standarddeviation);

    T1=mean-(k1*standarddeviation);
    T2=mean+(k2*standarddeviation);

    Threshold1(ka,1)=abs(T1+T2)/2;

end

Threshold1=sum(Threshold1(1:b));
Threshold1=Threshold1/b;
Threshold1=round(Threshold1);
AT1=Threshold1;
```

```
      AT1=num2str(AT1);
      set(handles.AT1Box, 'String', AT1);
      end

      %%% REFINED STATISTICAL-BASED

      number_Of_Air_T=0;
      number_Of_Solid_T=0;
      number_Of_Water_T=0;

      if strcmp(MethodName, 'Refined Statistical-Based')
          k1= handles.k1;
          k1=str2double(k1);
          k2= handles.k2;
          k2=str2double(k2);

set(handles.imageSlider,'Value',get(handles.imageSlider,'Min'));

          for ka = 1: b
              I = sprintf('CroppedImage_1.%02d.png', ka);
              I=imread(I);
              [Row,Col,r]=MaskPortion(I);
              sizeofmatrix = size(I,1);
              c=[];
              c1=zeros(sizeofmatrix,sizeofmatrix);
              c1=uint8(c1);
              e=0;
              for i=1:sizeofmatrix
                  for j=1:sizeofmatrix
                      d=(j - (Row))^2 + (i - (Col))^2;
                      if d <= (r)^2
                          e=e+1;
                          c(e)=I(i,j);
                          c1(i,j)=I(i,j);
                      end
                  end
              end

              e=uint8(c);
              [counts,N]=imhist(e);
              mu=cumsum(counts);   % To get the total summation of
intensity values
              mean=(sum(N.*counts))/mu(end);
              mean=round(mean);
              variance=sum((counts-mean).^2)/(mu-1);
              variance=sum(variance);
              standarddeviation=sqrt(variance);
              standarddeviation=round(standarddeviation);

              T1=mean-(k1*standarddeviation);
              T2=mean+(k2*standarddeviation);
              Threshold1=abs(T1+T2)/2;
              Threshold2(ka,1)=abs(T1-T2)/2;
          end
```

```
      Threshold2=sum(Threshold2(1:b));
      Threshold2=Threshold2/b;
      Threshold2=round(Threshold2);
      AT2=Threshold2;
      AT2=num2str(AT2);
      set(handles.AT2Box, 'String', AT2);
      end

      % --- Executes on slider movement.
      function imageSlider_Callback(hObject, eventdata, handles)
      % hObject    handle to imageSlider (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)

      % Hints: get(hObject,'Value') returns position of slider
      %        get(hObject,'Min') and get(hObject,'Max') to determine
range of slider

      imageSlider_value = get(hObject,'Value');

      %get the current max value from the slider
      numImages = get(handles.imageSlider, 'Max');

      %calculate the image number to display
      imageNum = floor(imageSlider_value)...
               + (sign(imageSlider_value)...
               * ( abs(imageSlider_value) - floor(imageSlider_value)
)...
               * numImages);

      %create a string from the number which always has 2 digits
      str = sprintf('%02.0f',imageNum);

      %retrieve filename data from app data
      ptNum    = getappdata(handles.axes1, 'ptNum');
      imageType = getappdata(handles.axes1, 'imageType');
      filepath = getappdata(handles.axes1, 'filePath');

      %create the filename as a cell string
      filename = strcat(ptNum, '.', str, '.', imageType);

      %create full path to the image
      imageStr = [filepath, filename{1}];

      %read in image data
      image = imread(imageStr);

      %bring current axes in focus and show image
      axes(handles.axes1);
      imshow(image, []); %[] = [Imin Imax]

      axes(handles.axes4);
```

```
imshow(image,[]);

%store image data and slice number in axes
setappdata(handles.axes1, 'image',     image);
setappdata(handles.axes1, 'sliceNum',  imageSlider_value);

axes(handles.axes2);
sizeofmatrix = size(image,1);
[Row,Col,r]=MaskPortion(image);
c=[];
c1=zeros(sizeofmatrix,sizeofmatrix);
c1=uint8(c1);
e=0;
for i=1:sizeofmatrix
    for j=1:sizeofmatrix
        d=(j - (Row))^2 + (i - (Col))^2;
        if d <= (r)^2
            e=e+1;
            c(e)=image(i,j);
            c1(i,j)=image(i,j);
        end
    end
end
histogram(c);

set(handles.CurrentName, 'String', filename)
contents = get(handles.Methods,'String');
  MethodName = contents{get(handles.Methods,'Value')};

 if strcmp(MethodName, 'Otsu ')
     axes(handles.axes3);
     sizeofmatrix = size(image,1);
     [Row,Col,r]=MaskPortion(image);
     c=[];
     c1=zeros(sizeofmatrix,sizeofmatrix);
     c1=uint8(c1);
     e=0;
     for i=1:sizeofmatrix
         for j=1:sizeofmatrix
             d=(j - (Row))^2 + (i - (Col))^2;
             if d <= (r)^2
                 e=e+1;
                 c(e)=image(i,j);
                 c1(i,j)=image(i,j);
             end
         end
     end

     e=uint8(c);
     p=imhist(e);
     mean=0;

for i=1:256
    mean=mean+(i*p(i,1));
```

```
end

mean=mean/256;
sumt=zeros(256,1);

for i=1:256
    meanli=0;
    for j=1:i-1
        meanli=meanli+(j*p(j,1));
    end
    meanli=meanli/256;
    meangi=0;
    for j=i:256
        meangi=meangi+(j*p(j,1));
    end
    meangi=meangi/256;
    sumli=0;
    sumgi=0;
    for ii=1:i-1
        sumli=sumli+p(ii,1)*((meanli-mean)^2);
    end
    for ii=i:256
        sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
    end
    sumt(i,1)=sumgi+sumli;

end

Threshold1=find(sumt==max(sumt));

% Following lines of code are used to determine Threshold 2

pmax = p(Threshold1,1);
p2 = p(1:Threshold1);
mean=0;

for i=1:Threshold1
    mean=mean+(i*p2(i,1));
end

mean=mean/256;
sumt=zeros(256,1);

for i=1:Threshold1
    meanli=0;
    for j=1:Threshold1-1
        meanli=meanli+(j*p2(j,1));
    end

    meanli=meanli/256;
    meangi=0;

    for j=i:Threshold1
        meangi=meangi+(j*p2(j,1));
```

```
    end

    meangi=meangi/256;
    sumli=0;
    sumgi=0;

    for ii=1:Threshold1-1
        sumli=sumli+p2(ii,1)*((meanli-mean)^2);
    end
    for ii=i:Threshold1
        sumgi=sumgi+p2(ii,1)*((meangi-mean)^2);
    end

    sumt(i,1)=sumgi+sumli;

end

Threshold2=find(sumt==max(sumt));

c2=zeros(sizeofmatrix,sizeofmatrix);
c2=uint8(c2);
for i=1:sizeofmatrix
    for j=1:sizeofmatrix
        d=(j - (Row))^2 + (i - (Col))^2;
        if d <= (r)^2
            if c1(i,j)>Threshold1
                c2(i,j)=100;
            elseif c1(i,j) < Threshold2
                c2(i,j)=20;
            else
                c2(i,j) = 60;
            end

        end
    end
end

imshow(c2,[]);
 end

if strcmp(MethodName, 'Iterative Otsu')
    axes(handles.axes3);
    sizeofmatrix = size(image,1);
    [Row,Col,r]=MaskPortion(image);
    c=[];
    c1=zeros(sizeofmatrix,sizeofmatrix);
    c1=uint8(c1);
    e=0;
    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
            d=(j - (Row))^2 + (i - (Col))^2;
            if d <= (r)^2
                e=e+1;
                c(e)=image(i,j);
```

```
                    c1(i,j)=image(i,j);
                end
            end
        end

    e=uint8(c);
    p=imhist(e);
    mean=0;

    for i=1:256
        mean=mean+(i*p(i,1));
    end

    mean=mean/256;
    sumt=zeros(256,1);

    for i=1:256
        meanli=0;
        for j=1:i-1
            meanli=meanli+(j*p(j,1));
        end

        meanli=meanli/256;
        meangi=0;

        for j=i:256
            meangi=meangi+(j*p(j,1));
        end

        meangi=meangi/256;
        sumli=0;
        sumgi=0;

        for ii=1:i-1
            sumli=sumli+p(ii,1)*((meanli-mean)^2);
        end
        for ii=i:256
            sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
        end

        sumt(i,1)=sumgi+sumli;

    end

    Threshold1=find(sumt==max(sumt));

    [counts,N]=imhist(e);
    counts1=counts(1:Threshold1);
    mu1=cumsum(counts1);   % To get the total summation of intensity
values
    N1 = (0:Threshold1-1)';
    mean=(sum(N1.*counts1))/mu1(end);
    mean=round(mean);
    IntensityRegion1=mean;
```

```
      counts2=counts(Threshold1+1:end);
      mu2=cumsum(counts2);   % To get the total summation of intensity
values
      N2 = (Threshold1:255)';
      mean=(sum(N2.*counts2))/mu2(end);
      mean=round(mean);
      IntensityRegion2=mean;

      Threshold2 = (IntensityRegion1+IntensityRegion2)/2;
      Threshold2 = round(Threshold2);

      % Iterative Process to Find the Ideal Value for Threshold2

      [counts,N]=imhist(e);
      counts3=counts(1:Threshold2);
      mu1=cumsum(counts3);   % To get the total summation of intensity
values
      N1 = (0:Threshold2-1)';
      mean=(sum(N1.*counts3))/mu1(end);
      mean=round(mean);
      IntensityRegion1=mean;

      counts4=counts(Threshold2+1:end);
      mu2=cumsum(counts4);   % To get the total summation of intensity
values
      N2 = (Threshold2:255)';
      mean=(sum(N2.*counts4))/mu2(end);
      mean=round(mean);
      IntensityRegion2 = mean;

      Thresholdnew = (IntensityRegion1+IntensityRegion2)/2;
      Thresholdnew = round(Thresholdnew);


      % Check if Thresholdnew is an Acceptable Choice for Threshold2

      loopCounter=1;

      while Thresholdnew-5 < Threshold2 && Thresholdnew+5 < Threshold2
          Threshold2=Thresholdnew;
          [counts,N]=imhist(e);
          counts5=counts(1:Threshold2);
          mu1=cumsum(counts5);   % To get the total summation of
intensity values
          N1 = (0:Threshold2-1)';
          mean=(sum(N1.*counts5))/mu1(end);
          mean=round(mean);
          IntensityRegion1=mean;

          counts6=counts(Threshold2+1:end);
          mu2=cumsum(counts6);   % To get the total summation of
intensity values
          N2 = (Threshold2:255)';
```

```
        mean=(sum(N2.*counts6))/mu2(end);
        mean=round(mean);
        IntensityRegion2=mean;

        Thresholdnew = (IntensityRegion1+IntensityRegion2)/2;
        Thresholdnew=round(Thresholdnew);

        if loopCounter>=10
            break;
        end

    loopCounter=loopCounter+1;

    end

    Threshold2=Thresholdnew;

    % Portion of Code that Calculates the Porosity and Void Ratio of
the Image

    c2=zeros(sizeofmatrix,sizeofmatrix);
    c2=uint8(c2);
    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
            d=(j - (Row))^2 + (i - (Col))^2;
            if d <= (r)^2
                if c1(i,j)>Threshold1
                    c2(i,j)=100;
                elseif c1(i,j) < Threshold2
                    c2(i,j)=20;
                else
                    c2(i,j) = 60;
                end

            end
        end
    end
    imshow(c2,[]);
          end

    %%% REFINED STATISTICAL-BASED

    if strcmp(MethodName, 'Refined Statistical-Based')
        k1= handles.k1;
        k1=str2double(k1);
        k2= handles.k2;
        k2=str2double(k2);
        axes(handles.axes3);
        sizeofmatrix = size(image,1);
        [Row,Col,r]=MaskPortion(image);
        c=[];
        c1=zeros(sizeofmatrix,sizeofmatrix);
        c1=uint8(c1);
        e=0;
```

```
          for i=1:sizeofmatrix
              for j=1:sizeofmatrix
                  d=(j - (Row))^2 + (i - (Col))^2;
                  if d <= (r)^2
                      e=e+1;
                      c(e)=image(i,j);
                      c1(i,j)=image(i,j);
                  end
              end
          end

          e=uint8(c);

      % Automatic Threshold using Refined Statistical-Based Method

      [counts,N]=imhist(e);
      mu=cumsum(counts);   % To get the total summation of intensity
values
      mean=(sum(N.*counts))/mu(end);
      mean=round(mean);
      variance=sum((counts-mean).^2)/(mu-1);
      variance=sum(variance);
      standarddeviation=sqrt(variance);
      standarddeviation=round(standarddeviation);

      T1=mean-(k1*standarddeviation);
      T2=mean+(k2*standarddeviation);

      Threshold1=abs(T1+T2)/2;
      Threshold1=round(Threshold1);

      Threshold2=abs(T1-T2)/2;
      Threshold2=round(Threshold2);

      % Portion of Code that Calculates the Porosity and Void Ratio of
the Image

      c2=zeros(sizeofmatrix,sizeofmatrix);
      c2=uint8(c2);
      for i=1:sizeofmatrix
          for j=1:sizeofmatrix
              d=(j - (Row))^2 + (i - (Col))^2;
              if d <= (r)^2
                  if c1(i,j)>Threshold1
                      c2(i,j)=100;
                  elseif c1(i,j) < Threshold2
                      c2(i,j)=20;
                  else
                      c2(i,j) = 60;
                  end

              end
          end
      end
```

```
      imshow(c2,[]);

      end

      function CurrentName_Callback(hObject, eventdata, handles)
      % hObject    handle to CurrentName (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)

      % Hints: get(hObject,'String') returns contents of CurrentName as
text
      %        str2double(get(hObject,'String')) returns contents of
CurrentName as a double
      set(handles.CurrentName, 'String', [pathName filename])

      % --- Executes during object creation, after setting all
properties.
      function CurrentName_CreateFcn(hObject, eventdata, handles)
      % hObject    handle to CurrentName (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    empty - handles not created until after all
CreateFcns called

      % Hint: edit controls usually have a white background on Windows.
      %       See ISPC and COMPUTER.
      if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
          set(hObject,'BackgroundColor','white');
      end

      function numImages_Callback(hObject, eventdata, handles)
      % hObject    handle to numImages (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)

      % Hints: get(hObject,'String') returns contents of numImages as
text
      %        str2double(get(hObject,'String')) returns contents of
numImages as a double

      updateSlider(handles);

      function updateSlider(handles)
      % This function updates the slider to have the correct min, max,
value, and
      %   step size

      %get the current slice number which were stored in the figure axes
      sliceNum = getappdata(handles.axes1,'sliceNum');

      if(isempty(sliceNum))%may be empty if the figure has not been
```

```
initialized
          sliceNum = 1;    %set it to a default
      end

      %get the number written in the text box which is the maximum
number of
      %images to be viewed
      NumImageslice = str2double(get(handles.numImages,'String'));

      %there are only NumImageslice - 1 images total, because we start
at 1
      step = 1/(NumImageslice - 1);

      %set values for the slider bar
      set(handles.imageSlider, 'Max',        NumImageslice);
      set(handles.imageSlider, 'Min',        1);
      set(handles.imageSlider, 'SliderStep', [step step]);

      %set current value to the slice we are viewing
      set(handles.imageSlider, 'Value', sliceNum);

      % --- Executes during object creation, after setting all
properties.
      function numImages_CreateFcn(hObject, eventdata, handles)
      % hObject    handle to numImages (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    empty - handles not created until after all
CreateFcns called

      % Hint: edit controls usually have a white background on Windows.
      %       See ISPC and COMPUTER.
      if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
          set(hObject,'BackgroundColor','white');
      end

      % --- Executes on button press in ContBright.
      function ContBright_Callback(hObject, eventdata, handles)
      axes(handles.axes4);
      imcontrast(gcf);

      function VoidText_Callback(hObject, eventdata, handles)
      % hObject    handle to VoidText (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)

      % Hints: get(hObject,'String') returns contents of VoidText as
text
      %        str2double(get(hObject,'String')) returns contents of
VoidText as a double

      % --- Executes during object creation, after setting all
```

```
properties.
      function VoidText_CreateFcn(hObject, eventdata, handles)
      % hObject    handle to VoidText (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    empty - handles not created until after all
CreateFcns called

      % Hint: edit controls usually have a white background on Windows.
      %       See ISPC and COMPUTER.
      if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
          set(hObject,'BackgroundColor','white');
      end

      function SatText_Callback(hObject, eventdata, handles)
      % hObject    handle to SatText (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)

      % Hints: get(hObject,'String') returns contents of SatText as text
      %        str2double(get(hObject,'String')) returns contents of
SatText as a double

      % --- Executes during object creation, after setting all
properties.
      function SatText_CreateFcn(hObject, eventdata, handles)
      % hObject    handle to SatText (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    empty - handles not created until after all
CreateFcns called

      % Hint: edit controls usually have a white background on Windows.
      %       See ISPC and COMPUTER.
      if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
          set(hObject,'BackgroundColor','white');
      end

      function AT1Box_Callback(hObject, eventdata, handles)
      % hObject    handle to AT1Box (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)

      % Hints: get(hObject,'String') returns contents of AT1Box as text
      %        str2double(get(hObject,'String')) returns contents of
AT1Box as a double

      % --- Executes during object creation, after setting all
properties.
      function AT1Box_CreateFcn(hObject, eventdata, handles)
```

```
      % hObject    handle to AT1Box (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    empty - handles not created until after all
CreateFcns called

      % Hint: edit controls usually have a white background on Windows.
      %       See ISPC and COMPUTER.
      if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
          set(hObject,'BackgroundColor','white');
      end

      function AT2Box_Callback(hObject, eventdata, handles)
      % hObject    handle to AT2Box (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)

      % Hints: get(hObject,'String') returns contents of AT2Box as text
      %        str2double(get(hObject,'String')) returns contents of
AT2Box as a double

      % --- Executes during object creation, after setting all
properties.
      function AT2Box_CreateFcn(hObject, eventdata, handles)
      % hObject    handle to AT2Box (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    empty - handles not created until after all
CreateFcns called

      % Hint: edit controls usually have a white background on Windows.
      %       See ISPC and COMPUTER.
      if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
          set(hObject,'BackgroundColor','white');
      end

      % --- Executes on selection change in Type.
      function Type_Callback(hObject, eventdata, handles)
      % hObject    handle to Type (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)

      % Hints: contents = cellstr(get(hObject,'String')) returns Type
contents as cell array
      %        contents{get(hObject,'Value')} returns selected item from
Type

      % --- Executes during object creation, after setting all
properties.
      function Type_CreateFcn(hObject, eventdata, handles)
```

```
      % hObject    handle to Type (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    empty - handles not created until after all
CreateFcns called

      % Hint: popupmenu controls usually have a white background on
Windows.
      %       See ISPC and COMPUTER.
      if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
          set(hObject,'BackgroundColor','white');
      end

      % --- Executes on button press in SingleThreshold.
      function SingleThreshold_Callback(hObject, eventdata, handles)
      contents = get(handles.Methods,'String');
      MethodName = contents{get(handles.Methods,'Value')};
      getappdata(handles.axes1,'fileName');

      number_Of_Air_T=0;
      number_Of_Solid_T=0;
      number_Of_Water_T=0;

      contents = get(handles.Methods,'String');
      MethodName = contents{get(handles.Methods,'Value')};
      contents2 = get(handles.Type,'String');
      TypeName = contents2{get(handles.Type,'Value')};
      getappdata(handles.axes1,'fileName');

      %%% OTSU

      if strcmp(MethodName, 'Otsu ') && strcmp(TypeName, 'Threshold
One')

set(handles.imageSlider,'Value',get(handles.imageSlider,'Min'));
          A = uigetfile({'*.jpg;*.tif;*.bmp;*.png;*.gif','All Image
Files'},'Please Select A Cropped Image from Folder that Appears');
          I=imread(A);
          set(handles.Name2, 'String', A);
          [Row,Col,r]=MaskPortion(I);
          sizeofmatrix = size(I,1);
          c=[];
          c1=zeros(sizeofmatrix,sizeofmatrix);
          c1=uint8(c1);
          e=0;
          for i=1:sizeofmatrix
              for j=1:sizeofmatrix
                  d=(j - (Row))^2 + (i - (Col))^2;
                  if d <= (r)^2
                      e=e+1;
                      c(e)=I(i,j);
                      c1(i,j)=I(i,j);
                  end
```

```
        end
    end

    e=uint8(c);
    p=imhist(e);
    mean=0;

    for i=1:256
        mean=mean+(i*p(i,1));
    end

    mean=mean/256;
    sumt=zeros(256,1);

    for i=1:256
        meanli=0;
        for j=1:i-1
            meanli=meanli+(j*p(j,1));
        end
        meanli=meanli/256;
        meangi=0;
        for j=i:256
            meangi=meangi+(j*p(j,1));
        end
        meangi=meangi/256;
        sumli=0;
        sumgi=0;
        for ii=1:i-1
            sumli=sumli+p(ii,1)*((meanli-mean)^2);
        end
        for ii=i:256
            sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
        end

        sumt(i,1)=sumgi+sumli;
    end

    Threshold1=find(sumt==max(sumt));
    T1=Threshold1;
    T1=num2str(T1);
    set(handles.TBox, 'String', T1);

end

number_Of_Air_T=0;
number_Of_Solid_T=0;
number_Of_Water_T=0;

contents = get(handles.Methods,'String');
MethodName = contents{get(handles.Methods,'Value')};
contents2 = get(handles.Type,'String');
TypeName = contents2{get(handles.Type,'Value')};
getappdata(handles.axes1,'fileName');
```

```
      %%% ITERATIVE OTSU

      if strcmp(MethodName, 'Iterative Otsu') && strcmp(TypeName,
'Threshold One')

set(handles.imageSlider,'Value',get(handles.imageSlider,'Min'));
          A = uigetfile({'*.jpg;*.tif;*.bmp;*.png;*.gif','All Image
Files'},'Please Select A Cropped Image from Folder that Appears');
          I=imread(A);
          set(handles.Name2, 'String', A);
          [Row,Col,r]=MaskPortion(I);
          sizeofmatrix = size(I,1);

          c=[];
          c1=zeros(sizeofmatrix,sizeofmatrix);
          c1=uint8(c1);
          e=0;
          for i=1:sizeofmatrix
              for j=1:sizeofmatrix
                  d=(j - (Row))^2 + (i - (Col))^2;
                  if d <= (r)^2
                      e=e+1;
                      c(e)=I(i,j);
                      c1(i,j)=I(i,j);
                  end
              end
          end

          e=uint8(c);
          p=imhist(e);
          mean=0;

          for i=1:256
              mean=mean+(i*p(i,1));
          end

          mean=mean/256;
          sumt=zeros(256,1);
          for i=1:256
              meanli=0;
              for j=1:i-1
                  meanli=meanli+(j*p(j,1));
              end
              meanli=meanli/256;
              meangi=0;
              for j=i:256
                  meangi=meangi+(j*p(j,1));
              end
              meangi=meangi/256;
              sumli=0;
              sumgi=0;
              for ii=1:i-1
                  sumli=sumli+p(ii,1)*((meanli-mean)^2);
              end
```

```
            for ii=i:256
                sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
            end

            sumt(i,1)=sumgi+sumli;
        end

        Threshold1=find(sumt==max(sumt));
        T1=Threshold1;
        T1=num2str(T1);
        set(handles.TBox, 'String', T1);
    end

    number_Of_Air_T=0;
    number_Of_Solid_T=0;
    number_Of_Water_T=0;

    contents = get(handles.Methods,'String');
    MethodName = contents{get(handles.Methods,'Value')};
    contents2 = get(handles.Type,'String');
    TypeName = contents2{get(handles.Type,'Value')};
    getappdata(handles.axes1,'fileName');

    %%% REFINED STATISTICAL-BASED

    if strcmp(MethodName, 'Refined Statistical-Based') &&
strcmp(TypeName, 'Threshold One')
        k = warndlg('Make sure to choose values for free/fitting
parameters k1 and k2!','Warning');
        waitfor(k);
        k1= handles.k1;
        k1=str2double(k1);
        k2= handles.k2;
        k2=str2double(k2);


set(handles.imageSlider,'Value',get(handles.imageSlider,'Min'));
        A = uigetfile({'*.jpg;*.tif;*.bmp;*.png;*.gif','All Image
Files'},'Please Select A Cropped Image from Folder that Appears');
        I=imread(A);
        set(handles.Name2, 'String', A);
        [Row,Col,r]=MaskPortion(I);
        sizeofmatrix = size(I,1);

        c=[];
        c1=zeros(sizeofmatrix,sizeofmatrix);
        c1=uint8(c1);
        e=0;
        for i=1:sizeofmatrix
            for j=1:sizeofmatrix
                d=(j - (Row))^2 + (i - (Col))^2;
                if d <= (r)^2
                    e=e+1;
                    c(e)=I(i,j);
```

```
                    c1(i,j)=I(i,j);
                end
            end
        end

        e = uint8(c);

    % Automatic Threshold using Refined Statistical-Based Method

    [counts,N]=imhist(e);
    mu=cumsum(counts);   % To get the total summation of intensity
values
    mean=(sum(N.*counts))/mu(end);
    mean=round(mean);
    variance=sum((counts-mean).^2)/(mu-1);
    variance=sum(variance);
    standarddeviation=sqrt(variance);
    standarddeviation=round(standarddeviation);

    T1=mean-(k1*standarddeviation);
    T2=mean+(k2*standarddeviation);

    Threshold1=abs(T1+T2)/2;
    Threshold1=round(Threshold1);
    T1=Threshold1;
    T1=num2str(T1);
    set(handles.TBox, 'String', T1);
    end

    contents = get(handles.Methods,'String');
    MethodName = contents{get(handles.Methods,'Value')};
    getappdata(handles.axes1,'fileName');

    number_Of_Air_T=0;
    number_Of_Solid_T=0;
    number_Of_Water_T=0;

    contents = get(handles.Methods,'String');
    MethodName = contents{get(handles.Methods,'Value')};
    contents2 = get(handles.Type,'String');
    TypeName = contents2{get(handles.Type,'Value')};
    getappdata(handles.axes1,'fileName');

    %%% OTSU

    if strcmp(MethodName, 'Otsu ') && strcmp(TypeName, 'Threshold
Two')

set(handles.imageSlider,'Value',get(handles.imageSlider,'Min'));
        A = uigetfile({'*.jpg;*.tif;*.bmp;*.png;*.gif','All Image
Files'},'Please Select A Cropped Image from Folder that Appears');
        I=imread(A);
        set(handles.Name2, 'String', A);
        [Row,Col,r]=MaskPortion(I);
```

```
    sizeofmatrix = size(I,1);

    c=[];
    c1=zeros(sizeofmatrix,sizeofmatrix);
    c1=uint8(c1);
    e=0;
    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
            d=(j - (Row))^2 + (i - (Col))^2;
            if d <= (r)^2
                e=e+1;
                c(e)=I(i,j);
                c1(i,j)=I(i,j);
            end
        end
    end

    e=uint8(c);
    p=imhist(e);
    mean=0;

    for i=1:256
        mean=mean+(i*p(i,1));
    end

    mean=mean/256;
    sumt=zeros(256,1);

    for i=1:256
        meanli=0;
        for j=1:i-1
            meanli=meanli+(j*p(j,1));
        end
        meanli=meanli/256;
        meangi=0;
        for j=i:256
            meangi=meangi+(j*p(j,1));
        end
        meangi=meangi/256;
        sumli=0;
        sumgi=0;
        for ii=1:i-1
            sumli=sumli+p(ii,1)*((meanli-mean)^2);
        end
        for ii=i:256
            sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
        end
        sumt(i,1)=sumgi+sumli;
    end

    Threshold1=find(sumt==max(sumt));

% Following lines of code are used to determine Threshold 2
```

```
pmax = p(Threshold1,1);
p2 = p(1:Threshold1);
mean=0;

for i=1:Threshold1
    mean=mean+(i*p2(i,1));
end

mean=mean/256;
sumt=zeros(256,1);

for i=1:Threshold1
    meanli=0;
    for j=1:Threshold1-1
        meanli=meanli+(j*p2(j,1));
    end
    meanli=meanli/256;
    meangi=0;
    for j=i:Threshold1
        meangi=meangi+(j*p2(j,1));
    end
    meangi=meangi/256;
    sumli=0;
    sumgi=0;
    for ii=1:Threshold1-1
        sumli=sumli+p2(ii,1)*((meanli-mean)^2);
    end
    for ii=i:Threshold1
        sumgi=sumgi+p2(ii,1)*((meangi-mean)^2);
    end
    sumt(i,1)=sumgi+sumli;
end

Threshold2=find(sumt==max(sumt));
T2=Threshold2;
T2=num2str(T2);
set(handles.TBox, 'String', T2);
end

contents = get(handles.Methods,'String');
MethodName = contents{get(handles.Methods,'Value')};
getappdata(handles.axes1,'fileName');

number_Of_Air_T=0;
number_Of_Solid_T=0;
number_Of_Water_T=0;

contents = get(handles.Methods,'String');
MethodName = contents{get(handles.Methods,'Value')};
contents2 = get(handles.Type,'String');
TypeName = contents2{get(handles.Type,'Value')};
getappdata(handles.axes1,'fileName');

%%% ITERATIVE OTSU
```

```
     if strcmp(MethodName, 'Iterative Otsu') && strcmp(TypeName,
'Threshold Two')

set(handles.imageSlider,'Value',get(handles.imageSlider,'Min'));
         A = uigetfile({'*.jpg;*.tif;*.bmp;*.png;*.gif','All Image
Files'},'Please Select A Cropped Image from Folder that Appears');
         I=imread(A);
         set(handles.Name2, 'String', A);
         [Row,Col,r]=MaskPortion(I);
         sizeofmatrix = size(I,1);

         c=[];
         c1=zeros(sizeofmatrix,sizeofmatrix);
         c1=uint8(c1);
         e=0;
         for i=1:sizeofmatrix
             for j=1:sizeofmatrix
                 d=(j - (Row))^2 + (i - (Col))^2;
                 if d <= (r)^2
                     e=e+1;
                     c(e)=I(i,j);
                     c1(i,j)=I(i,j);
                 end
             end
         end

         e=uint8(c);
         p=imhist(e);
         mean=0;

         for i=1:256
         mean=mean+(i*p(i,1));
         end

         mean=mean/256;
         sumt=zeros(256,1);

         for i=1:256
         meanli=0;
         for j=1:i-1
             meanli=meanli+(j*p(j,1));
         end
         meanli=meanli/256;
         meangi=0;
         for j=i:256
             meangi=meangi+(j*p(j,1));
         end
         meangi=meangi/256;
         sumli=0;
         sumgi=0;
         for ii=1:i-1
             sumli=sumli+p(ii,1)*((meanli-mean)^2);
         end
```

```
          for ii=i:256
              sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
          end

          sumt(i,1)=sumgi+sumli;

      end

      Threshold1=find(sumt==max(sumt));

      [counts,N]=imhist(e);
      counts1=counts(1:Threshold1);
      mu1=cumsum(counts1);   % To get the total summation of intensity
values
      N1 = (0:Threshold1-1)';
      mean=(sum(N1.*counts1))/mu1(end);
      mean=round(mean);
      IntensityRegion1=mean;

      counts2=counts(Threshold1+1:end);
      mu2=cumsum(counts2);   % To get the total summation of intensity
values
      N2 = (Threshold1:255)';
      mean=(sum(N2.*counts2))/mu2(end);
      mean=round(mean);
      IntensityRegion2=mean;

      Threshold2 = (IntensityRegion1+IntensityRegion2)/2;
      Threshold2=round(Threshold2);

      % Iterative Process to Find the Ideal Value for Threshold2

      [counts,N]=imhist(e);
      counts3=counts(1:Threshold2);
      mu1=cumsum(counts3);   % To get the total summation of intensity
values
      N1 = (0:Threshold2-1)';
      mean=(sum(N1.*counts3))/mu1(end);
      mean=round(mean);
      IntensityRegion1=mean;

      counts4=counts(Threshold2+1:end);
      mu2=cumsum(counts4);   % To get the total summation of intensity
values
      N2 = (Threshold2:255)';
      mean=(sum(N2.*counts4))/mu2(end);
      mean=round(mean);
      IntensityRegion2=mean;

      Thresholdnew = (IntensityRegion1+IntensityRegion2)/2;
      Thresholdnew=round(Thresholdnew);


      % Check if Thresholdnew is an Acceptable Choice for Threshold2
```

```
      loopCounter=1;

      while Thresholdnew-5 < Threshold2 && Thresholdnew+5 < Threshold2
          Threshold2=Thresholdnew;
          [counts,N]=imhist(e);
          counts5=counts(1:Threshold2);
          mu1=cumsum(counts5);   % To get the total summation of
intensity values
          N1 = (0:Threshold2-1)';
          mean=(sum(N1.*counts5))/mu1(end);
          mean=round(mean);
          IntensityRegion1=mean;

          counts6=counts(Threshold2+1:end);
          mu2=cumsum(counts6);   % To get the total summation of
intensity values
          N2 = (Threshold2:255)';
          mean=(sum(N2.*counts6))/mu2(end);
          mean=round(mean);
          IntensityRegion2=mean;

          Thresholdnew = (IntensityRegion1+IntensityRegion2)/2;
          Thresholdnew=round(Thresholdnew);

          if loopCounter>=10
              break;
          end

          loopCounter=loopCounter+1;

      end

      Threshold2=Thresholdnew;
      T2=Threshold2;
      T2=num2str(T2);
      set(handles.TBox, 'String', T2);
      end

      contents = get(handles.Methods,'String');
      MethodName = contents{get(handles.Methods,'Value')};
      getappdata(handles.axes1,'fileName');

      number_Of_Air_T=0;
      number_Of_Solid_T=0;
      number_Of_Water_T=0;

      contents = get(handles.Methods,'String');
      MethodName = contents{get(handles.Methods,'Value')};
      contents2 = get(handles.Type,'String');
      TypeName = contents2{get(handles.Type,'Value')};
      getappdata(handles.axes1,'fileName');

      %%% REFINED STATISTICAL-BASED
```

```
      if strcmp(MethodName, 'Refined Statistical-Based') &&
strcmp(TypeName, 'Threshold Two')
          k = warndlg('Make sure to choose values for free/fitting
parameters k1 and k2!','Warning');
          waitfor(k);
          k1= handles.k1;
          k1=str2double(k1);
          k2= handles.k2;
          k2=str2double(k2);

set(handles.imageSlider,'Value',get(handles.imageSlider,'Min'));
          A = uigetfile({'*.jpg;*.tif;*.bmp;*.png;*.gif','All Image
Files'},'Please Select A Cropped Image from Folder that Appears');
          I=imread(A);
          set(handles.Name2, 'String', A);

          [Row,Col,r]=MaskPortion(I);
          sizeofmatrix = size(I,1);
          c=[];
          c1=zeros(sizeofmatrix,sizeofmatrix);
          c1=uint8(c1);
          e=0;
          for i=1:sizeofmatrix
              for j=1:sizeofmatrix
                  d=(j - (Row))^2 + (i - (Col))^2;
                  if d <= (r)^2
                      e=e+1;
                      c(e)=I(i,j);
                      c1(i,j)=I(i,j);
                  end
              end
          end

          e = uint8(c);

      % Automatic Threshold using Refined Statistical-Based Method

      [counts,N]=imhist(e);
      mu=cumsum(counts);   % To get the total summation of intensity
values
      mean=(sum(N.*counts))/mu(end);
      mean=round(mean);
      variance=sum((counts-mean).^2)/(mu-1);
      variance=sum(variance);
      standarddeviation=sqrt(variance);
      standarddeviation=round(standarddeviation);

      T1=mean-(k1*standarddeviation);
      T2=mean+(k2*standarddeviation);

      Threshold1=abs(T1+T2)/2;
      Threshold1=round(Threshold1);
```

```
      Threshold2=abs(T1-T2)/2;
      Threshold2=round(Threshold2);
      T2=Threshold2;
      T2=num2str(T2);
      set(handles.TBox, 'String', T2);
      end

      function TBox_Callback(hObject, eventdata, handles)
      % hObject    handle to TBox (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)

      % Hints: get(hObject,'String') returns contents of TBox as text
      %        str2double(get(hObject,'String')) returns contents of
TBox as a double

      % --- Executes during object creation, after setting all
properties.
      function TBox_CreateFcn(hObject, eventdata, handles)
      % hObject    handle to TBox (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    empty - handles not created until after all
CreateFcns called

      % Hint: edit controls usually have a white background on Windows.
      %       See ISPC and COMPUTER.
      if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
          set(hObject,'BackgroundColor','white');
      end

      function Name2_Callback(hObject, eventdata, handles)
      % hObject    handle to Name2 (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)

      % Hints: get(hObject,'String') returns contents of Name2 as text
      %        str2double(get(hObject,'String')) returns contents of
Name2 as a double

      % --- Executes during object creation, after setting all
properties.
      function Name2_CreateFcn(hObject, eventdata, handles)
      % hObject    handle to Name2 (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    empty - handles not created until after all
CreateFcns called

      % Hint: edit controls usually have a white background on Windows.
      %       See ISPC and COMPUTER.
```

```
      if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
          set(hObject,'BackgroundColor','white');
      end

      % --- Executes on button press in Excel.
      function Excel_Callback(hObject, eventdata, handles)
      k = warndlg('Make sure to choose values for free/fitting
parameters k1 and k2!','Warning');
      waitfor(k);
      k1= handles.k1;
      k1=str2double(k1);
      k2= handles.k2;
      k2=str2double(k2);

      str1='Segmentation Method';
      O='Otsu';
      P= 'Iterative Otsu';
      KSW= 'Refined Statistical-Based';

      str2='Void Ratio';
      str3='Degree of Saturation';
      str3='Average Threshold One Value for all Images';
      str4='Average Threshold Two Value for all Images';

      d = uigetdir('*.png');
      cd(d);
      q=dir('*.png');
      b = numel(q);

      k = warndlg('Exporting of data can take a couple of minutes
depending on the number of images being analyzed. Please do not click
anything until data exporting is completed.','Warning');
      waitfor(k);

      %%% OTSU

      number_Of_Air_T=0;
      number_Of_Solid_T=0;
      number_Of_Water_T=0;

      for ka = 1: b
          I = sprintf('CroppedImage_1.%02d.png', ka);
          I=imread(I);
          [Row,Col,r]=MaskPortion(I);
          sizeofmatrix = size(I,1);
          c=[];
          c1=zeros(sizeofmatrix,sizeofmatrix);
          c1=uint8(c1);
          e=0;
          for i=1:sizeofmatrix
              for j=1:sizeofmatrix
                  d=(j - (Row))^2 + (i - (Col))^2;
                  if d <= (r)^2
```

```
                e=e+1;
                c(e)=I(i,j);
                c1(i,j)=I(i,j);
            end
        end
    end

% Automatic Threshold using Otsu's Method

e=uint8(c);
p=imhist(e);
mean=0;

for i=1:256
    mean=mean+(i*p(i,1));
end

mean=mean/256;
sumt=zeros(256,1);

for i=1:256
    meanli=0;
    for j=1:i-1
        meanli=meanli+(j*p(j,1));
    end
    meanli=meanli/256;
    meangi=0;
    for j=i:256
        meangi=meangi+(j*p(j,1));
    end
    meangi=meangi/256;
    sumli=0;
    sumgi=0;
    for ii=1:i-1
        sumli=sumli+p(ii,1)*((meanli-mean)^2);
    end
    for ii=i:256
        sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
    end
    sumt(i,1)=sumgi+sumli;

end

Threshold1=find(sumt==max(sumt));
Threshold1a(ka,1)=find(sumt==max(sumt));

% Following lines of code are used to determine Threshold 2

pmax = p(Threshold1,1);
p2 = p(1:Threshold1);
mean=0;

for i=1:Threshold1
    mean=mean+(i*p2(i,1));
```

```
end

mean=mean/256;
sumt=zeros(256,1);

for i=1:Threshold1
    meanli=0;
    for j=1:Threshold1-1
        meanli=meanli+(j*p2(j,1));
    end
    meanli=meanli/256;
    meangi=0;
    for j=i:Threshold1
        meangi=meangi+(j*p2(j,1));
    end
    meangi=meangi/256;
    sumli=0;
    sumgi=0;
    for ii=1:Threshold1-1
        sumli=sumli+p2(ii,1)*((meanli-mean)^2);
    end
    for ii=i:Threshold1
        sumgi=sumgi+p2(ii,1)*((meangi-mean)^2);
    end

    sumt(i,1)=sumgi+sumli;

end

Threshold2=find(sumt==max(sumt));
Threshold2a(ka,1)=find(sumt==max(sumt));

c2=zeros(sizeofmatrix,sizeofmatrix);
c2=uint8(c2);
for i=1:sizeofmatrix
    for j=1:sizeofmatrix
        d=(j - (Row))^2 + (i - (Col))^2;
        if d <= (r)^2
            if c1(i,j)>Threshold1
                c2(i,j)=100;
            elseif c1(i,j) < Threshold2
                c2(i,j)=20;
            else
                c2(i,j) = 60;
            end

        end
    end
end

number_Of_Air_Pixels=sum(sum(c2==20));
number_Of_Solid_Pixels=sum(sum(c2==100));
number_Of_Water_Pixels=sum(sum(c2==60));
```

```
      number_Of_Air_T=number_Of_Air_T+number_Of_Air_Pixels;
      number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;
      number_Of_Water_T=number_Of_Water_T+number_Of_Water_Pixels;

      end

      Threshold1a=sum(Threshold1a(1:b));
      Threshold1a=Threshold1a/(b);
      Threshold1a=round(Threshold1a);
      AT1=Threshold1a;
      AT1=num2str(AT1);
      str5=AT1;

      Threshold2a=sum(Threshold2a(1:b));
      Threshold2a=Threshold2a/(b);
      Threshold2a=round(Threshold2a);
      AT2=Threshold2a;
      AT2=num2str(AT2);
      str6=AT2;

      void_ratio =
(number_Of_Air_T+number_Of_Water_T)/number_Of_Solid_T;
      void_ratio=round(void_ratio,2);
      v=void_ratio;
      v=num2str(v);
      str7=v;

      degree_of_saturation =
(number_Of_Water_T/(number_Of_Air_T+number_Of_Water_T))*100;
      p=degree_of_saturation;
      p=round(p,2);
      p=num2str(p);
      str8=p;

      AH=cell(6,1);
      AH{1,1}='Otsu segmentation method complete.';
      AH{2,1}='Two methods remain...';
      set(handles.listbox3, 'String', AH);
      drawnow();

      %%% ITERATIVE OTSU

      number_Of_Air_T=0;
      number_Of_Solid_T=0;
      number_Of_Water_T=0;

      for ka = 1: b
          I = sprintf('CroppedImage_1.%02d.png', ka);
          I=imread(I);
          [Row,Col,r]=MaskPortion(I);
          sizeofmatrix = size(I,1);
          c=[];
          c1=zeros(sizeofmatrix,sizeofmatrix);
          c1=uint8(c1);
```

```
        e=0;
        for i=1:sizeofmatrix
            for j=1:sizeofmatrix
                d=(j - (Row))^2 + (i - (Col))^2;
                if d <= (r)^2
                    e=e+1;
                    c(e)=I(i,j);
                    c1(i,j)=I(i,j);
                end
            end
        end

        e=uint8(c);
        p=imhist(e);
        mean=0;

        for i=1:256
        mean=mean+(i*p(i,1));
        end

        mean=mean/256;
        sumt=zeros(256,1);

        for i=1:256
        meanli=0;
        for j=1:i-1
            meanli=meanli+(j*p(j,1));
        end
        meanli=meanli/256;
        meangi=0;
        for j=i:256
            meangi=meangi+(j*p(j,1));
        end
        meangi=meangi/256;
        sumli=0;
        sumgi=0;
        for ii=1:i-1
            sumli=sumli+p(ii,1)*((meanli-mean)^2);
        end
        for ii=i:256
            sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
        end

        sumt(i,1)=sumgi+sumli;

    end

    Threshold1=find(sumt==max(sumt));
    Threshold1a(ka,1)=find(sumt==max(sumt));

    [counts,N]=imhist(e);
    counts1=counts(1:Threshold1);
    mu1=cumsum(counts1);   % To get the total summation of intensity
values
```

```
      N1 = (0:Threshold1-1)';
      mean=(sum(N1.*counts1))/mu1(end);
      mean=round(mean);
      IntensityRegion1=mean;

      counts2=counts(Threshold1+1:end);
      mu2=cumsum(counts2);   % To get the total summation of intensity
values
      N2 = (Threshold1:255)';
      mean=(sum(N2.*counts2))/mu2(end);
      mean=round(mean);
      IntensityRegion2=mean;

      Threshold2 = (IntensityRegion1+IntensityRegion2)/2;
      Threshold2=round(Threshold2);

      % Iterative Process to Find the Ideal Value for Threshold2

      [counts,N]=imhist(e);
      counts3=counts(1:Threshold2);
      mu1=cumsum(counts3);   % To get the total summation of intensity
values
      N1 = (0:Threshold2-1)';
      mean=(sum(N1.*counts3))/mu1(end);
      mean=round(mean);
      IntensityRegion1=mean;

      counts4=counts(Threshold2+1:end);
      mu2=cumsum(counts4);   % To get the total summation of intensity
values
      N2 = (Threshold2:255)';
      mean=(sum(N2.*counts4))/mu2(end);
      mean=round(mean);
      IntensityRegion2=mean;

      Thresholdnew = (IntensityRegion1+IntensityRegion2)/2;
      Thresholdnew=round(Thresholdnew);

      % Check if Thresholdnew is an Acceptable Choice for Threshold2

      loopCounter=1;

      while Thresholdnew-5 < Threshold2 && Thresholdnew+5 < Threshold2
          Threshold2=Thresholdnew;
          [counts,N]=imhist(e);
          counts5=counts(1:Threshold2);
          mu1=cumsum(counts5);   % To get the total summation of
intensity values
          N1 = (0:Threshold2-1)';
          mean=(sum(N1.*counts5))/mu1(end);
          mean=round(mean);
          IntensityRegion1=mean;

          counts6=counts(Threshold2+1:end);
```

```
          mu2=cumsum(counts6);   % To get the total summation of
intensity values
          N2 = (Threshold2:255)';
          mean=(sum(N2.*counts6))/mu2(end);
          mean=round(mean);
          IntensityRegion2=mean;

          Thresholdnew = (IntensityRegion1+IntensityRegion2)/2;
          Thresholdnew=round(Thresholdnew);

          if loopCounter>=10
              break;
          end

          loopCounter=loopCounter+1;

      end

      Threshold2=Thresholdnew;
      Threshold2a(ka,1)=Thresholdnew;

      c2=zeros(sizeofmatrix,sizeofmatrix);
      c2=uint8(c2);
      for i=1:sizeofmatrix
          for j=1:sizeofmatrix
              d=(j - (Row))^2 + (i - (Col))^2;
              if d <= (r)^2
                  if c1(i,j)>Threshold1
                      c2(i,j)=100;
                  elseif c1(i,j) < Threshold2
                      c2(i,j)=20;
                  else
                      c2(i,j) = 60;
                  end

              end
          end
      end

      number_Of_Air_Pixels=sum(sum(c2==20));
      number_Of_Solid_Pixels=sum(sum(c2==100));
      number_Of_Water_Pixels=sum(sum(c2==60));

      number_Of_Air_T=number_Of_Air_T+number_Of_Air_Pixels;
      number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;
      number_Of_Water_T=number_Of_Water_T+number_Of_Water_Pixels;

      end

      Threshold1a=sum(Threshold1a(1:b));
      Threshold1a=Threshold1a/(b);
      Threshold1a=round(Threshold1a);
      AT1=Threshold1a;
      AT1=num2str(AT1);
```

```
      str9=AT1;

      Threshold2a=sum(Threshold2a(1:b));
      Threshold2a=Threshold2a/(b);
      Threshold2a=round(Threshold2a);
      AT2=Threshold2a;
      AT2=num2str(AT2);
      str10=AT2;

      void_ratio =
(number_Of_Air_T+number_Of_Water_T)/number_Of_Solid_T;
      void_ratio=round(void_ratio,2);
      v=void_ratio;
      v=num2str(v);
      str11=v;

      degree_of_saturation =
(number_Of_Water_T/(number_Of_Air_T+number_Of_Water_T))*100;
      p=degree_of_saturation;
      p=round(p,2);
      p=num2str(p);
      str12=p;

      AH{3,1}='Iterative Otsu segmentation method complete.';
      AH{4,1}='One methods remain...';
      set(handles.listbox3, 'String', AH);
      drawnow()

      %%% REFINED STATISTICAL-BASED

      number_Of_Air_T=0;
      number_Of_Solid_T=0;
      number_Of_Water_T=0;

      for ka = 1: b
          I = sprintf('CroppedImage_1.%02d.png', ka);
          I=imread(I);
          [Row,Col,r]=MaskPortion(I);
          sizeofmatrix = size(I,1);
          c=[];
          c1=zeros(sizeofmatrix,sizeofmatrix);
          c1=uint8(c1);
          e=0;
          for i=1:sizeofmatrix
              for j=1:sizeofmatrix
                  d=(j - (Row))^2 + (i - (Col))^2;
                  if d <= (r)^2
                      e=e+1;
                      c(e)=I(i,j);
                      c1(i,j)=I(i,j);
                  end
              end
          end
```

```
          e=uint8(c);

      % Automatic Threshold using Refined Statistical-Based Method

      [counts,N]=imhist(e);
      mu=cumsum(counts);   % To get the total summation of intensity
values
      mean=(sum(N.*counts))/mu(end);
      mean=round(mean);
      variance=sum((counts-mean).^2)/(mu-1);
      variance=sum(variance);
      standarddeviation=sqrt(variance);
      standarddeviation=round(standarddeviation);

      T1=mean-(k1*standarddeviation);
      T2=mean+(k2*standarddeviation);

      Threshold1=abs(T1+T2)/2;
      Threshold1a(ka,1)=abs(T1+T2)/2;
      Threshold1=round(Threshold1);

      Threshold2=abs(T1-T2)/2;
      Threshold2a(ka,1)=abs(T1-T2)/2;
      Threshold2=round(Threshold2);

      c2=zeros(sizeofmatrix,sizeofmatrix);
      c2=uint8(c2);
      for i=1:sizeofmatrix
          for j=1:sizeofmatrix
              d=(j - (Row))^2 + (i - (Col))^2;
              if d <= (r)^2
                  if c1(i,j)>Threshold1
                      c2(i,j)=100;
                  elseif c1(i,j) < Threshold2
                      c2(i,j)=20;
                  else
                      c2(i,j) = 60;
                  end
              end
          end
      end

      number_Of_Air_Pixels=sum(sum(c2==20));
      number_Of_Solid_Pixels=sum(sum(c2==100));
      number_Of_Water_Pixels=sum(sum(c2==60));

      number_Of_Air_T=number_Of_Air_T+number_Of_Air_Pixels;
      number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;
      number_Of_Water_T=number_Of_Water_T+number_Of_Water_Pixels;

      end

      Threshold1a=sum(Threshold1a(1:b));
      Threshold1a=Threshold1a/(b);
```

```
      Threshold1a=round(Threshold1a);
      AT1=Threshold1a;
      AT1=num2str(AT1);
      str13=AT1;

      Threshold2a=sum(Threshold2a(1:b));
      Threshold2a=Threshold2a/(b);
      Threshold2a=round(Threshold2a);
      AT2=Threshold2a;
      AT2=num2str(AT2);
      str14=AT2;

      void_ratio =
(number_Of_Air_T+number_Of_Water_T)/number_Of_Solid_T;
      void_ratio=round(void_ratio,2);
      v=void_ratio;
      v=num2str(v);
      str15=v;

      degree_of_saturation =
(number_Of_Water_T/(number_Of_Air_T+number_Of_Water_T))*100;
      p=degree_of_saturation;
      p=round(p,2);
      p=num2str(p);
      str16=p;

      AH{5,1}='Refined Statistical-Based segmentation method complete.';
      AH{6,1}='All methods complete.';
      set(handles.listbox3, 'String', AH);
      drawnow();

      f = figure;
      set(f,'visible','off');
      r = cell(3,5);
      t=uitable(r);

      ColumnName = {'Segmentation Method','Void Ratio','Degree of
Saturation','Average Threshold One','Average Threshold Two'};
      d=
{O,str7,str8,str5,str6;P,str11,str12,str9,str10;KSW,str15,str16,str13,s
tr14};

      t.Data = d;
      t.Position = [20 200 400 150];

      A=[ColumnName ; d];
      filename='SegmentationData.xlsx';
      xlswrite(filename,A);

      msg = msgbox('Data is succesfully saved as
SegmentationData.xlsx');
      waitfor(msg);

      % --- Executes on selection change in listbox3.
```

```
      function listbox3_Callback(hObject, eventdata, handles)
      % hObject    handle to listbox3 (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)

      % Hints: contents = cellstr(get(hObject,'String')) returns
listbox3 contents as cell array
      %        contents{get(hObject,'Value')} returns selected item from
listbox3

      % --- Executes during object creation, after setting all
properties.
      function listbox3_CreateFcn(hObject, eventdata, handles)
      % hObject    handle to listbox3 (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    empty - handles not created until after all
CreateFcns called

      % Hint: listbox controls usually have a white background on
Windows.
      %       See ISPC and COMPUTER.
      if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
          set(hObject,'BackgroundColor','white');
      end

      % --------------------------------------------------------------
----
      function FileOpen_Callback(hObject, eventdata, handles)
      % hObject    handle to FileOpen (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)

      function New_Project_Callback(hObject, eventdata, handles)
      close gcf;
      ThreePhaseImageSegmentation;

      % --------------------------------------------------------------
----
      function CloseProgram_Callback(hObject, eventdata, handles)
      % hObject    handle to CloseProgram (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)
      close(gcf);

      % --------------------------------------------------------------
----
      function Segmented_Save_Callback(hObject, eventdata, handles)
      k = warndlg('Make sure to choose values for free/fitting
parameters k1 and k2!','Warning');
```

```
      waitfor(k);

      %%% OTSU

      number_Of_Void_T=0;
      number_Of_Solid_T=0;
      number_Of_Water_T=0;

      AA = uigetdir('','Please Select the Folder Containing the Cropped
Images');
      cd (AA);
      BB = mkdir('Otsu Segmented Images');
      CC = strcat(AA,'\Otsu Segmented Images')
      cd (AA);
      x = '*.png';
      b=dir(x);
      b=length(b);

      for ka = 1: b
          I = sprintf('CroppedImage_1.%02d.png', ka);
          I=imread(I);
          [Row,Col,r]=MaskPortion(I);
          sizeofmatrix = size(I,1);
          c=[];
          c1=zeros(sizeofmatrix,sizeofmatrix);
          c1=uint8(c1);
          e=0;
          for i=1:sizeofmatrix
              for j=1:sizeofmatrix
                  d=(j - (Row))^2 + (i - (Col))^2;
                  if d <= (r)^2
                      e=e+1;
                      c(e)=I(i,j);
                      c1(i,j)=I(i,j);
                  end
              end
          end

          e=uint8(c);
          p=imhist(e);
          mean=0;

          for i=1:256
          mean=mean+(i*p(i,1));
      end
      mean=mean/256;
      sumt=zeros(256,1);
      for i=1:256
          meanli=0;
          for j=1:i-1
              meanli=meanli+(j*p(j,1));
          end
          meanli=meanli/256;
          meangi=0;
```

```
    for j=i:256
        meangi=meangi+(j*p(j,1));
    end
    meangi=meangi/256;
    sumli=0;
    sumgi=0;
    for ii=1:i-1
        sumli=sumli+p(ii,1)*((meanli-mean)^2);
    end
    for ii=i:256
        sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
    end

    sumt(i,1)=sumgi+sumli;

end

Threshold1=find(sumt==max(sumt));

% Following lines of code are used to determine Threshold 2

pmax = p(Threshold1,1);
p2 = p(1:Threshold1);
mean=0;

for i=1:Threshold1
    mean=mean+(i*p2(i,1));
end

mean=mean/256;
sumt=zeros(256,1);

for i=1:Threshold1
    meanli=0;
    for j=1:Threshold1-1
        meanli=meanli+(j*p2(j,1));
    end
    meanli=meanli/256;
    meangi=0;
    for j=i:Threshold1
        meangi=meangi+(j*p2(j,1));
    end
    meangi=meangi/256;
    sumli=0;
    sumgi=0;
    for ii=1:Threshold1-1
        sumli=sumli+p2(ii,1)*((meanli-mean)^2);
    end
    for ii=i:Threshold1
        sumgi=sumgi+p2(ii,1)*((meangi-mean)^2);
    end

    sumt(i,1)=sumgi+sumli;
```

```
end

Threshold2=find(sumt==max(sumt));

c2=zeros(sizeofmatrix,sizeofmatrix);
c2=uint8(c2);
for i=1:sizeofmatrix
    for j=1:sizeofmatrix
        d=(j - (Row))^2 + (i - (Col))^2;
        if d <= (r)^2
            if c1(i,j)>Threshold1
                c2(i,j)=100;
            elseif c1(i,j) < Threshold2
                c2(i,j)=20;
            else
                c2(i,j) = 60;
            end
        end
    end
end

number_Of_Void_Pixels=sum(sum(c2==20));
number_Of_Solid_Pixels=sum(sum(c2==100));

number_Of_Void_T=number_Of_Void_T+number_Of_Void_Pixels;
number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;

QQ = 'Otsu Segmented Image_1.0';
cd (CC);
imwrite(c2,[QQ,num2str(ka),'.png']);
cd (AA);

end

%%% ITERATIVE OTSU

number_Of_Void_T=0;
number_Of_Solid_T=0;
number_Of_Water_T=0;


cd (AA);
BB = mkdir('Iterative Otsu Segmented Images');
CC = strcat(AA,'\Iterative Otsu Segmented Images')
cd (AA);
x = '*.png';
b=dir(x);
b=length(b);

for ka = 1: b
    I = sprintf('CroppedImage_1.%02d.png', ka);
    I=imread(I);
    [Row,Col,r]=MaskPortion(I);
    sizeofmatrix = size(I,1);
```

```
    c=[];
    c1=zeros(sizeofmatrix,sizeofmatrix);
    c1=uint8(c1);
    e=0;

    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
            d=(j - (Row))^2 + (i - (Col))^2;
            if d <= (r)^2
                e=e+1;
                c(e)=I(i,j);
                c1(i,j)=I(i,j);
            end
        end
    end

    e=uint8(c);
    p=imhist(e);
    mean=0;
    for i=1:256
    mean=mean+(i*p(i,1));
    end

    mean=mean/256;
    sumt=zeros(256,1);

    for i=1:256
        meanli=0;
        for j=1:i-1
            meanli=meanli+(j*p(j,1));
        end
        meanli=meanli/256;
        meangi=0;
        for j=i:256
            meangi=meangi+(j*p(j,1));
        end
        meangi=meangi/256;
        sumli=0;
        sumgi=0;
        for ii=1:i-1
            sumli=sumli+p(ii,1)*((meanli-mean)^2);
        end
        for ii=i:256
            sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
        end

        sumt(i,1)=sumgi+sumli;

    end

Threshold1=find(sumt==max(sumt));

[counts,N]=imhist(e);
counts1=counts(1:Threshold1);
```

```
      mu1=cumsum(counts1);   % To get the total summation of intensity
values
      N1 = (0:Threshold1-1)';
      mean=(sum(N1.*counts1))/mu1(end);
      mean=round(mean);
      IntensityRegion1=mean;

      counts2=counts(Threshold1+1:end);
      mu2=cumsum(counts2);   % To get the total summation of intensity
values
      N2 = (Threshold1:255)';
      mean=(sum(N2.*counts2))/mu2(end);
      mean=round(mean);
      IntensityRegion2=mean;

      Threshold2 = (IntensityRegion1+IntensityRegion2)/2;
      Threshold2 = round(Threshold2);

      % Iterative Process to Find the Ideal Value for Threshold2

      [counts,N]=imhist(e);
      counts3=counts(1:Threshold2);
      mu1=cumsum(counts3);   % To get the total summation of intensity
values
      N1 = (0:Threshold2-1)';
      mean=(sum(N1.*counts3))/mu1(end);
      mean=round(mean);
      IntensityRegion1=mean;

      counts4=counts(Threshold2+1:end);
      mu2=cumsum(counts4);   % To get the total summation of intensity
values
      N2 = (Threshold2:255)';
      mean=(sum(N2.*counts4))/mu2(end);
      mean=round(mean);
      IntensityRegion2 = mean;

      Thresholdnew = (IntensityRegion1+IntensityRegion2)/2;
      Thresholdnew = round(Thresholdnew);


      % Check if Thresholdnew is an Acceptable Choice for Threshold2

      loopCounter=1;

      while Thresholdnew-5 < Threshold2 && Thresholdnew+5 < Threshold2
          Threshold2=Thresholdnew;
          [counts,N]=imhist(e);
          counts5=counts(1:Threshold2);
          mu1=cumsum(counts5);   % To get the total summation of
intensity values
          N1 = (0:Threshold2-1)';
          mean=(sum(N1.*counts5))/mu1(end);
          mean=round(mean);
```

```
          IntensityRegion1=mean;

          counts6=counts(Threshold2+1:end);
          mu2=cumsum(counts6);   % To get the total summation of
intensity values
          N2 = (Threshold2:255)';
          mean=(sum(N2.*counts6))/mu2(end);
          mean=round(mean);
          IntensityRegion2=mean;

          Thresholdnew = (IntensityRegion1+IntensityRegion2)/2;
          Thresholdnew=round(Thresholdnew);

          if loopCounter>=10
              break;
          end

          loopCounter=loopCounter+1;
      end

      Threshold2=Thresholdnew;

      % Portion of Code that Calculates the Porosity and Void Ratio of
the Image

      c2=zeros(sizeofmatrix,sizeofmatrix);
      c2=uint8(c2);

      for i=1:sizeofmatrix
          for j=1:sizeofmatrix
              d=(j - (Row))^2 + (i - (Col))^2;
              if d <= (r)^2
                  if c1(i,j)>Threshold1
                      c2(i,j)=100;
                  elseif c1(i,j) < Threshold2
                      c2(i,j)=20;
                  else
                      c2(i,j) = 60;
                  end
              end
          end
      end

      number_Of_Void_Pixels=sum(sum(c2==20));
      number_Of_Solid_Pixels=sum(sum(c2==100));

      number_Of_Void_T=number_Of_Void_T+number_Of_Void_Pixels;
      number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;

      QQ = 'Iterative Otsu Segmented Image_1.0';
      cd (CC);
      imwrite(c2,[QQ,num2str(ka),'.png']);
      cd (AA);
      end
```

```
%%% REFINED STATISTICAL-BASED

number_Of_Void_T=0;
number_Of_Solid_T=0;
number_Of_Water_T=0;

k1= handles.k1;
k1=str2double(k1);
k2= handles.k2;
k2=str2double(k2);

cd (AA);
BB = mkdir('Refined Statistical-Based Segmented Images');
CC = strcat(AA,'\Refined Statistical-Based Segmented Images')
cd (AA);
x = '*.png';
b=dir(x);
b=length(b);

for ka = 1: b
    I = sprintf('CroppedImage_1.%02d.png', ka);
    I=imread(I);
    [Row,Col,r]=MaskPortion(I);
    sizeofmatrix = size(I,1);
    c=[];
    c1=zeros(sizeofmatrix,sizeofmatrix);
    c1=uint8(c1);
    e=0;
    for i=1:sizeofmatrix
        for j=1:sizeofmatrix
            d=(j - (Row))^2 + (i - (Col))^2;
            if d <= (r)^2
                e=e+1;
                c(e)=I(i,j);
                c1(i,j)=I(i,j);
            end
        end
    end

e=uint8(c);

% Automatic Threshold using Refined Statistical-Based Method

[counts,N]=imhist(e);
mu=cumsum(counts);   % To get the total summation of intensity
values
mean=(sum(N.*counts))/mu(end);
mean=round(mean);
variance=sum((counts-mean).^2)/(mu-1);
variance=sum(variance);
standarddeviation=sqrt(variance);
standarddeviation=round(standarddeviation);
```

```
      T1=mean-(k1*standarddeviation);
      T2=mean+(k2*standarddeviation);

      Threshold1=abs(T1+T2)/2;
      Threshold1=round(Threshold1);

      Threshold2=abs(T1-T2)/2;
      Threshold2=round(Threshold2);

      % Portion of Code that Calculates the Porosity and Void Ratio of
the Image

      c2=zeros(sizeofmatrix,sizeofmatrix);
      c2=uint8(c2);
      for i=1:sizeofmatrix
          for j=1:sizeofmatrix
              d=(j - (Row))^2 + (i - (Col))^2;
              if d <= (r)^2
                  if c1(i,j)>Threshold1
                      c2(i,j)=100;
                  elseif c1(i,j) < Threshold2
                      c2(i,j)=20;
                  else
                      c2(i,j) = 60;
                  end
              end
          end
      end

      number_Of_Void_Pixels=sum(sum(c2==20));
      number_Of_Solid_Pixels=sum(sum(c2==100));

      number_Of_Void_T=number_Of_Void_T+number_Of_Void_Pixels;
      number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;

      QQ = 'Refined Statistical-Based Segmented Image_1.0';
      cd (CC);
      imwrite(c2,[QQ,num2str(ka),'.png']);
      cd (AA);
      end

      h=msgbox('All Segmented Images Successfully Saved!');

      % ------------------------------------------------------------------
----
      function Tools_Callback(hObject, eventdata, handles)
      % hObject    handle to Tools (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)

      % ------------------------------------------------------------------
----
      function CAndB_Callback(hObject, eventdata, handles)
```

```
      axes(handles.axes4);
      imcontrast(gcf);

      % ------------------------------------------------------------------
----
      function Export_Callback(hObject, eventdata, handles)
      k = warndlg('Make sure to choose values for free/fitting
parameters k1 and k2!','Warning');
      waitfor(k);
      k1= handles.k1;
      k1=str2double(k1);
      k2= handles.k2;
      k2=str2double(k2);

      str1='Segmentation Method';
      O='Otsu';
      P= 'Iterative Otsu';
      KSW= 'Refined Statistical-Based';

      str2='Void Ratio';
      str3='Degree of Saturation';
      str3='Average Threshold One Value for all Images';
      str4='Average Threshold Two Value for all Images';

      d = uigetdir('*.png');
      cd(d);
      q=dir('*.png');
      b = numel(q);

      k = warndlg('Exporting of data can take a couple of minutes
depending on the number of images being analyzed. Please do not click
anything until data exporting is completed.','Warning');
      waitfor(k);

      %%% OTSU

      number_Of_Air_T=0;
      number_Of_Solid_T=0;
      number_Of_Water_T=0;

      for ka = 1: b
          I = sprintf('CroppedImage_1.%02d.png', ka);
          I=imread(I);
          [Row,Col,r]=MaskPortion(I);
          sizeofmatrix = size(I,1);
          c=[];
          c1=zeros(sizeofmatrix,sizeofmatrix);
          c1=uint8(c1);
          e=0;
          for i=1:sizeofmatrix
              for j=1:sizeofmatrix
                  d=(j - (Row))^2 + (i - (Col))^2;
                  if d <= (r)^2
                      e=e+1;
```

```
                c(e)=I(i,j);
                c1(i,j)=I(i,j);
            end
        end
    end

% Automatic Threshold using Otsu's Method

e=uint8(c);
p=imhist(e);
mean=0;

for i=1:256
    mean=mean+(i*p(i,1));
end

mean=mean/256;
sumt=zeros(256,1);

for i=1:256
    meanli=0;
    for j=1:i-1
        meanli=meanli+(j*p(j,1));
    end
    meanli=meanli/256;
    meangi=0;
    for j=i:256
        meangi=meangi+(j*p(j,1));
    end
    meangi=meangi/256;
    sumli=0;
    sumgi=0;
    for ii=1:i-1
        sumli=sumli+p(ii,1)*((meanli-mean)^2);
    end
    for ii=i:256
        sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
    end

    sumt(i,1)=sumgi+sumli;

end

Threshold1=find(sumt==max(sumt));
Threshold1a(ka,1)=find(sumt==max(sumt));

% Following lines of code are used to determine Threshold 2

pmax = p(Threshold1,1);
p2 = p(1:Threshold1);
mean=0;

for i=1:Threshold1
    mean=mean+(i*p2(i,1));
```

```
end

mean=mean/256;
sumt=zeros(256,1);

for i=1:Threshold1
    meanli=0;
    for j=1:Threshold1-1
        meanli=meanli+(j*p2(j,1));
    end
    meanli=meanli/256;
    meangi=0;
    for j=i:Threshold1
        meangi=meangi+(j*p2(j,1));
    end
    meangi=meangi/256;
    sumli=0;
    sumgi=0;
    for ii=1:Threshold1-1
        sumli=sumli+p2(ii,1)*((meanli-mean)^2);
    end
    for ii=i:Threshold1
        sumgi=sumgi+p2(ii,1)*((meangi-mean)^2);
    end

    sumt(i,1)=sumgi+sumli;

end

Threshold2=find(sumt==max(sumt));
Threshold2a(ka,1)=find(sumt==max(sumt));

c2=zeros(sizeofmatrix,sizeofmatrix);
c2=uint8(c2);

for i=1:sizeofmatrix
    for j=1:sizeofmatrix
        d=(j - (Row))^2 + (i - (Col))^2;
        if d <= (r)^2
            if c1(i,j)>Threshold1
                c2(i,j)=100;
            elseif c1(i,j) < Threshold2
                c2(i,j)=20;
            else
                c2(i,j) = 60;
            end
        end
    end
end

number_Of_Air_Pixels=sum(sum(c2==20));
number_Of_Solid_Pixels=sum(sum(c2==100));
number_Of_Water_Pixels=sum(sum(c2==60));
```

```
      number_Of_Air_T=number_Of_Air_T+number_Of_Air_Pixels;
      number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;
      number_Of_Water_T=number_Of_Water_T+number_Of_Water_Pixels;

      end

      Threshold1a=sum(Threshold1a(1:b));
      Threshold1a=Threshold1a/(b);
      Threshold1a=round(Threshold1a);
      AT1=Threshold1a;
      AT1=num2str(AT1);
      str5=AT1;

      Threshold2a=sum(Threshold2a(1:b));
      Threshold2a=Threshold2a/(b);
      Threshold2a=round(Threshold2a);
      AT2=Threshold2a;
      AT2=num2str(AT2);
      str6=AT2;

      void_ratio =
(number_Of_Air_T+number_Of_Water_T)/number_Of_Solid_T;
      void_ratio=round(void_ratio,2);
      v=void_ratio;
      v=num2str(v);
      str7=v;

      degree_of_saturation =
(number_Of_Water_T/(number_Of_Air_T+number_Of_Water_T))*100;
      p=degree_of_saturation;
      p=round(p,2);
      p=num2str(p);
      str8=p;

      AH=cell(6,1);
      AH{1,1}='Otsu segmentation method complete.';
      AH{2,1}='Two methods remain...';
      set(handles.listbox3, 'String', AH);
      drawnow();

      %%% Iterative Otsu

      number_Of_Air_T=0;
      number_Of_Solid_T=0;
      number_Of_Water_T=0;

      for ka = 1: b
          I = sprintf('CroppedImage_1.%02d.png', ka);
          I=imread(I);
          [Row,Col,r]=MaskPortion(I);
          sizeofmatrix = size(I,1);
          c=[];
          c1=zeros(sizeofmatrix,sizeofmatrix);
          c1=uint8(c1);
```

```
        e=0;

        for i=1:sizeofmatrix
            for j=1:sizeofmatrix
                d=(j - (Row))^2 + (i - (Col))^2;
                if d <= (r)^2
                    e=e+1;
                    c(e)=I(i,j);
                    c1(i,j)=I(i,j);
                end
            end
        end

        e=uint8(c);
        p=imhist(e);
        mean=0;

        for i=1:256
            mean=mean+(i*p(i,1));
        end
        mean=mean/256;
        sumt=zeros(256,1);
        for i=1:256
            meanli=0;
            for j=1:i-1
                meanli=meanli+(j*p(j,1));
            end
            meanli=meanli/256;
            meangi=0;
            for j=i:256
                meangi=meangi+(j*p(j,1));
            end
            meangi=meangi/256;
            sumli=0;
            sumgi=0;
            for ii=1:i-1
                sumli=sumli+p(ii,1)*((meanli-mean)^2);
            end
            for ii=i:256
                sumgi=sumgi+p(ii,1)*((meangi-mean)^2);
            end

            sumt(i,1)=sumgi+sumli;

    end

    Threshold1=find(sumt==max(sumt));
    Threshold1a(ka,1)=find(sumt==max(sumt));

    [counts,N]=imhist(e);
    counts1=counts(1:Threshold1);
    mu1=cumsum(counts1);   % To get the total summation of intensity
values
    N1 = (0:Threshold1-1)';
```

```
      mean=(sum(N1.*counts1))/mu1(end);
      mean=round(mean);
      IntensityRegion1=mean;

      counts2=counts(Threshold1+1:end);
      mu2=cumsum(counts2);   % To get the total summation of intensity
values
      N2 = (Threshold1:255)';
      mean=(sum(N2.*counts2))/mu2(end);
      mean=round(mean);
      IntensityRegion2=mean;

      Threshold2 = (IntensityRegion1+IntensityRegion2)/2;
      Threshold2=round(Threshold2);

      % Iterative Process to Find the Ideal Value for Threshold2

      [counts,N]=imhist(e);
      counts3=counts(1:Threshold2);
      mu1=cumsum(counts3);   % To get the total summation of intensity
values
      N1 = (0:Threshold2-1)';
      mean=(sum(N1.*counts3))/mu1(end);
      mean=round(mean);
      IntensityRegion1=mean;

      counts4=counts(Threshold2+1:end);
      mu2=cumsum(counts4);   % To get the total summation of intensity
values
      N2 = (Threshold2:255)';
      mean=(sum(N2.*counts4))/mu2(end);
      mean=round(mean);
      IntensityRegion2=mean;

      Thresholdnew = (IntensityRegion1+IntensityRegion2)/2;
      Thresholdnew=round(Thresholdnew);

      % Check if Thresholdnew is an Acceptable Choice for Threshold2

      loopCounter=1;

      while Thresholdnew-5 < Threshold2 && Thresholdnew+5 < Threshold2
          Threshold2=Thresholdnew;
          [counts,N]=imhist(e);
          counts5=counts(1:Threshold2);
          mu1=cumsum(counts5);   % To get the total summation of
intensity values
          N1 = (0:Threshold2-1)';
          mean=(sum(N1.*counts5))/mu1(end);
          mean=round(mean);
          IntensityRegion1=mean;

          counts6=counts(Threshold2+1:end);
          mu2=cumsum(counts6);   % To get the total summation of
```

```
intensity values
          N2 = (Threshold2:255)';
          mean=(sum(N2.*counts6))/mu2(end);
          mean=round(mean);
          IntensityRegion2=mean;

          Thresholdnew = (IntensityRegion1+IntensityRegion2)/2;
          Thresholdnew=round(Thresholdnew);

          if loopCounter>=10
              break;
          end

          loopCounter=loopCounter+1;
      end

      Threshold2=Thresholdnew;
      Threshold2a(ka,1)=Thresholdnew;

      c2=zeros(sizeofmatrix,sizeofmatrix);
      c2=uint8(c2);
      for i=1:sizeofmatrix
          for j=1:sizeofmatrix
              d=(j - (Row))^2 + (i - (Col))^2;
              if d <= (r)^2
                  if c1(i,j)>Threshold1
                      c2(i,j)=100;
                  elseif c1(i,j) < Threshold2
                      c2(i,j)=20;
                  else
                      c2(i,j) = 60;
                  end
              end
          end
      end

      number_Of_Air_Pixels=sum(sum(c2==20));
      number_Of_Solid_Pixels=sum(sum(c2==100));
      number_Of_Water_Pixels=sum(sum(c2==60));

      number_Of_Air_T=number_Of_Air_T+number_Of_Air_Pixels;
      number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;
      number_Of_Water_T=number_Of_Water_T+number_Of_Water_Pixels;

      end

      Threshold1a=sum(Threshold1a(1:b));
      Threshold1a=Threshold1a/(b);
      Threshold1a=round(Threshold1a);
      AT1=Threshold1a;
      AT1=num2str(AT1);
      str9=AT1;

      Threshold2a=sum(Threshold2a(1:b));
```

```
      Threshold2a=Threshold2a/(b);
      Threshold2a=round(Threshold2a);
      AT2=Threshold2a;
      AT2=num2str(AT2);
      str10=AT2;

      void_ratio =
(number_Of_Air_T+number_Of_Water_T)/number_Of_Solid_T;
      void_ratio=round(void_ratio,2);
      v=void_ratio;
      v=num2str(v);
      str11=v;

      degree_of_saturation =
(number_Of_Water_T/(number_Of_Air_T+number_Of_Water_T))*100;
      p=degree_of_saturation;
      p=round(p,2);
      p=num2str(p);
      str12=p;

      AH{3,1}='Iterative Otsu segmentation method complete.';
      AH{4,1}='One methods remain...';
      set(handles.listbox3, 'String', AH);
      drawnow();

      %%% REFINED STATISTICAL-BASED

      number_Of_Air_T=0;
      number_Of_Solid_T=0;
      number_Of_Water_T=0;

      for ka = 1: b
          I = sprintf('CroppedImage_1.%02d.png', ka);
          I=imread(I);
          [Row,Col,r]=MaskPortion(I);
          sizeofmatrix = size(I,1);
          c=[];
          c1=zeros(sizeofmatrix,sizeofmatrix);
          c1=uint8(c1);
          e=0;

          for i=1:sizeofmatrix
              for j=1:sizeofmatrix
                  d=(j - (Row))^2 + (i - (Col))^2;
                  if d <= (r)^2
                      e=e+1;
                      c(e)=I(i,j);
                      c1(i,j)=I(i,j);
                  end
              end
          end

      e=uint8(c);
```

```
      % Automatic Threshold using New 3 Phase

      [counts,N]=imhist(e);
      mu=cumsum(counts);   % To get the total summation of intensity
values
      mean=(sum(N.*counts))/mu(end);
      mean=round(mean);
      variance=sum((counts-mean).^2)/(mu-1);
      variance=sum(variance);
      standarddeviation=sqrt(variance);
      standarddeviation=round(standarddeviation);

      T1=mean-(k1*standarddeviation);
      T2=mean+(k2*standarddeviation);

      Threshold1=abs(T1+T2)/2;
      Threshold1=round(Threshold1);
      Threshold1a(ka,1)=abs(T1+T2)/2;

      Threshold2=abs(T1-T2)/2;
      Threshold2=round(Threshold2);
      Threshold2a(ka,1)=abs(T1-T2)/2;

      c2=zeros(sizeofmatrix,sizeofmatrix);
      c2=uint8(c2);

      for i=1:sizeofmatrix
          for j=1:sizeofmatrix
              d=(j - (Row))^2 + (i - (Col))^2;
              if d <= (r)^2
                  if c1(i,j)>Threshold1
                      c2(i,j)=100;
                  elseif c1(i,j) < Threshold2
                      c2(i,j)=20;
                  else
                      c2(i,j) = 60;
                  end
              end
          end
      end

      number_Of_Air_Pixels=sum(sum(c2==20));
      number_Of_Solid_Pixels=sum(sum(c2==100));
      number_Of_Water_Pixels=sum(sum(c2==60));

      number_Of_Air_T=number_Of_Air_T+number_Of_Air_Pixels;
      number_Of_Solid_T=number_Of_Solid_T+number_Of_Solid_Pixels;
      number_Of_Water_T=number_Of_Water_T+number_Of_Water_Pixels;

      end

      Threshold1a=sum(Threshold1a(1:b));
      Threshold1a=Threshold1a/(b);
      Threshold1a=round(Threshold1a);
```

```
      AT1=Threshold1a;
      AT1=num2str(AT1);
      str13=AT1;

      Threshold2a=sum(Threshold2a(1:b));
      Threshold2a=Threshold2a/(b);
      Threshold2a=round(Threshold2a);
      AT2=Threshold2a;
      AT2=num2str(AT2);
      str14=AT2;

      void_ratio =
(number_Of_Air_T+number_Of_Water_T)/number_Of_Solid_T;
      void_ratio=round(void_ratio,2);
      v=void_ratio;
      v=num2str(v);
      str15=v;

      degree_of_saturation =
(number_Of_Water_T/(number_Of_Air_T+number_Of_Water_T))*100;
      p=degree_of_saturation;
      p=round(p,2);
      p=num2str(p);
      str16=p;

      AH{5,1}='Refined Statistical-Based segmentation method complete.';
      AH{6,1}='All methods complete.';
      set(handles.listbox3, 'String', AH);
      drawnow();

      f =figure;
      set(f,'visible','off');
      r = cell(3,5);
      t=uitable(r);

      ColumnName = {'Segmentation Method','Void Ratio','Degree of
Saturation','Average Threshold One','Average Threshold Two'};
      d=
{O,str7,str8,str5,str6;P,str11,str12,str9,str10;KSW,str15,str16,str13,s
tr14};

      t.Data = d;
      t.Position = [20 200 400 150];

      A=[ColumnName ; d];
      filename='SegmentationData.xlsx';
      xlswrite(filename,A);

      msg = msgbox('Data is succesfully saved as
SegmentationData.xlsx');
      waitfor(msg);

      % ------------------------------------------------------------------
----
```

```
      function Help_Callback(hObject, eventdata, handles)
      % hObject    handle to Help (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)

      % ------------------------------------------------------------
----
      function User_Guide_Callback(hObject, eventdata, handles)
      % hObject    handle to User_Guide (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)
      h = msgbox({'For proper program usage, the following steps must be
used in the displayed order:';'';'Step 1: Click "Rename Images." This
button renames all of the images in the folder of interest. Make sure
that the original images are saved to a folder.';'';'Step 2: Click
"Crop All Images." This button crops the renamed images and saves the
cropped images to a folder of choice. In order to apply cropping,
please double click within the drawn ellipse.';'';'Step 3: Click "Load
Image." This button loads the first cropped image and its corresponding
histogram. If necessary, the contrast and brightness of the image can
be adjusted as well. The slider allows for the viewing of all of the
remaining cropped images.';'';'Step 4: After choosing a segmentation
method, click "Run." This button allows for the selection of all of the
cropped images. The void ratio (e), degree of saturation, and the two
average threshold values for all the cropped images are displayed. The
segmentation of the first cropped image is shown and the slider allows
for the viewing of the remaining cropped images with the chosen
segmentation method applied.';'';'Note: The "Export Data to Excel"
button saves the void ratio (e), degree of saturation, and the two
average threshold values for all images, per segmentation technique, to
an Excel file.' },'Program Instructions');

      % ------------------------------------------------------------
----
      function About_Us_Callback(hObject, eventdata, handles)
      % hObject    handle to About_Us (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    structure with handles and user data (see GUIDATA)

      h = msgbox({'University of Delaware';'Geotechnical
Engineering';'Civil and Environmental Engineering';'301 DuPont
Hall';'Newark, DE 19716';'USA'});

      % --- Executes during object creation, after setting all
properties.
      function edit5_CreateFcn(hObject, eventdata, handles)
      % hObject    handle to edit5 (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    empty - handles not created until after all
CreateFcns called
```

```
      % Hint: edit controls usually have a white background on Windows.
      %       See ISPC and COMPUTER.
      if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
          set(hObject,'BackgroundColor','white');
      end

      % --- Executes during object creation, after setting all
properties.
      function edit4_CreateFcn(hObject, eventdata, handles)
      % hObject    handle to edit4 (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    empty - handles not created until after all
CreateFcns called

      % Hint: edit controls usually have a white background on Windows.
      %       See ISPC and COMPUTER.
      if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
          set(hObject,'BackgroundColor','white');
      end

      % --- Executes during object creation, after setting all
properties.
      function imageSlider_CreateFcn(hObject, eventdata, handles)
      % hObject    handle to imageSlider (see GCBO)
      % eventdata  reserved - to be defined in a future version of
MATLAB
      % handles    empty - handles not created until after all
CreateFcns called

      % Hint: slider controls usually have a light gray background.
      if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
          set(hObject,'BackgroundColor',[.9 .9 .9]);
      end
```