Spring 2015

Development and Application of a Method for Unwrapping Single Images of Cylindrical Objects

Aaron Zosel
Montana Tech of the University of Montana

DEVELOPMENT AND APPLICATION OF A METHOD FOR UNWRAPPING SINGLE IMAGES OF CYLINDRICAL OBJECTS

by
Aaron R. Zosel

A Non-thesis Research Paper submitted as partial requirements for:

Master of Science Degree

Geosciences: Geological Engineering Option

Montana Tech of the University of Montana

2014

Abstract

The goal of this project was to create a software tool to unwrap digital images of cylindrical rock cores and also to create a tool to measure the accuracy of the unwrapping transform.

Measurements of an object can be taken directly from an image if the object is planar and the image plane of the camera is perpendicular to the object. If these conditions are met then picture scale allows a user to relate measurements in the image to a real world coordinate system after correcting for radial distortion. But if the object in the image is not planar or is not perpendicular to the image plane then additional corrections or geometric transformations must be applied to the object in the image.

The project is focused on creating a coded software solution, using digital images from an off the shelf-camera, to transform or unwrap single images of cylindrical rock cores so that all objects in the image are co-planar. In addition to tools created to unwrap digital images a second software tool was created to verify the accuracy of the unwrapping software.

The project is broken down into three major components. The first is a technique for taking pictures of rock core, or image acquisition. A single rock core is wrapped with a paper grid pattern that is used to determine a pixel scale for a series of images of a rock core sample. The grid wrapped core is placed in a carriage made of a set of two rollers. This carriage is then centered below a camera that is remotely controlled from a desktop computer. Two halogen lights are used to evenly illuminate the surface of the core and an image is taken. Once the grid pattern is photographed the pattern is removed and a series of images of the outer surface of the core are taken.

The next phase of the unwrapping is to take the original image and pass it through a series of software modules that will assist the user to perform a rectification of the original images. First the scale of each pixel in the image is found using images of the original grid pattern wrapped around the core. Next the core itself is isolated from other background objects present in each image. Then a geometric transform is applied to the image of the core, using the unique pixel scale, with the goal to unwrap each image so that all objects in the image are co-planar and no longer suffer any distortion.

In order to measure the effectiveness of this transform on rectifying all areas of the image, a second software tool was created. This software tool allows the user the ability to measure the spacing of gridlines in the original image of the grid wrapped core and compare these measurements to the same image after it has been unwrapped. The code returns a database of measurements for every gridline present in a image as well as different plots of the data for the user to make final conclusions as to the effectiveness of unwrapping on each area of the core images.

Keywords: Image Rectification, Image Unwrapping, Developable Surfaces, Matlab, Image Analysis

Acknowledgements

This project began as a joint venture between Montana Tech and the University of North Florida and my journey here to Montana would not have been possible without the support of both institutions.

I would like to express my sincere gratitude to the following people for all their support and guidance:

- Dr. Mary MacLaughlin – Dr. MacLaughlin gave me an opportunity to study at Montana Tech and provided guidance throughout the project.

- Dr. Nick Hudyma- Professor at University of North Florida. Dr. Hudyma has been a part of my engineering education from my undergrad through my masters, and I would not be here today without his support. Dr. Hudyma has always had time for my questions both in my academic and professional life, and I will always be grateful for his guidance.

- Graduate Committee: Chris Gammons, Butch Gerbrandt, and Larry Smith. My project committee has helped me to complete my education goals and I sincerely thank them.

I would like to acknowledge the National Science Foundation for funding under grant number CMMI-0555812.

Table of Contents

List of Figures

List of Tables

INTRODUCTION

A large void in a rock specimen that can be seen with the unaided eye is termed a macro-pore. It is understood that as macro-porosity of a sample of rock increases the strength and stiffness will decrease. This is intuitive as there will be less material to support a given load. What is less understood is the relationship between void size, shape, distribution, proximity and the strength and stiffness of the rock mass. A sample that has a macro-porosity of say 50% may contain a single larger void or several smaller voids and the physical relationships of the void or voids may affect the engineering properties of the rock to varying degrees.

The Large Voids Project, funded by the National Science Foundation in collaboration with Montana Tech and the University of North Florida, was born out of the Yucca Mountain Nuclear Waste Repository project located in southeast Nevada. The mountain is primarily composed of a macro-porous tuff, or welded ash, known as the Topopah Spring Tuff. The nature of the material made it difficult to sample and transport to a laboratory for testing and there was a need to better understand the *in situ* nature of the macro-pores present in the rock and their effects on both short and long term loading as tunneling would produce caverns for nuclear waste storage that needed to stand for thousands of years. (Erfourth, 2006)

Under the Large Voids Project, individual projects have studied rock-like materials created in the laboratory which controlled void shape, size, distribution and proximity to investigate the relationship of macro-porosity on material strength and stiffness. The original scope of this project was to test the strength and stiffness of an actual macro-porous rock after characterizing the void distribution of a sample using images of the outer surface of a rock core. Several images of the outer surface of the core would be unwrapped and stitched together to form a single planar image of the cores surface. Voids on the surface could then be measured in

the image and the size, shape, distribution and proximity could be characterized. This information could be compared to strength testing data to try to reveal a trend that might help classify different distributions of voids according to apparent strength.

Part of the original scope of this project was to create a method that incorporated digital images of the exterior of a rock core to quantify the characteristics of voids present on the surface of a rock core. The project was later modified to specifically focus on developing a tool to create a single unwrapped planar image of the outer surface of a cylindrical rock core.

Background

With the numerous advancements in technology, more processing power and less expensive electronic components, digital image acquisition and processing have become common in many industries and disciplines. Digital image acquisition and processing has applications in such fields as machine vision, quality assurance for manufacturing, and medical imaging. Digital camera technologies have led to the expansion of the use of digital images as a quantifiable measuring tool for a variety of applications such as the development of digital terrain models for open pit mining slopes, mineralogical assessment of thin sections of rock and in-situ digital image mapping of drill holes.

Core imaging of the full circumference of drill cores is possible using the DMT CoreScan system to create full color unwrapped image of drill core with resolution of 0.2 mm. (Schepers et al., 2001) Sections of core from 0.2 to 200 mm in diameter and up to 1 meter in length are laid on set of rollers. The core scanner rotates the core along its long axis and incrementally scans the core using a digital line scanner. The core scanner rotates the core by the increment of a pixel between each scan. The individual line scans are combined to create a complete image of the

outer surface of the core. These systems are used for different applications such as geotechnical, oil and gas site investigation for exploration activities. Unwrapped images of each core section can be combined to produce a continuous image of the entire borehole. With additional information from drilling activities images of the core can be rotated to produce an oriented image of the borehole.

Other image techniques for the development of orthogonal projections of cylindrical objects have been developed mainly in the field of historical architectural documentation and preservation. These unwrapped images are used for archiving and analysis of cylindrical structures like historical towers. Several authors, listed below, have proposed single image techniques for unwrapping surfaces that can be approximated as cylinders.

Karras et al. (1996) proposed a technique that fits an image of a cylindrical object to a 3D cylindrical wire frame in a CAD program in order to get a set of XYZ coordinates for each pixel in the image. These coordinates were then used to map the 3D cylinder to a 2D unwrapped image. A cylindrical tower was photographed with several images. Several control points on the tower were surveyed and the overall dimensions of the tower were measured. These control points and dimensions were then used to create an analytical solid in a CAD program to which the images were fit using the control points. Images were cropped to only show the developable surface of the tower. Overlapping images, were stitched and the entire outer surface of the object was unwrapped.

Further work by Karras et al. (1997) proposes another technique in which images of a vaulted ceiling are unwrapped by fitting a cylinder to the control points measured in cross section across the ceiling. The cylinder provided the authors with an XYZ coordinate for each

pixel in the image and they were able to map these coordinates to a 2D space to produce an unwrapped image. The authors were able to compare the control point measures with points in the unwrapped image to evaluate the accuracy of the unwrapping by comparing "the root mean 'planimetric' deviation of the 2D similarity transformations between the raster and object developments" at several control points.

Hemmleb et al. and Pavelka et al. discuss the use of the calibrated camera model to solve for the Z coordinate of a pixel (X,Y) in the image when the object photographed can be approximated as a cylinder. These techniques find the position of the camera in space and use projective ray tracing to compute the distance of a point on the cylinder of an object in order to compute the Z coordinate of a pixel. A set of collinear equations govern the orientation of the camera with respect to the object. These equations are solved to find the position of a pixel in XYZ space to perform the image unwrapping.

Summary

Several techniques have been proposed to unwrap images of cylindrical objects. The more labor intensive methods require the use of surveyed control points on the object in order to fit the image to an XYZ space. The more elegant methods require a calibrated camera and do not require the surveyed control points.

Unwrapped images of rock core have many uses. Chief among them is the ability to measure strike and dip of joints, or layers present in a rock core.

Problem Statement

A three-dimensional object projected onto the two-dimensional image plane does not necessarily preserve the object's true shape. While there may be some portion, or plane, of the object that is parallel to the image plane, other portions will be askew and will suffer some distortion in the image.

The distortion prevents direct and accurate planar measurements of the objects projected to the image plane. This project presents method for correcting the distortions of a three dimensional cylindrical rock core as it is projected onto the two dimensional image plane.

Figure 1 shows an aluminum cylinder, wrapped with a square grid pattern that is lying on top of a green background. The pattern as applied to the cylinder has squares of equal area. Each line of the grid pattern is equidistant, parallel and horizontal lines meet perpendicular to vertical lines.

It may not be that immediately apparent but in Figure 1 the squares furthest away from the horizontal center line of the image are no longer completely square and are in fact now rectangular. While they maintain approximately the same width as squares near the center of the image, their height is reduced.

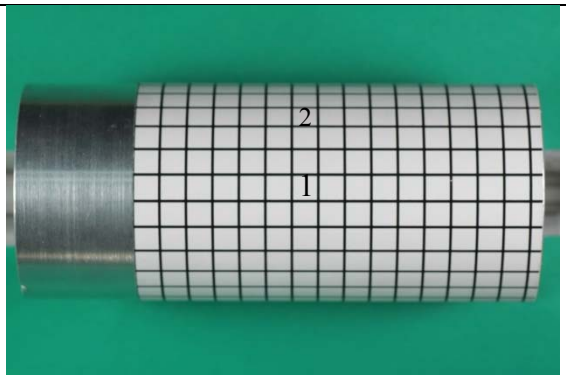


Figure 1: Aluminum cylinder wrapped with a square grid pattern.

A quick measure of the width and height of each numbered square in Figure 1, shows an almost 47% reduction in the height of square number two.

Square #	Height (number of rows)	Width (number of columns)
1	227	227
2	121	227

This simple exercise shows that direct planar measurement of surface features on the cylinder suffer a distortion. In order to make meaningful measurements, a correction has been developed and applied to the image of the cylinder in order to rectify the entire visible surface of the core to a common plane. Essentially, the correction will restore each of the squares visible in Figure 1 back to square instead of rectangular. This idea is similar to physically removing the label from a bottle and laying it flat on a counter. This "unwrapping" rectification produces a final image on which objects can be intuitively measured in a planar space. The unwrapping procedure does not account for radial distortion in the horizontal direction that must be corrected using methods that are not included in this study.

Unwrapping Procedure

In the previous section it was shown that the image of the core suffers from a distortion in the vertical direction of the image. A geometric transformation was developed to restore all portions of the image to a common plane. The following discussion outlines the geometric properties of the transformation and the application of this transform. The term 'unwrap' is an analogous term used to describe the geometric transformation.

Going back to Figure 1, the squares at the outer edge of the cylinder are much shorter in height. In order to restore these squares to their original shape, the squares should be elongated in the

vertical direction. It is assumed that each point on the surface of the core is projected orthogonally to the image plane.

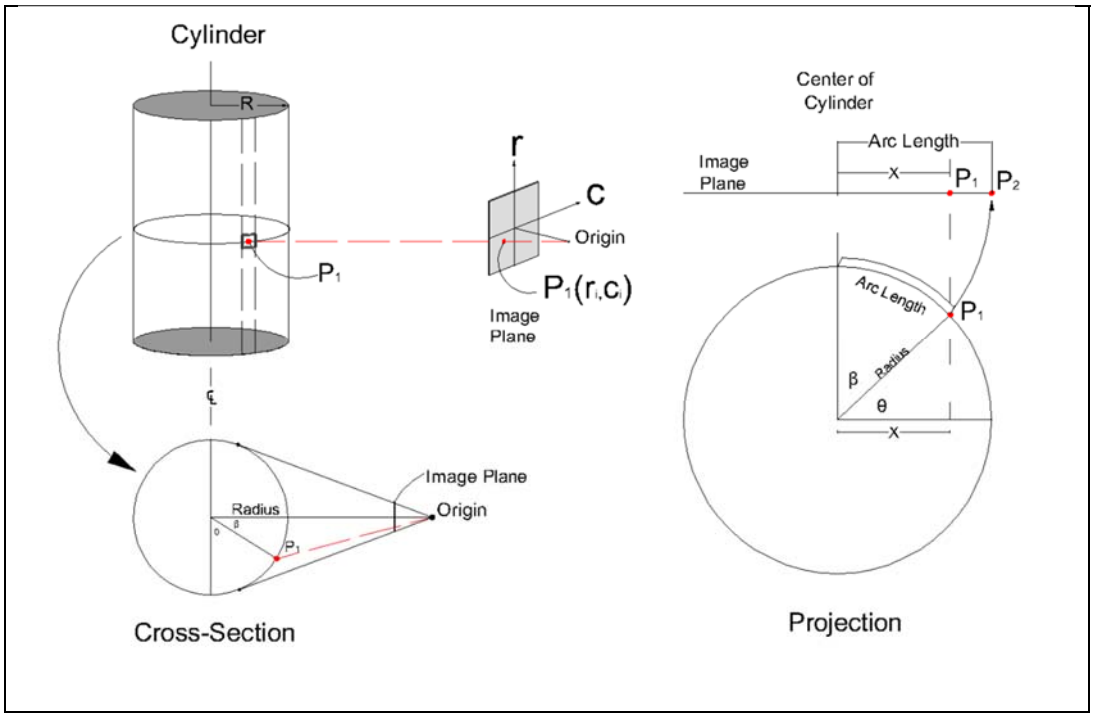


Figure 2: Cylinder Projection

Figure 2 shows the projection of point P_1 to the image plane. Distance X is the position of any pixel P in the image plane measured from the center of the cylinder. The center of the cylinder is not necessarily coincident with the origin of the image plane.

The goal of the unwrapping transformation is to move, or map, an image pixel at point P_1 to a new pixel at position P_2 . Point-P_2 represents the arc length from the center of the cylinder to point-P_1 along the surface of the cylinder.

A geometric transformation has been developed that can be applied to each pixel, at a distance X in the image to move, or map the pixel to a new position that represents the pixel's arc length, as shown in equation 1. This will essentially unwrap the image of a cylinder. The

geometric transformation formula is presented below and further discussion of the development is provided in the following document.

$$Unwrapped\ Position\ of\ a\ Pixel = Radius\ of\ Cylinder \times \left(\pi/2 - \text{acos}\left(\frac{Distance\ of\ pixel\ from\ center}{Radius\ of\ Cylinder}\right)\right) \quad (1)$$

The new location of a pixel is a function of the distance of the pixel from the center of the cylinder in the original image, the radius of the core. The radius of the core in the image is not known but the radius can be physically measured with a set of calipers in units of inches. This measure can be converted from units of inches to image units if the scale of a pixel in the image is found. Determining these three dependent variables comprises the bulk of the workflow to unwrap the original image of the cylindrical core.

Basic Digital Image Concepts

Digital camera sensors are arrays of photo receptors. Each photo receptor is covered by a filter designed to only allow in a specific wave length of light. Typically this light is either red, green or blue (RGB).

In order to develop a digital image, multiple receptors are combined and RGB values are interpolated for individual pixels in the digital image. The final digital image is stored as a three dimensional (M x N x 3) matrix of color pixels. M and N represent the row and column coordinates of each pixel. The value of a pixel at each location in the matrix is a integer between 1 and 256 which reflects the intensity of red, green or blue (RGB), Figure 3. When viewing an image, the RGB intensities are combined to create the unique color of an individual pixel in the image. This is considered a true color image (Gonzalez et. al, 2009).

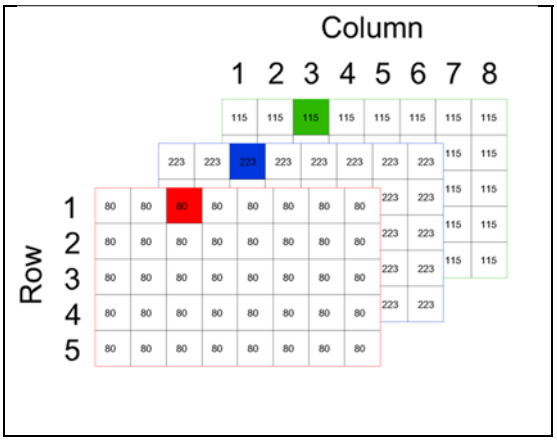


Figure 3: Typical RGB Image Matrix

A grayscale image combines the RGB components of the true color image by forming a weighted sum of the RGB components using the formula below.(Gonzalez et al., 2009) Pixel values in a grayscale image are also 1-256.

$$\text{Pixel value} = 0.2989xR + 0.5870xG + 0.1140xB$$

Image Coordinate System

A pixel has a color value and a location in a matrix. A color value is stored at specific cell in a matrix that has a row and column number, Figure 4. The row and column number are located at the center of the cell. Row and column numbers can only be whole numbers. The origin of the matrix is located at (0.5, 0.5). The cell located nearest to the origin has the row and column numbers of (1, 1).

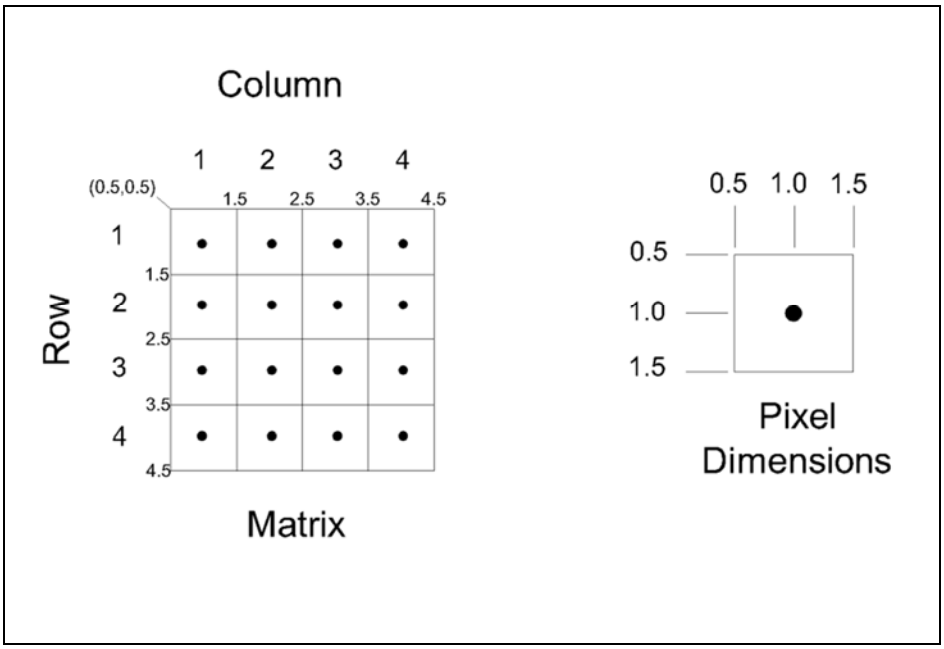


Figure 4: Image Coordinate System

Software tools

The coding portion of this project was created using the MathWorks program Matlab. Matlab is a high-level computer language that offers several specialized toolboxes that contain specific functions for use across a wide variety of engineering disciplines. The Matlab Image Processing Toolbox is a critical component of the software developed for this project.

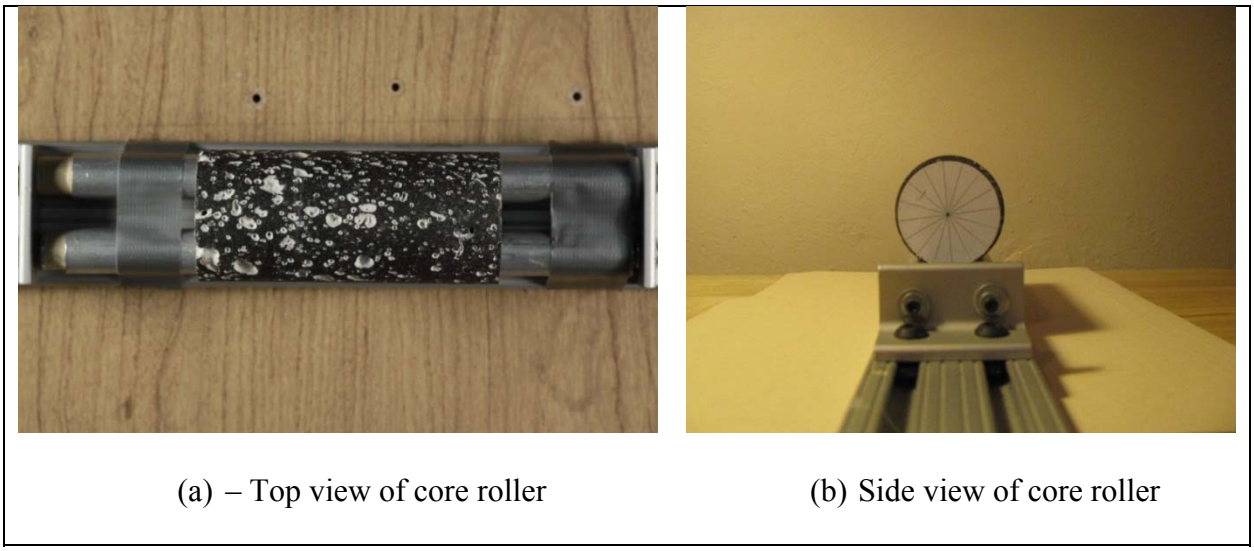
IMAGE ACQUISITION

Digital Camera

A Canon EOS50D camera with a fixed 90mm lens was used for image acquisition. The camera produces images with a pixel resolution of 3168 rows and 4752 columns. The camera allows the operator the flexibility to fix and control manual settings creating repeatability between images. In addition, the camera can be operated remotely from a computer via a USB cable. This operation eliminates any inadvertent movement of the camera between images that may be created by manually operating the camera shutter. The remote operation from the computer provided several options for adjusting the camera settings. It also provided a preview window of the image before it is taken. The preview window became an important part of the camera alignment since it offered a grid option that would overlay perpendicular and parallel lines that the core could be aligned with.

Core Stand

The core is placed on a set of rollers to prevent movement of the core during imaging. (Figure 5)



(a) – Top view of core roller (b) Side view of core roller

Figure 5: Core Roller

Core Alignment

Although the core roller fixes the core's relative vertical position between images, it does not by itself assure that the core edges are parallel to the upper and lower edges of the image. It was found that even the best alignment of the camera and core did produce some rotation of the core's long axis with respect to the upper and lower edges of the image. In addition to the rotation, the core itself is not exactly centered in the image with respect to all sides of the image.

Through trial and error several techniques were devised and then improved upon to reduce rotation and centering errors to improve camera/core alignment.

Camera Stand

Initially the camera was placed on a stack of books and aimed at the core from the side.

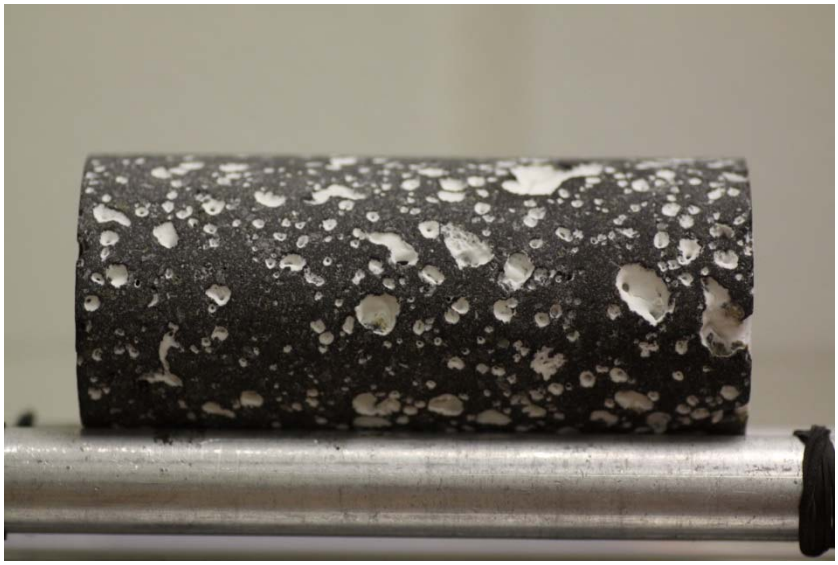


Figure 6: Camera looking at the core from the side.

This technique did not provide centering control for the core in the image. In addition, there was no way to control the alignment of the camera with the core surface. Either the left or right side of the core could be closer to the camera.

Shortly after the project began a camera stand was added to the acquisition process. Figure 7 shows the camera mounted on the stand looking down on the core, which rests on the core roller apparatus.

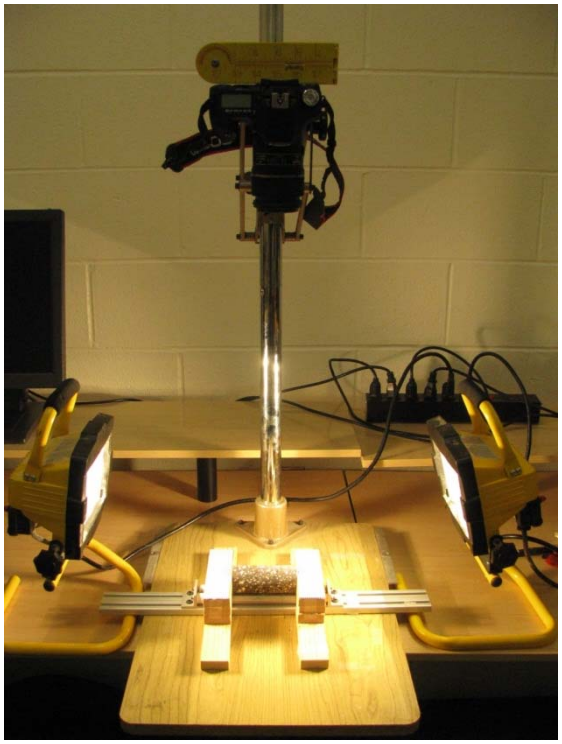


Figure 7: Camera Stand

The camera stand assures that the camera will remain at a fixed distance from the core throughout a series of images. A bubble level is placed on top of the camera to ensure that the camera is level and parallel to the surface of the core. The camera stand by itself does not ensure that the core will be centered in the image nor does it ensure that the core will not be rotated in the final image.

The bracket that the camera is mounted to is held in position on the chrome tube by a set screw, Figure 8.

Figure 8: Camera mounting bracket on camera stand

Once the screw is loosened the bracket is free to move up or down and it can also be rotated around the tube. The rotation of the bracket around the tube makes it difficult to align the bracket, and therefore the camera, parallel to the base. An attempt was made to string a plum bob from the center of the bracket down to the base. The bracket was then rotated about the chrome tube and a curved line was traced in pencil on the wood grained base. The idea was to attach the core roller to the base on a line drawn tangent to the arc. It proved very difficult to even draw a tangent line let alone re-align the camera bracket to the tangent line. In addition the chrome surface of the mounting tube does not hold fast the set screw, and just attaching the camera to the bracket would rotate the bracket off of the tangent line to some degree. Without a firm connection between the bracket set screw and the chrome tube the whole operation was abandoned in favor of a much more agile and adjustable alignment method.

Grid Pattern

An improvement to core alignment came with the addition of a grid pattern to the image series. A grid with 2 point lines spaced 0.125 inches, center to center, was printed out and wrapped around the core, see Figure 9.

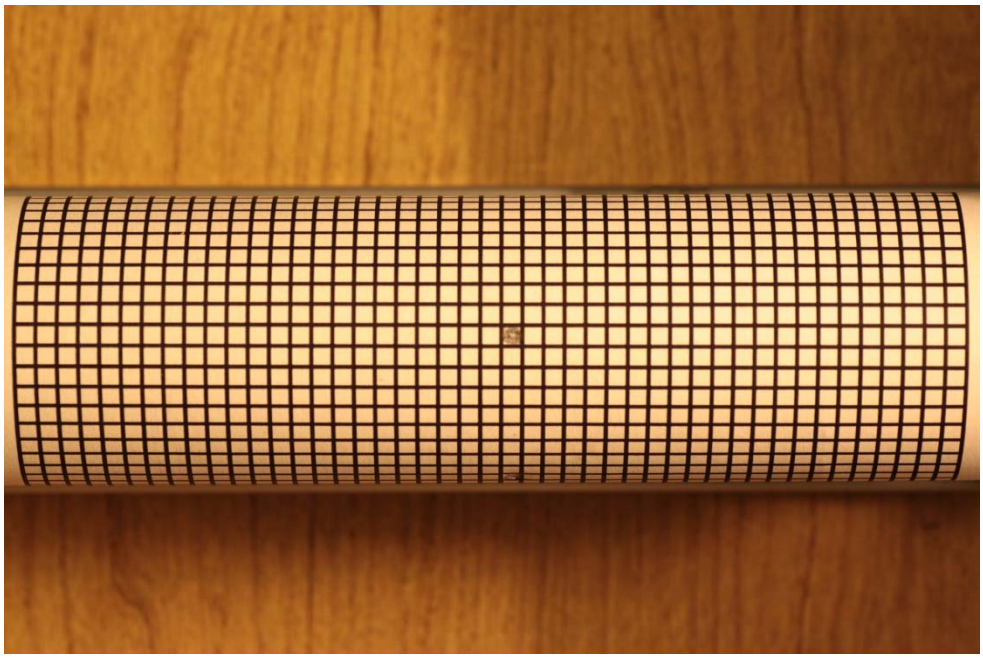


Figure 9: Grid pattern wrapped around rock core

The camera software provides a live preview window that is viewed on a computer monitor. The preview window allows the user an option to superimpose a grid over the live image currently being captured by the camera. The grid is shown highlighted in green in Figure 10.

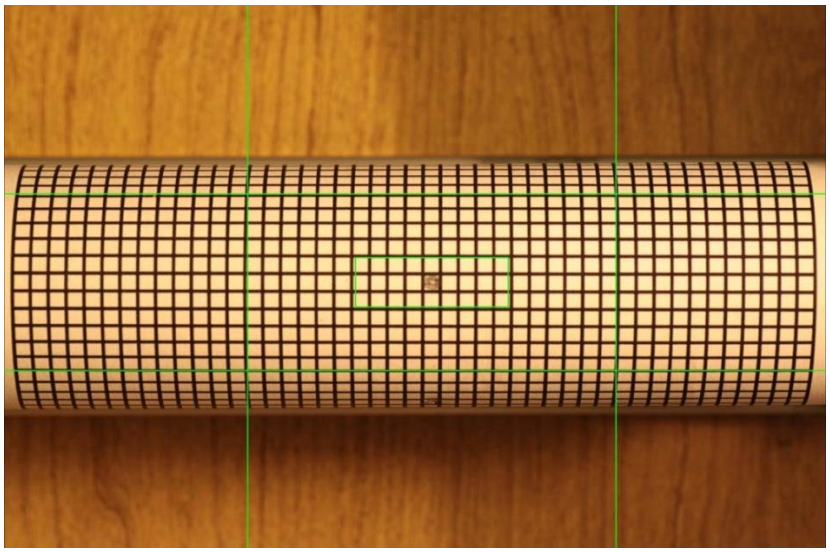


Figure 10: Gridded core with preview window grid superimposed

Although alignment was greatly improved, slight rotation of the core from horizontal still exists in most image series. Measures of the rotation average ± 0.20 degrees from horizontal.

The relatively low resolution of the preview window makes it nearly impossible to get the core absolutely square in the image. Figure 11 shows a typical image of the gridded core.

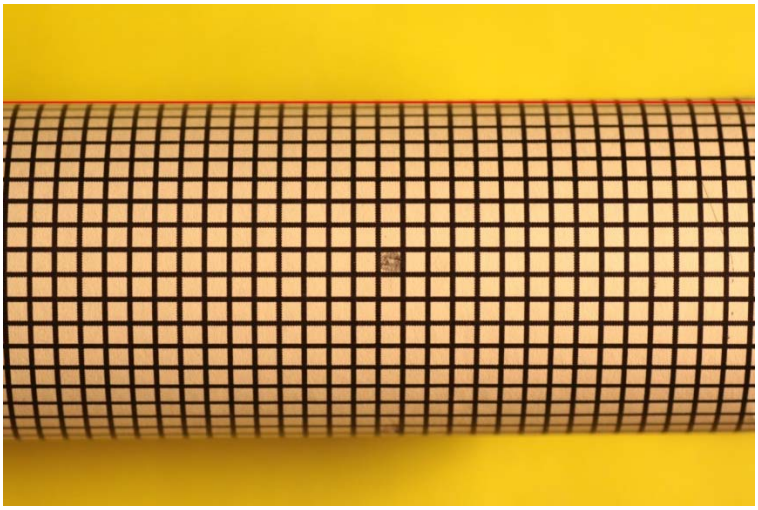


Figure 11: Counter clockwise rotation of the core in an image

The red line in the figure is superimposed on the image perpendicular to the vertical image borders. The red line approaches a lower gridline towards the right of the image. This indicates a positive rotation of the core from horizontal.

Lighting

The lighting of the core prevents shadowing of portions of the core and obscuring of voids on the surface of the core. Ambient overhead fluorescent lighting often flickers and creates inconsistent surface illumination between images. In order to eliminate any shadowing and inconsistencies between images, two halogen work lights were placed on both ends of the core as it was photographed. These lights provided even surface illumination between images. An example of uneven lighting can be seen in Figure 11 where only a single halogen light was used on the right hand side of the core. It is apparent that the single light creates a shadow on the lower left hand side of the image.

Colored Background

Edge detection of the core in the image is critical to the unwrapping process. To aid in edge detection a uniform background was added to the image acquisition process. The uniform colored background provides an easily identifiable contrast between the core and the background in an image.

A green background color was chosen to remain consistent with other studies being conducted outside of this project, Figure 12.

Figure 12: Core with green background

Summary of acquisition methods

Initially the core is wrapped with a grid pattern. This grid pattern is used in association with the camera preview window to center and align the core within the image. The camera is positioned so that the camera is looking straight down on the core, and the core is centered as best as possible in the center of the image.

UNWRAPPING

The following discussion will present procedures and methodology for unwrapping an image of a cylindrical core. A formula has been developed to map pixels in a wrapped image to an unwrapped image. This formula, or transform, requires several input parameters like the radius of the core, scale of a pixel in the core, and the location of the outer edges of the core in the image.

Radius of the core

One of the dependent variables of the unwrapping transform equation is the radius of the core. The radius is measured in inches and converted to units of rows (r) using the pixel scale(in/row) of the image. The diameter of the core is measured using a caliper. Measurements are taken at three locations along the core. At each location two measurements are taken at a 90 degree angle from each other for a total of six measurements. These measurements are averaged and the value is recorded, in inches, as the diameter of the core. The radius is then calculated and converted from the physical units of inches to the units of the image, rows.

$$Radius\ (row) = Radius\ (in)/Scale\ (in/row)$$

The entire surface of the core is not visible in a single image. The visible portion of a cylinder in an image is a function of the core radius and the distance of the focal point from the surface of the cylinder (Tanner, 2014).

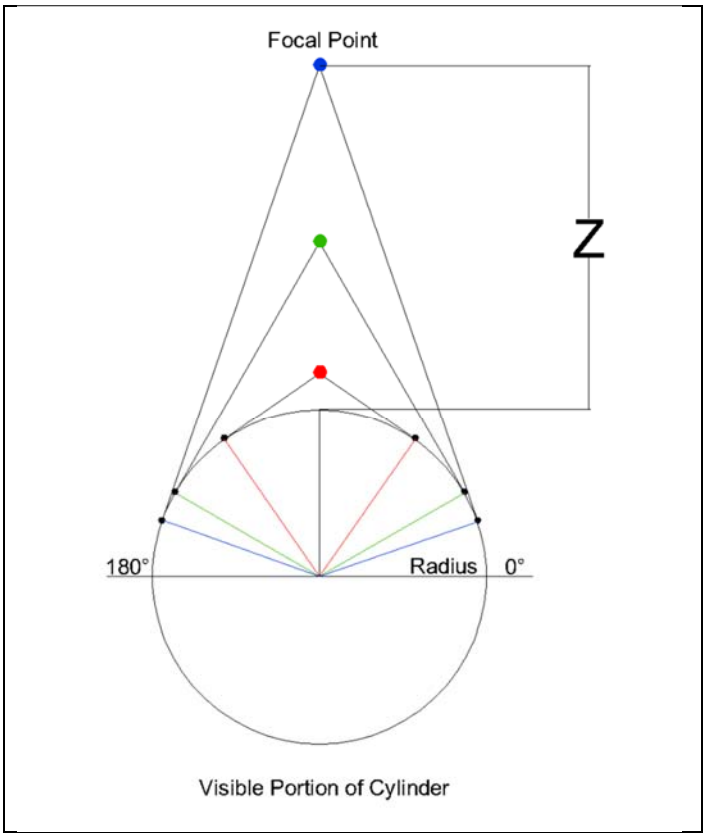


Figure 13: Visible portion of a cylinder surface in an image

The visible portion of the cylinder present in the image is calculated as:

$$Visible\ Portion = \frac{Z}{2\ (Radius + Z))} \times 360°$$

The camera focal point was measured for a few select image sets during image acquisition. Typical camera height and radius are 27.0625 (in) and 1.066 (in). For these parameters the visible portion of the core would be:

$$Visible\ Portion = \frac{27.0625}{2\ (1.066 + 27.0625))} = 0.48\ x\ 360° = \ 172.8°$$

This calculation is presented to show that a typical image of a cylindrical core does not represent 180° of the cylinder and no assumption should be made of the degree of the visible portion of the cylinder in the image.

Pixel scale

Introduction

As outlined in the introduction, one of the dependent variables necessary to implement the unwrapping transform equation is the pixel scale. The pixel scale is used to convert the measured radius of the core prior to substituting the radius in the unwrapping equation, below.

$$Radius\ (row) = Radius\ (in)/Scale\ (in/row)$$

$$Unwrapped\ Position = \ (Radius\ of\ Core\ (r) \times \left(\frac{\pi}{2} - acos\left(\frac{Distance\ of\ row\ from\ center\ (r)}{Radius\ of\ Core\ (r)}\right)\right)$$

The following discussion will show that it is possible to determine the pixel scale directly from the image by finding and comparing the spacing of two horizontal gridlines with a known physical measurement.

The scale is calculated as:

$$Scale = Know\ Gridline\ Spacing(inches)\ /Gridlines\ Spacing\ (rows)$$

Line Spacing

Several operations throughout the project are dependent on accurate identification and location of the center of the gridlines within an image. Measuring the distance between two lines

within an image may seem like a trivial exercise. The horizontal and vertical lines may appear to be well defined with crisp edges against the white background of the paper.

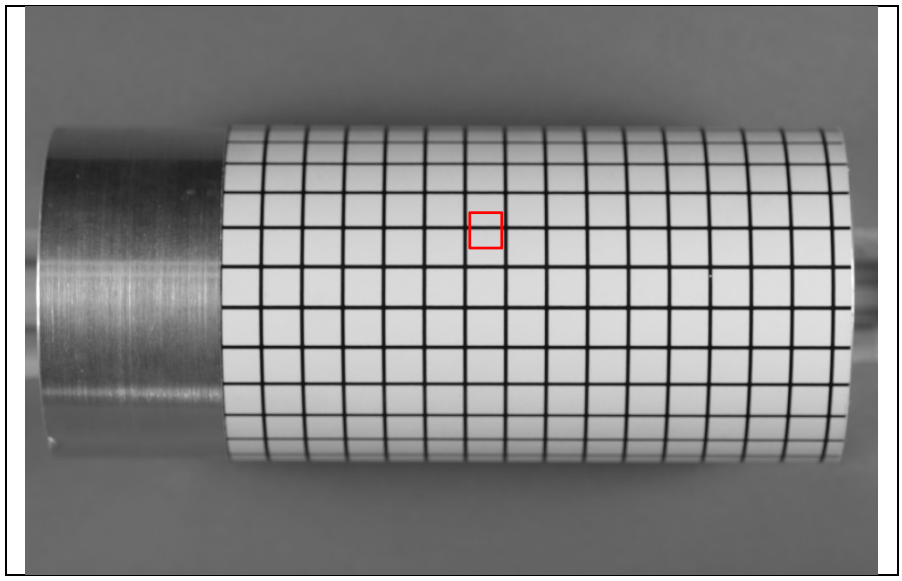


Figure 14: Typical Image of grid pattern wrapped on core. Red box outlines area for further discussion

When a line is viewed under magnification it becomes apparent that the edges of the line are not well defined, but there does seem to be a fairly symmetrical gradation of pixel values around darker pixels near the center of the line.

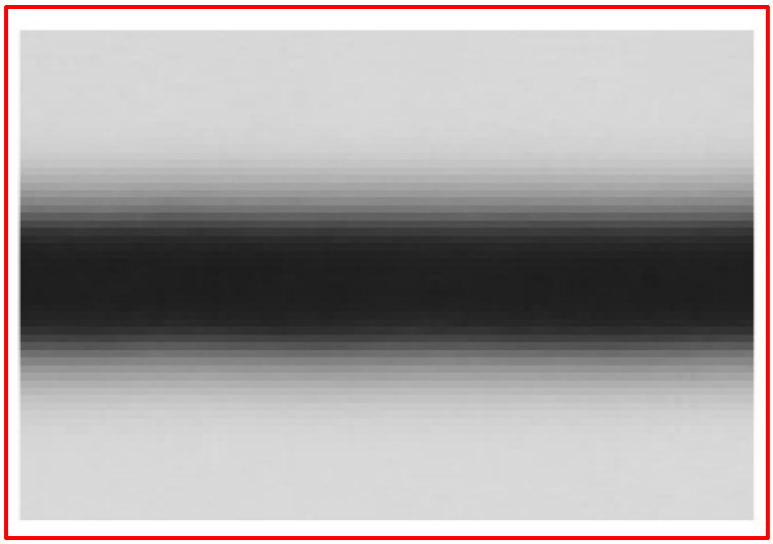


Figure 15: Magnified gridline segment from Figure 14.

The black and white grid pattern lines of Figure 16 were created in a drafting program and printed using 2 point lines with a center-to-center spacing of 0.25 inches. The black vertical line superimposed on the image is at the location of column 2567.

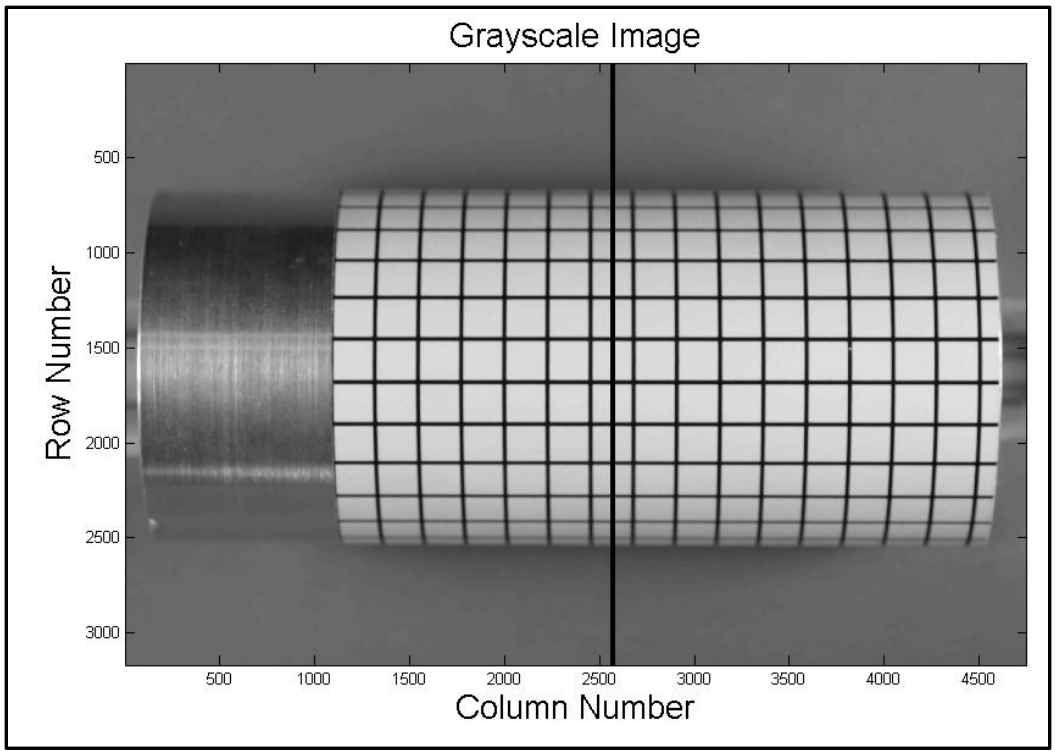


Figure 16: Grid Pattern with 0.25” spacing

The pixel values from column 2567 are extracted from the image in Figure 16 and plotted on the bar chart below. The pixel color values are plotted on the y-axis and the row number of the pixel is plotted on the x-axis. The eleven sharp valleys are the location of grid lines within the image. The two broader valleys on either side of the graph are at the boundaries between the edges of the core and the gray background, in the image.

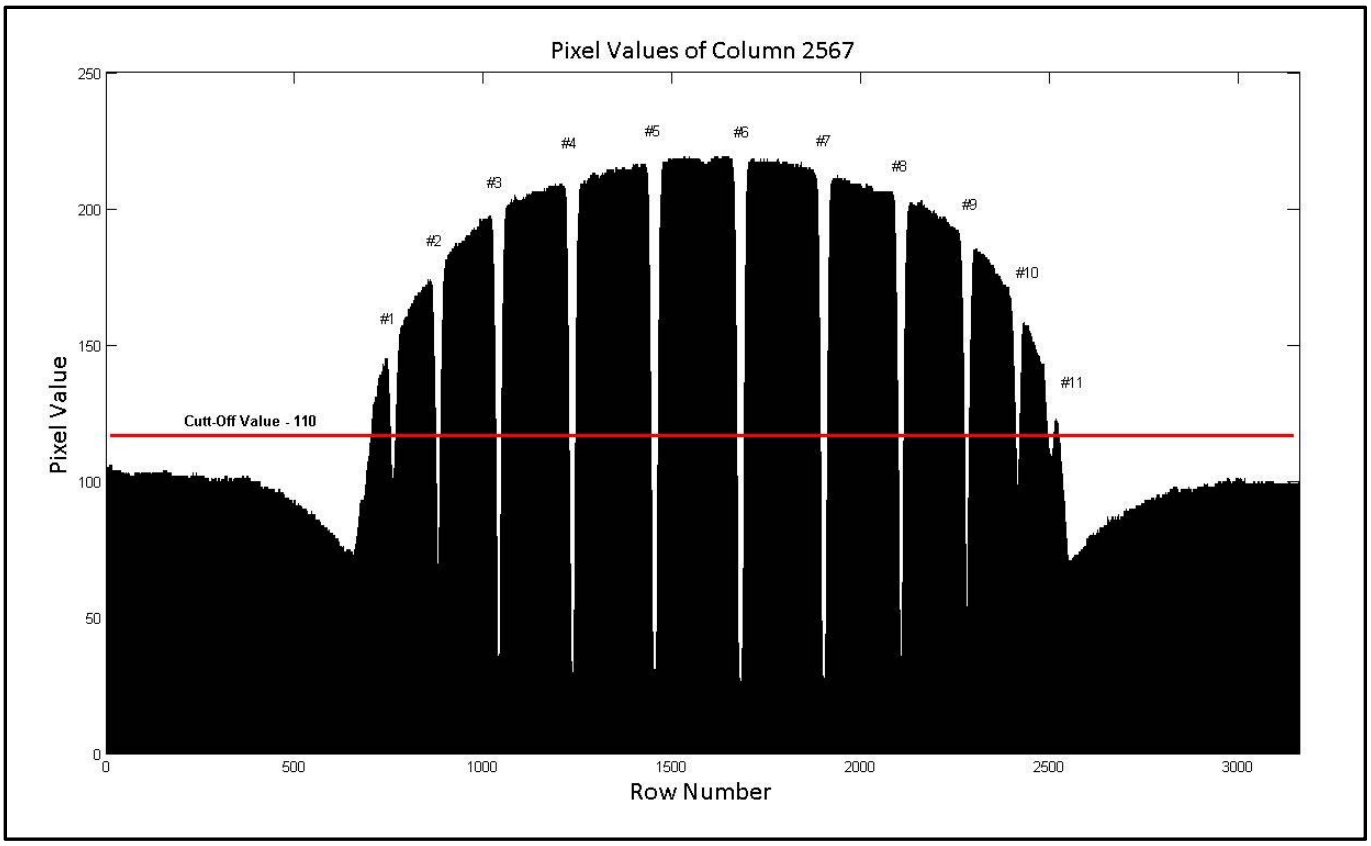


Figure 17: Pixel values of vertical line passing through horizontal gridlines graphed on a bar chart

Scanning from left to right in Figure 17, the darker colored gridlines appear to show a symmetrical decrease and increase in pixel values around a local minimum. Since the pixel values appear to be symmetrical about a point it would be possible to fit a parabola to the data around each gridline and solve an algebraic equation to find a local minimum or axis of symmetry about the center of the gridline. This would be a cumbersome process where a data set for each gridline would have to be individually selected, fitted with a parabola, and each equation of the parabola would be solved for the location of the axis of symmetry.

Another approach would be to simply select the pixel with a minimum value in the region of each gridline. A closer look at the pixel value gradation around each gridline shows that while

the data appears to be fairly symmetrical there is not necessarily a single minimum pixel value that could be identified as the center of the symmetry, Figure 18.

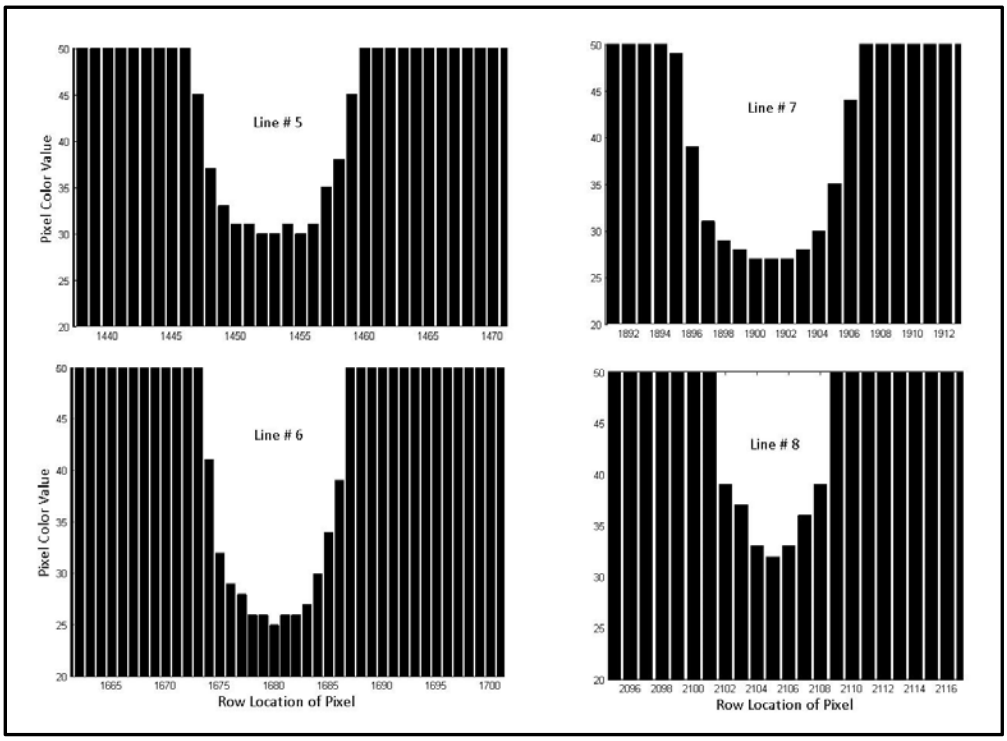


Figure 18: Local minimum of pixel values around select gridlines

While line numbers 6 and 8 do have a single minimum value that appears to be at the center of symmetry for each gridline, line numbers 5 and 7 show multiple pixels that have the same minimum values near an apparent center of the symmetry.

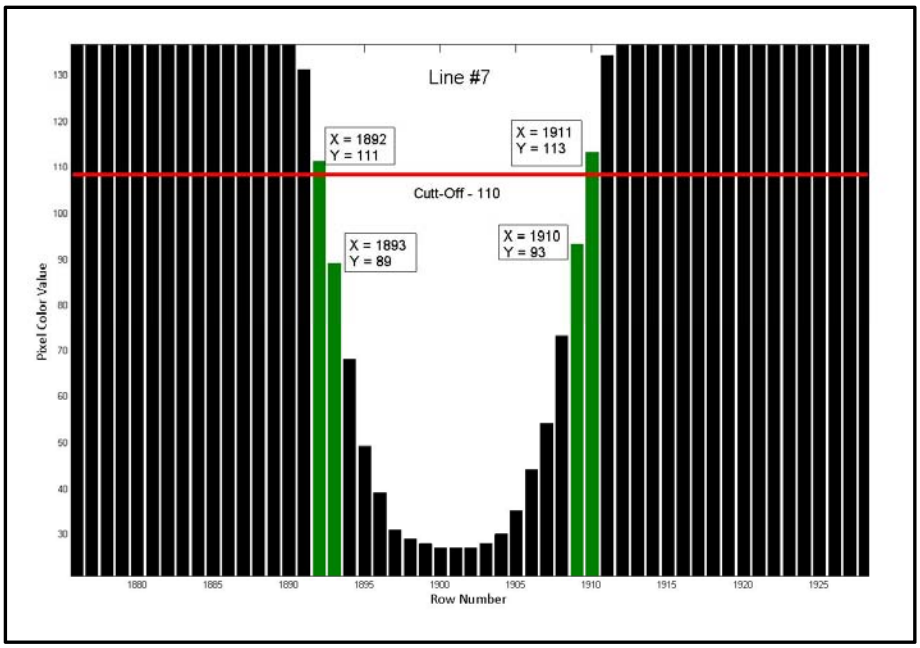


Figure 19: Gridline #7 showing cut-off values and corresponding row numbers in green

A closer inspection of line 7 shows that the difference in color values from one pixel to the next is not constant and any analysis that would look for a maximum gradient between pixel color values would not necessarily find a maximum gradient at or near the apparent center of the gridline since the color values near the center would have relatively small gradient values from one pixel to the next. Another approach to search for the axis of symmetry, or center of a gridline, is to divide the distance between two pixels selected on opposite sides of the apparent axis of symmetry to find the location of the center of the gridline.

In Figure 19, four of the pixels have been brushed green. These pixels have color values that are closest to a user specified color value called the cut-off value. If the shape of the bar chart is thought of as a valley then there would be two pixels on the left and two pixels on the right side of the valley. Using the cut-off value, the pixel values can be divided into two pairs with one pixel from each side of the valley. The first pair will be those two pixels with color values above the cut-off value, 110. The second pair will be those two pixels with values below

the cut-off value. Each pixel in a pair has a row location and the midpoint between these rows can be calculated as:

$$Midpoint\ Row\ \#\ =\ Left\ Side(row\#) + \frac{Right\ Side(row\#) - Left\ Side(row\#)}{2}$$

The following table shows the calculated midpoint between the two pairs of pixels identified in Figure 19.

Table I: Midpoint of Gridline

Cut-Off Value (110)	Left-Side		Right-Side		
	Pixel-Value	Row #	Pixel-Value	Row #	Midpoint
Above	111	1892	113	1911	1901.5
Below	89	1893	93	1910	1901.5

There appears to be a common midpoint between both pairs. The midpoint calculation is carried out for additional pairs of pixels present in gridline valley number 7. The cut-off value is varied between 199 and 39 by a value of one, Figure 20. Instead of selecting two pairs of pixels closest to each cut-off value, only a pair of pixels with color values below the cut-off is selected and the midpoint between the pair is calculated.

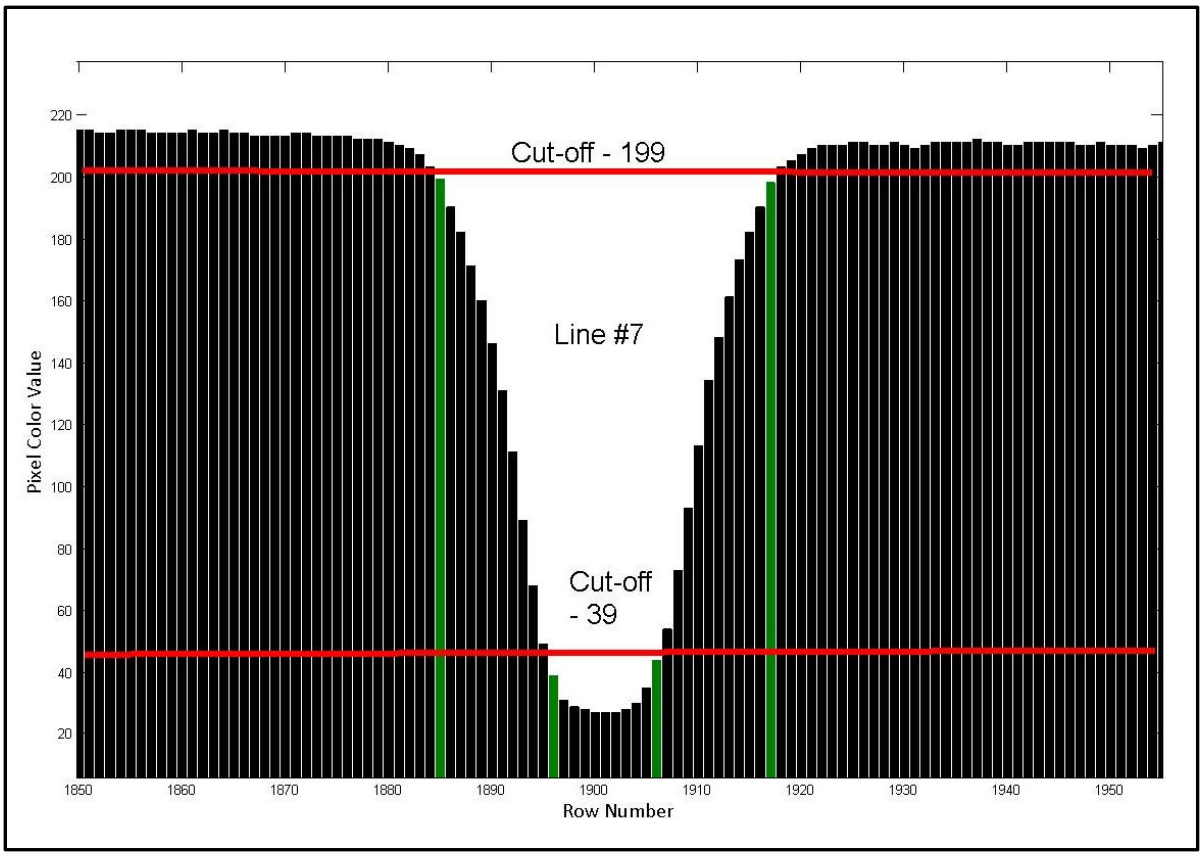


Figure 20: Cut-off range for analysis

The following table contains each of the midpoint values returned for a range of cut-off values and the occurrence of each value.

Table II: Cut-off range 199-39

Midpoint	1900.5	1901	1901.5
Count	23	133	5

For a user selected cut-off value, only one midpoint value is calculated and the midpoint can vary by ±1 row, between different cut-off values. The variation of the midpoint location occurs when one unique pixel, on either the right or left side of the valley, is paired with multiple pixels at separate cut-off values.

Table III: Pixel pairs

Cut-Off Value	Left-Side		Right-Side		
	Pixel-Value	Row #	Pixel-Value	Row #	Midpoint
185	181	1887	184	1915	1901
184	181	1887	172	1914	1900.5

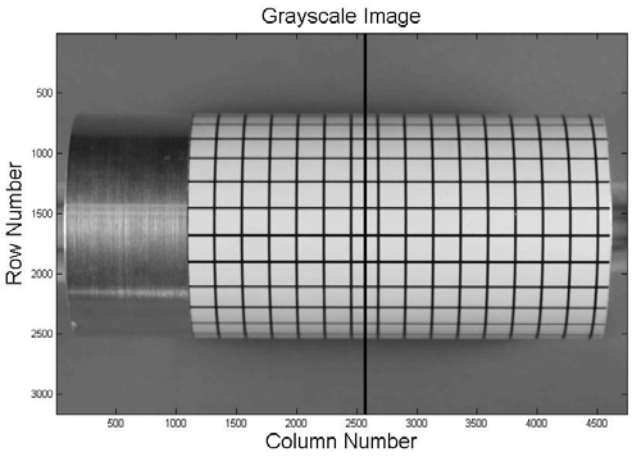
The pixels selected for each pair are only those pixels that have color values that are the closest to but less than the cut-off value.

Summary of midpoint selections

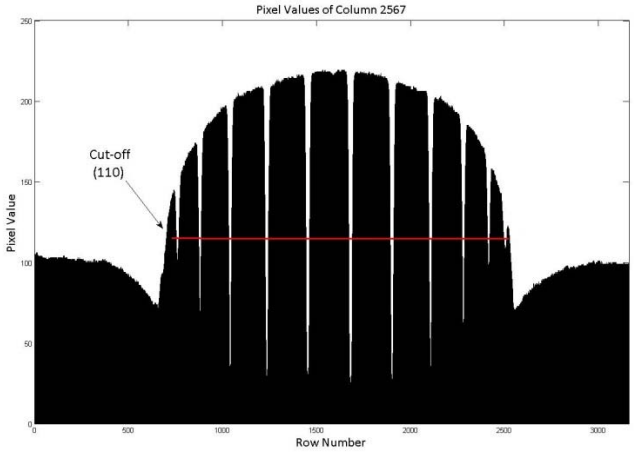
The pixel values within the range of the gridline appear to show a symmetrical distribution about a midpoint. A user selected cut-off value is used to pair pixels on either side of the gridline valley. The distance between the paired pixels is bisected to find the midpoint of the gridline. The calculated midpoint could vary by ± 1 row for a user selected cut-off values. These concepts have been adapted to create a semi-automated method for finding the midpoint of a gridline present in the image. This technique is referred to as the cut-off method.

Cut-off Method

The cut-off method code, Appendix A, first prompts the user to visually select a column of pixel data from an image.



(a)- User Selected Column



(b) – Bar Graph of pixel values from Column

Figure 21: Order of operations for cut-off method

The user selected column of data is plotted as a bar chart with the pixel color value on the y-axis, and the row number of the pixel on the x-axis. A horizontal red line superimposed on the bar chart is stretched or moved by the user so that it will intersect the apparent gridline valleys in the bar chart. The position of the line on the y-axis will become the user defined cut-off value. The ends of the lines are used to define a range within the column data that will be scanned to find pixel pairs for each of the gridlines.

Once the user defined cut-off and range are returned, the code begins scanning the data for two pixels whose color values are closest to but less than the cut-off value in each gridline valley. The pair will have a pixel located on the left and right side of the midpoint of the gridline valley. The data are scanned in order of ascending row value X. The location of the each pixel of the pair is defined as:

$$Left(x) = X_x + 1\,\{if\ Y_x\ \geq Cutoff\ \&\ Y_{x+1} < Cutoff\}$$

$$Right(x) = X_x - 1\,\{if\ Y_x\ \geq Cutoff\ \&\ Y_{x-1} < Cutoff\}$$

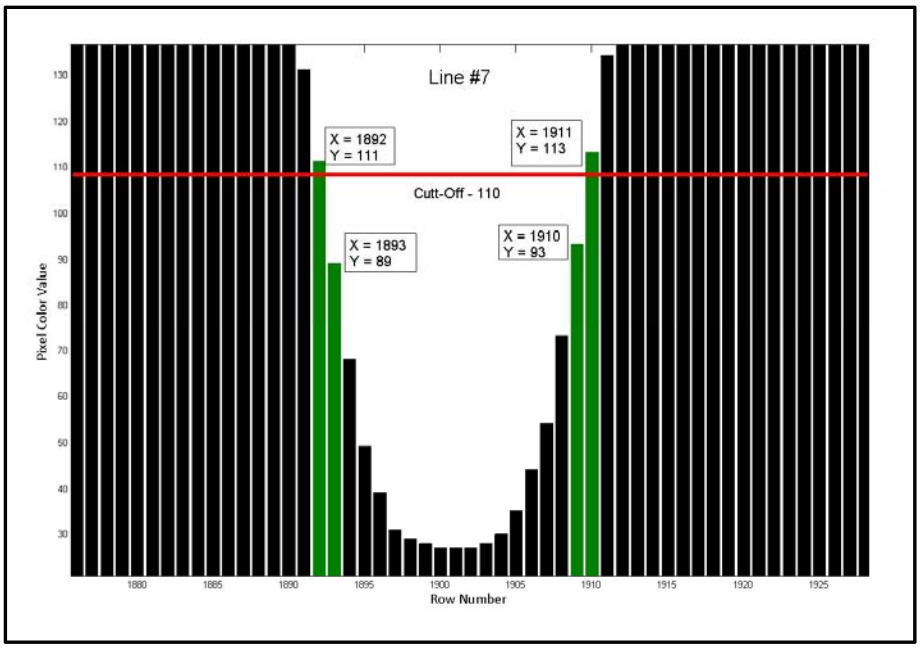


Figure 22: Returns for cut-off method using a pixel value of 110

The conditional statements above have been evaluated using the data from Figure 22.

$$Left(x) = 1892 + 1 = 1893\,\{since\ 111\ \geq 110\ \&\ 89 < 110\}$$

$$Right(x) = 1911 - 1 = 1910\,\{since\ 113\ \geq 110\ \&93 < 110\}$$

These conditional statements are applied over the entire user defined range of row values and the left and right side row number from each valley is returned. The coordinate pairs of rows are then bisected to find the row number of the midpoint of each gridline.

$$Midpoint\ of\ Line = Left + \left(\frac{Right - Left}{2}\right)$$

$$Midpoint\ of\ Line\ \#7 = 1893 + \left(\frac{1910-1893}{2}\right) = 1901.5$$

After all of the midpoints are calculated the code will return the original image. A gridline number is superimposed over the image at the midpoint of each gridline. The image

provides a visual check to see if the midpoints of each gridline were properly identified by the cut-off method.

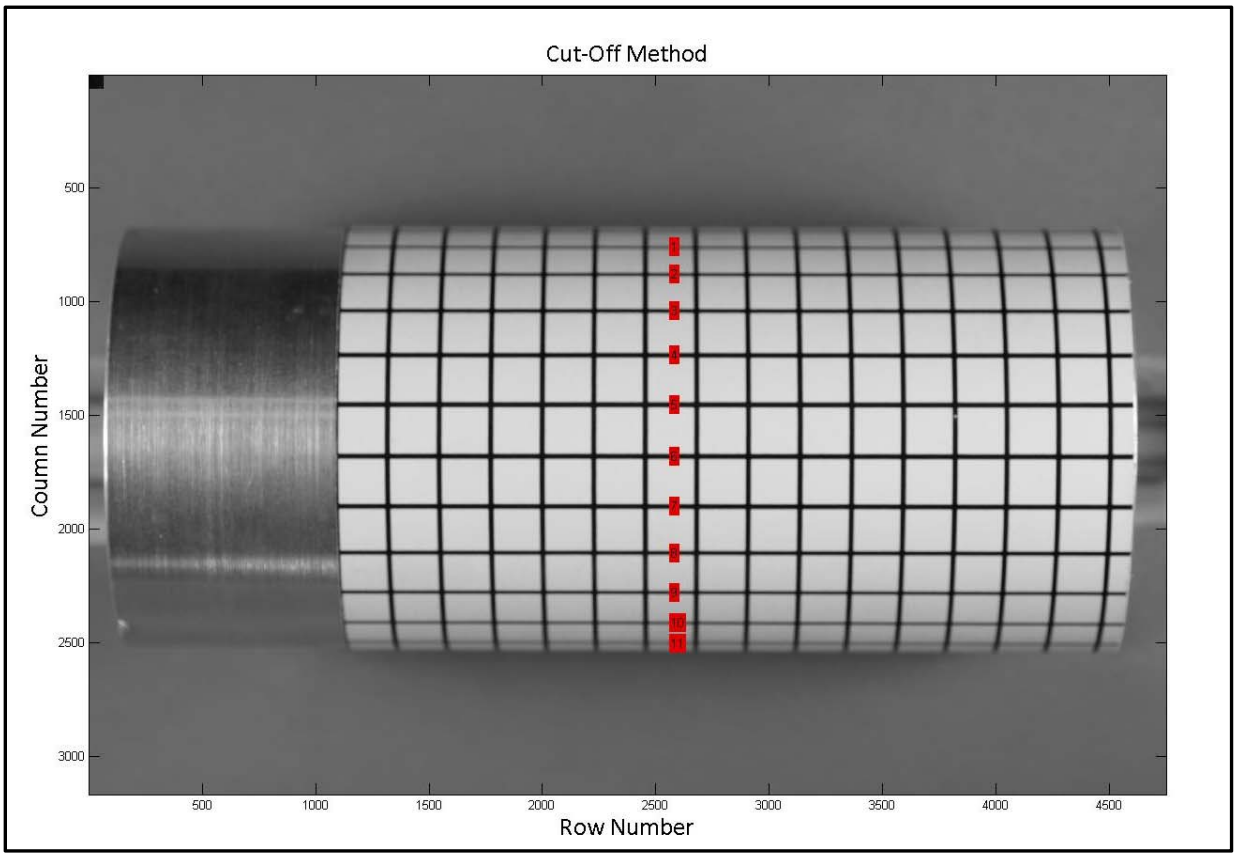


Figure 23: Visual check of cut-off method, red boxes indicate located gridlines

The location of the midpoint of the gridlines will be used for several different calculations throughout this project. The spacing between gridlines can be computed by bisecting the distance between the midpoints of each gridline. The spacing of the gridlines can be compared before and after applying the unwrapping transformation to an image. The spacing of two gridlines nearest the center of the core in the image can be used to calculate the scale of a pixel in the image. But before proceeding on to a discussion of how gridline spacing will be used to determine a pixel's scale, a short analysis of the effectiveness of the cut-off method is presented.

Cut-off Method Analysis

The previous discussion showed the development of the cut-off method using a grid pattern wrapped around the core. The curvature of the core warps the gridline spacing and lines are not evenly spaced. It is difficult to make an accurate prediction of the gridline location and spacing in this image. Without knowing the expected location and spacing of the gridlines in the image it is difficult to determine if the cut-off method is accurately locating the midpoint of each line in the image of the grid pattern wrapped on the cylindrical core.

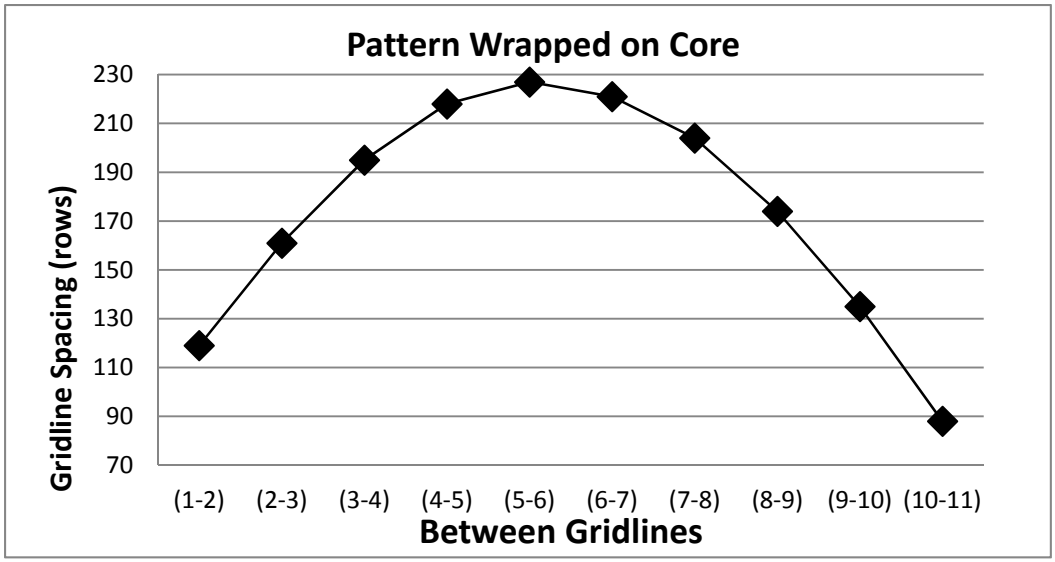


Figure 24: Number of rows between gridlines of pattern wrapped on a core

The same grid pattern laid flat in an image does not have the distorted effects of the wrapped grid. It is expected that the gridlines are evenly spaced in the image. The cut-off method should find the location of the midpoint of each gridline such that the spacing of each gridline should be equal. The results are shown below in Figure 25.

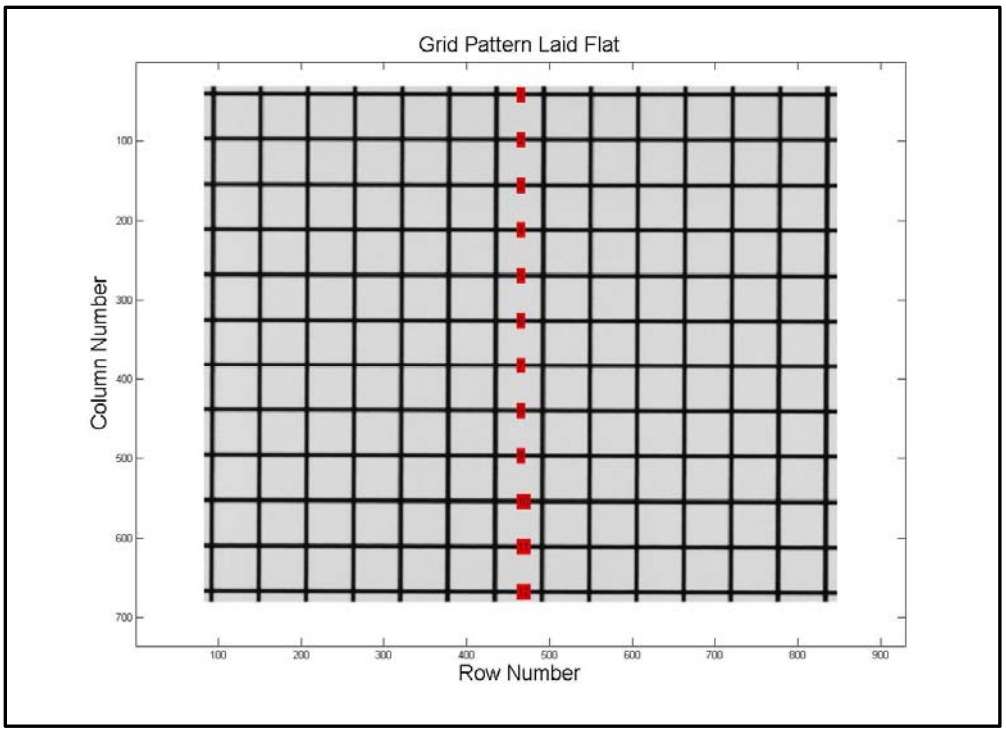


Figure 25: Flat grid pattern, red boxes indicate found gridlines

The cut-off method found the location of 12 gridlines in the image. The spacing between each gridline is calculated by subtracting the midpoint of a gridline by the location of the midpoint of the preceding gridline.

Table IV: Midpoint of lines

Line Number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Midpoint of line (row)	41	98	155	212	269	326	383	440	497	554	611	668
Spacing (# of rows)		57	57	57	57	57	57	57	57	57	57	57

The gridline spacing values are all equal and this was expected. This shows that the cut-off method accurately calculated the correct position of the midpoint of each line and the cut-off

method is an appropriate methodology for locating and measuring the spacing of gridline in an image.

Scale Determination

The grid pattern wrapped on the core in the image provides a reference for determining the scale of a pixel within the image. The pixel scale is controlled by the camera height but the camera height is not necessarily known.

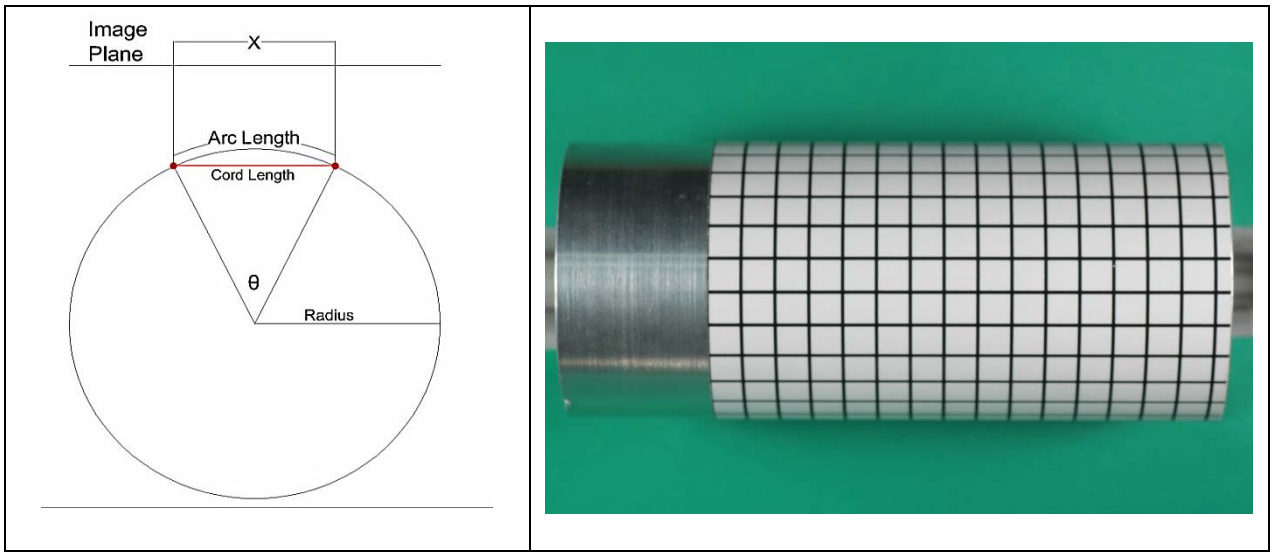


Figure 26: Cord length between gridlines wrapped on cylinder

For the image above, the pattern was printed with a gridline spacing of 0.25 inches. If the thickness of the paper is neglected than the gridlines on the wrapped core will demarcate an arc length of 0.25 inches. Since the radius and arc length are known a cord length can be calculated as:

$$Cord\ Length = 2\ \times Radius \times \sin(\theta\ /\ 2)$$

$$\theta = \frac{Arc\ Length}{2\pi Radius} \times 360°$$

The cylinder used in the image above has a radius of 1.062 inches. Assuming the arc length is 0.25 inches the cord length is found:

$$\theta = \frac{0.25}{2\pi(1.062)} \times 360° = 13.48°$$

$$Cord\ Length = 2\ \times 1.062 \times \sin(13.40° / 2) = 0.249$$

It is assumed that the gridlines are orthogonally projected to the image plane and it is assumed that the two horizontal gridlines nearest the center of the image will have negligible distortion caused by the cylindrical nature of the core. The two lines nearest the center will have a spacing that should be equal to the cord length, shown in the figure above. Using the cut-off method the spacing of two gridlines nearest the center of the image is found in units of rows. This spacing can be compared to the cord length to determine the scale of a pixel in the image.

$$Scale(in/pixel) = Cord\ length(in.)/Cord\ Length\ (rows)$$

The cord length is found in Figure 28 image (a) below. In addition the cord length will be compared with the spacing of a flat grid pattern that was photographed from the same distance as image (a) in Figure 28. This will show that there exists a spacing of gridlines in the wrapped image nearest to the center of the image that does not suffer distortion and can be used for the cord length in the scale calculation.

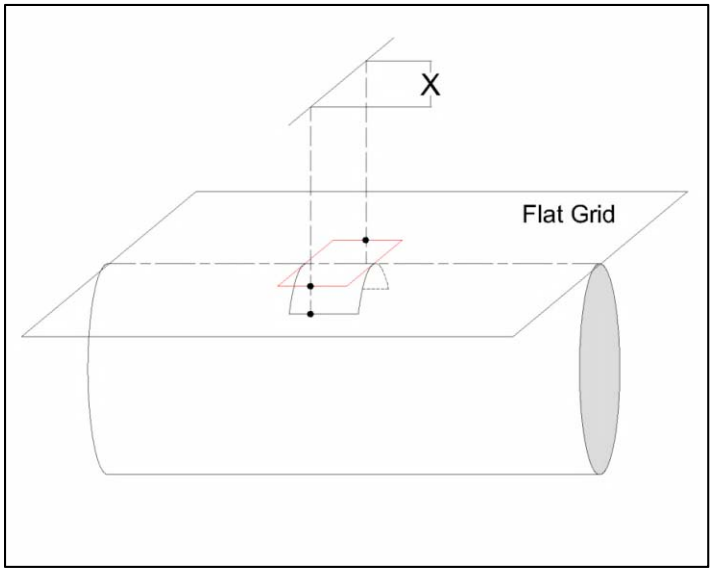
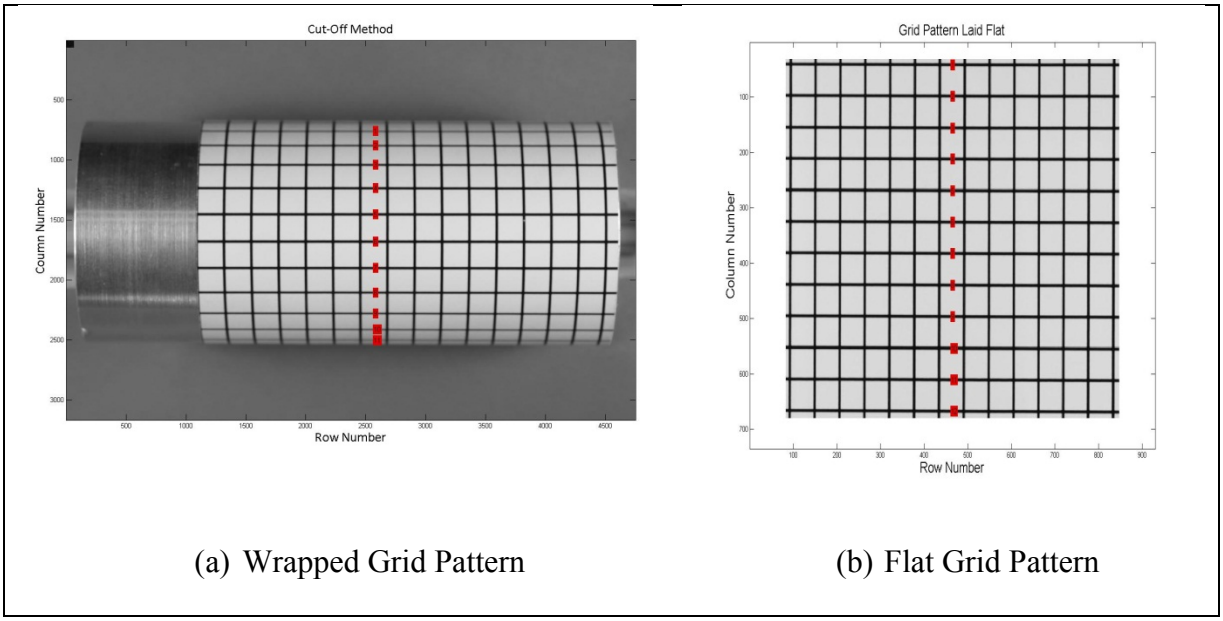


Figure 27: Curvature of grid pattern wrapped on a core



(a) Wrapped Grid Pattern (b) Flat Grid Pattern

Figure 28: Comparison of gridlines, photographed both wrapped and laid flat

The cut-off method is used to find the center of each line present in images a and b. The center-to-center spacing of each line is calculated and presented below.

	Line Number											
Line Spacing	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Flat Grid		228	228	226	228	228	228	227	228	228	229	228
Wrapped		119	161	195	218	227	221	204	174	135	88	

The spacing between gridlines 5-6 in the wrapped image has the maximum spacing value. This value is within ± 1 rows of the spacing of lines 5-6 of the flat pattern. This is a good fit since it was shown that the cut-off method will find the center of a line within ± 1 row. The spacing of lines 5-6 from the wrapped image will be used as the cord length (rows) in the scale calculation:

$$Scale(in/pixel) = 0.249(in.)/227\,(rows) = 0.0011\ in/pixel$$

Table V: Scale sensitivity

(+/-)Row	**Scale**	**Diff.**
-50	0.001407	0.000310
-10	0.001147	0.000051
-5	0.001122	0.000025
-1	0.001102	0.000005
227	0.001097	0
+1	0.001092	-0.000005
+5	0.001073	-0.000024
+10	0.001051	-0.000046
+50	0.000899	-0.000198

Table V shows that the scale is not that sensitive to variations in the gridline spacing, or cord length. The cut-off code is modified to find the scale of an image. The user visually selects a column and cut-off value from an image. The cut-off method returns the location of each gridline and the spacing between lines is calculated. The maximum spacing value is found and returned as the cord length (rows). The user specifies the original gridline spacing and then the core radius and the scale are calculated and stored for future use.

Conclusion

The cut-off method is a valid method for determining the midpoint of a gridline in an image. The line spacing can then be determined from these measurements and thus the scale of the image can also be determined.

The cut-off method also has other applications, and it will be used later on to help evaluate the accuracy of the image unwrapping method.

Core edge detection

Another of the dependent variables of the unwrapping equation is a given pixel's location with respect to the center of the core in the image. This location can be expressed as the distance of the pixel away from the center of the core. The distance away from the center is shown highlighted in the unwrapping transform equation below.

$$Unwrapped\ Position\ of\ a\ Pixel = Radius\ of\ Cylinder \times \left({}^{\pi}/_{2} - \operatorname{acos}\left(\frac{Distance\ of\ pixel\ from\ center}{Radius\ of\ Cylinder} \right) \right)$$

It is important to note that the arc-cosine function has a domain of -1 to 1. For the unwrapping equation this means that the 'Distance away from center' has to be less than or equal to the

'Radius of the core'. In other words, if a pixel is found to be outside of the radius of the core the unwrapping equation will fail.

In order to determine a pixel's distance from the center of the core, the center of the core must be located within the image. The image acquisition procedures do not necessarily align the center of the core with the center of the image. All attempts are made to try and make the center of the core coincident with the center of the image but the accuracy of this alignment cannot be assured. Because of this limitation, the edges of the core are used to determine the center of the core instead of assuming the center row of the image is also the center of the core.

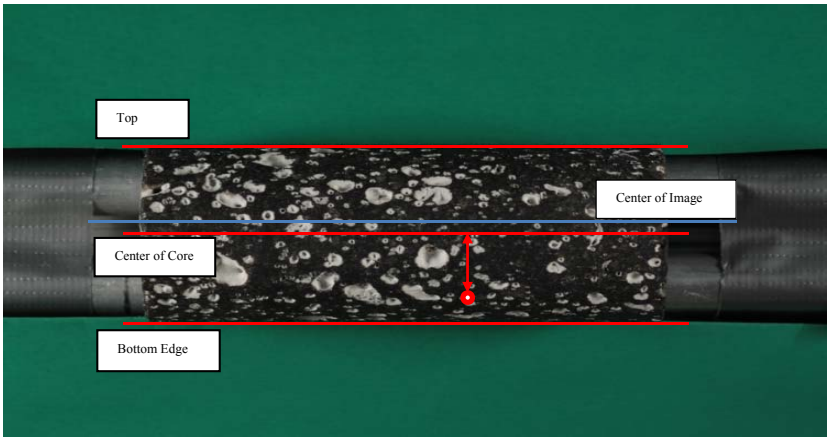


Figure 29: The center of the core is not coincident with center of image

The approach to finding the center of the core is to determine the row number of the horizontal edges of the core and then crop out the core from the original image. Once the core is cropped the resulting image can be bisected to determine the row location of the middle of the core. The preceding figure shows a typical image of a rock core, Figure 29. The key features labeled in the image are the top edge, center of image, center of core, and bottom edge. In this figure, the lines representing these features are drawn for illustrative purposes and are not the exact locations of these features.

The red circle illustrates a typical pixel and its relative location with respect to the center of the core. The purpose of finding the center of core in the image is ultimately to find the distance of any given pixel from the centerline of the core.

Finding the edges of the core is not a trivial exercise. Edge detection of the core is hampered by discontinuities in depth of field, differences in surface illumination, and subtle changes in textures across the image. It was shown in the preceding section that even a simple line does not have a clearly defined edge, and in order to find the center of a line a method was developed to take advantage of a unique gradient of pixel values that formed about the center of the line

Visual Edge Detection

The first attempt to determine the location of the core in the image, for edge detection, is to visually inspect the core using Matlab image viewing tools and determine the row coordinate of the horizontal edges of the core in the image. For the following discussions an image of an aluminum billet, wrapped with a grid pattern, on a green background is used throughout.

The image viewing function, imshow, is the primary image display function provided by the Matlab image processing toolbox. This function provides user tools for exploring image matrices. The two tools useful for edge detection are the image zoom capability and the object property tool. The object property tool allows the user to visually select a pixel and returns information about the individual pixel's location, column and row, and its color(s) value, Figure 30.

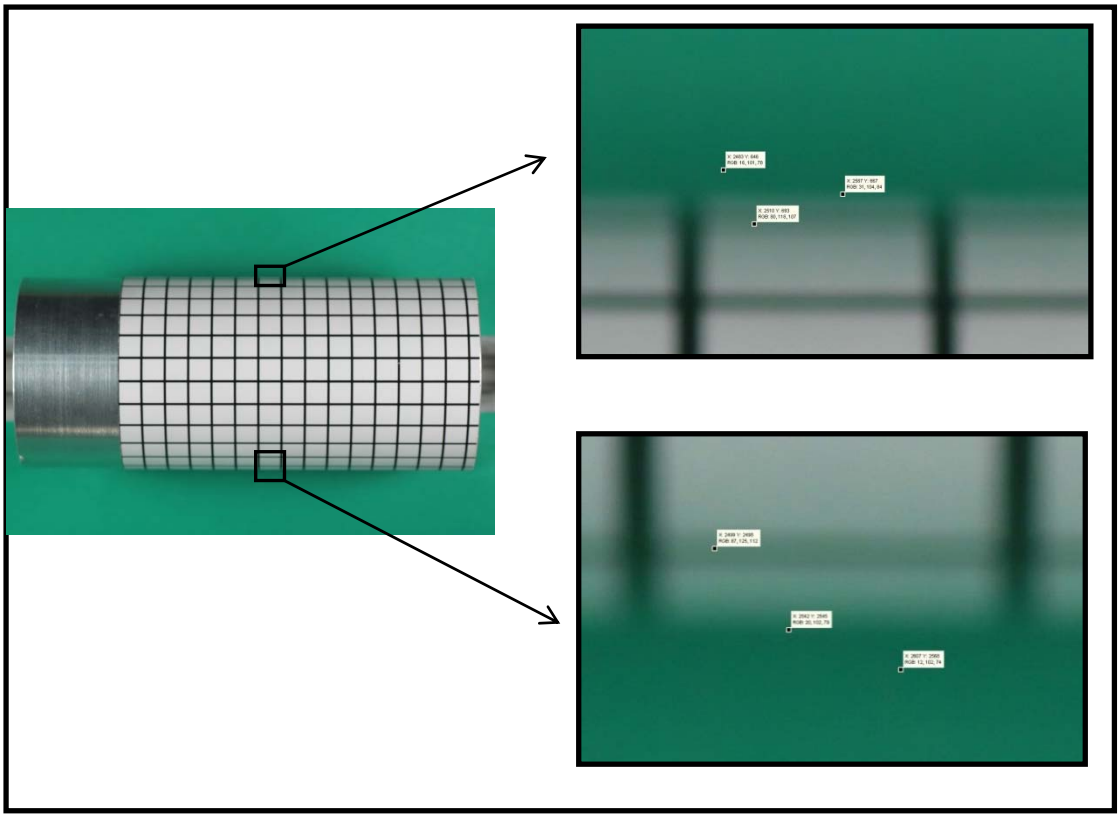


Figure 30: MatLab's imshow function, each white box contains pixel info: row, column, and RGB values

After the image has been read into the Matlab workspace, the imshow function is used to determine the apparent edges of the core by visual inspection.

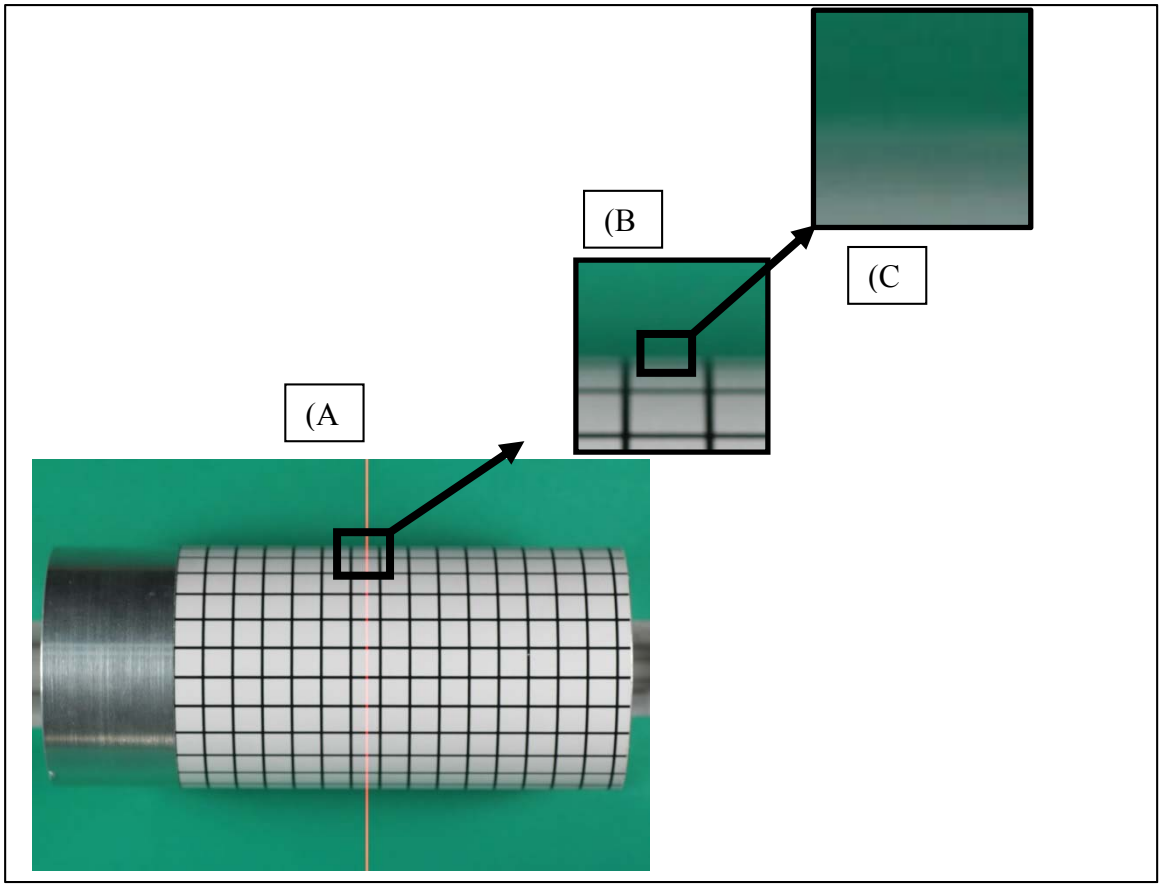


Figure 31: Visual Edge Detection,"By-Eye"

The first step is to zoom in on the edge of the core, as in sub-figurer C of Figure **31**. The next step is to use the pixel object tool to select a pixel that appears to be at the edge of the core. The row number of the apparent edge of the top of the core is recorded, Figure 32.

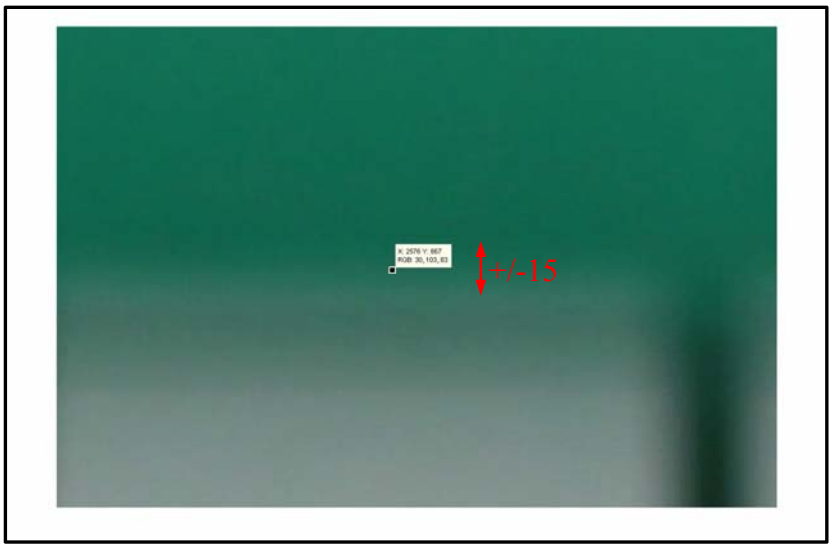


Figure 32: Visual Identification of the top edge

The process is repeated to determine the row number of the apparent edge of the bottom of the core. For the image of the aluminum billet the top and bottom edges of the core are at rows 667 and 2545, respectively. These rows are highlighted in the following figure.

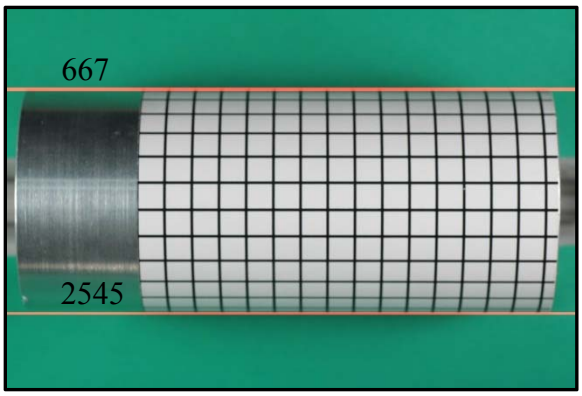


Figure 33: Top/Bottom Edge rows, 667 and 2545.

Depending on the image, visual inspection by the user can assure that the true edge of the core is probably within ±15 rows, Figure 32. This range was determined subjectively by the author after selecting two pixels. The first pixel appeared to be definitely of the green background. The second pixel appeared to be definitely of the white grid pattern on the core.

There is nothing empirical about this measurement, but it suggests that the edges of the core will probably be found within the range of ±15 rows rather than say ± 100 rows. This will be a useful range to keep in mind for later discussions of the automatic edge detection methods.

Conclusion

It is apparent from visual inspection of Figure **31** that the edge of the core is not well defined by a stark contrast in color as one visually scans from the green background to the black and white surface of the core. Edge detection of the core by visual inspection requires user judgment and therefore determining edge coordinates is not necessarily a repeatable exercise but it is a good estimate to start the following exploration of automatic edge detection methods.

Graphical Inspection

When the edge of the core is inspected visually there does not appear to be a stark contrast between the core and the background, Figure 32.

Rather than visually looking at the edge of the core for a pronounced edge it may be more informative to look closer at the pixel values as the values change from the colored background to the foreground core. The true color image was separated into its three matrices, red, green and blue. A vertical line, column, was chosen through the gridded core that only intersected horizontal lines of the grid. The pixel data for the line was "pulled" from the respective RGB matrices for the selected column 2567, Figure 34.

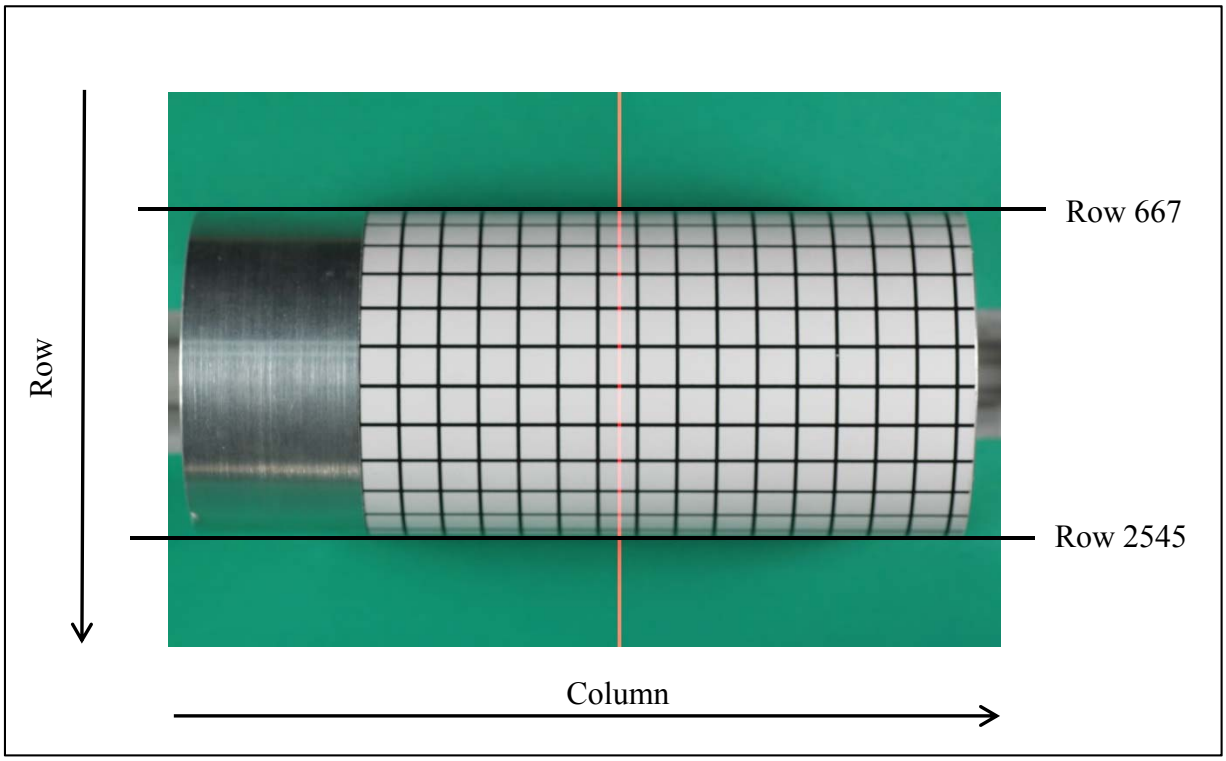


Figure 34: Vertical column of Interest, 2576 highlighted in orange

The RGB pixel values from the column are plotted on a graph where the Y-axis is the pixel value and the X-axis is the row number of the pixel, Figure 34. The RGB values can be combined to calculate a grayscale pixel value which can also be plotted with the RGB values. Recall that the grayscale value is calculated as: Grayscale Pixel Value = 0.2989xR + 0.5870xG + 0.1140xB.

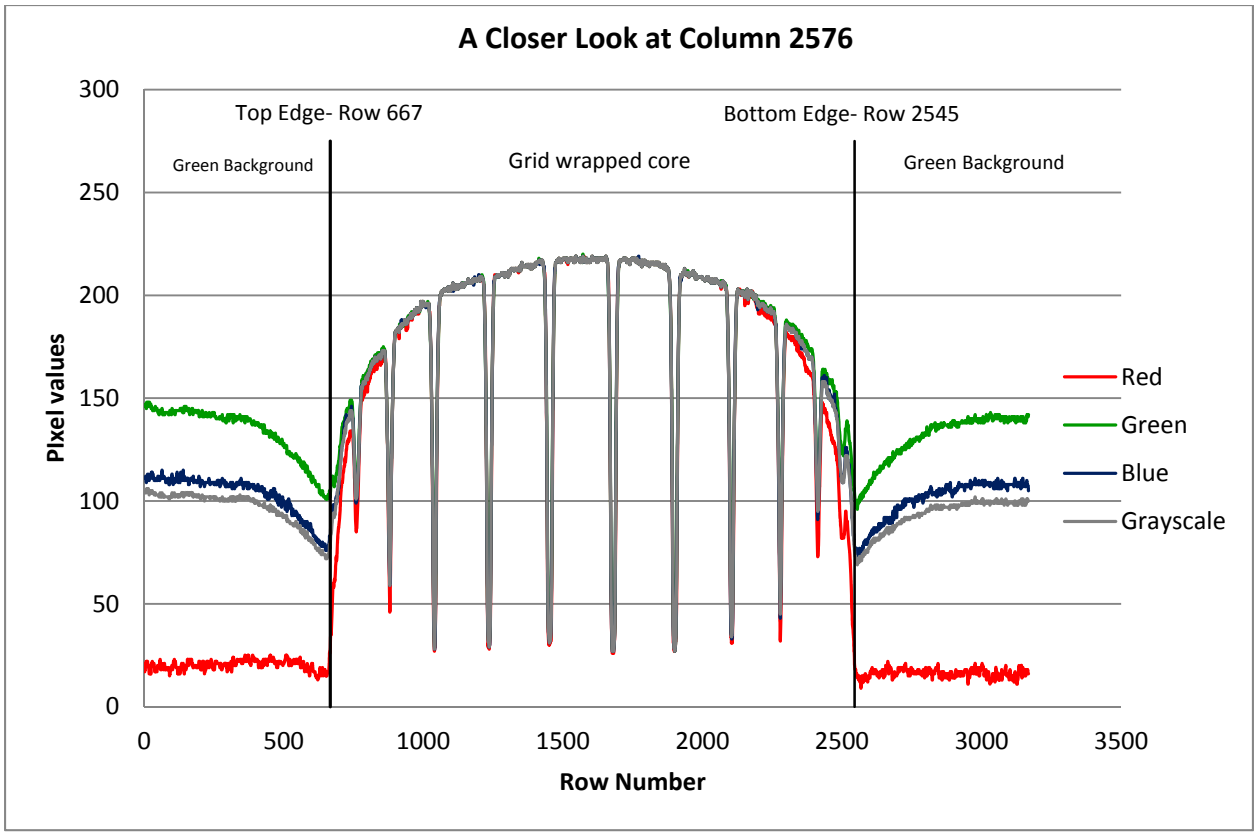


Figure 35: Graph of pixel values for column 2576.

At first glance at the graph in Figure 35 above, there appears to be a steep change in pixel values that may coincide with the edges of the core, identified by visual inspection. The data immediately around each of the "edges", row 667 and row 2545, are plotted separately on individual graphs below.

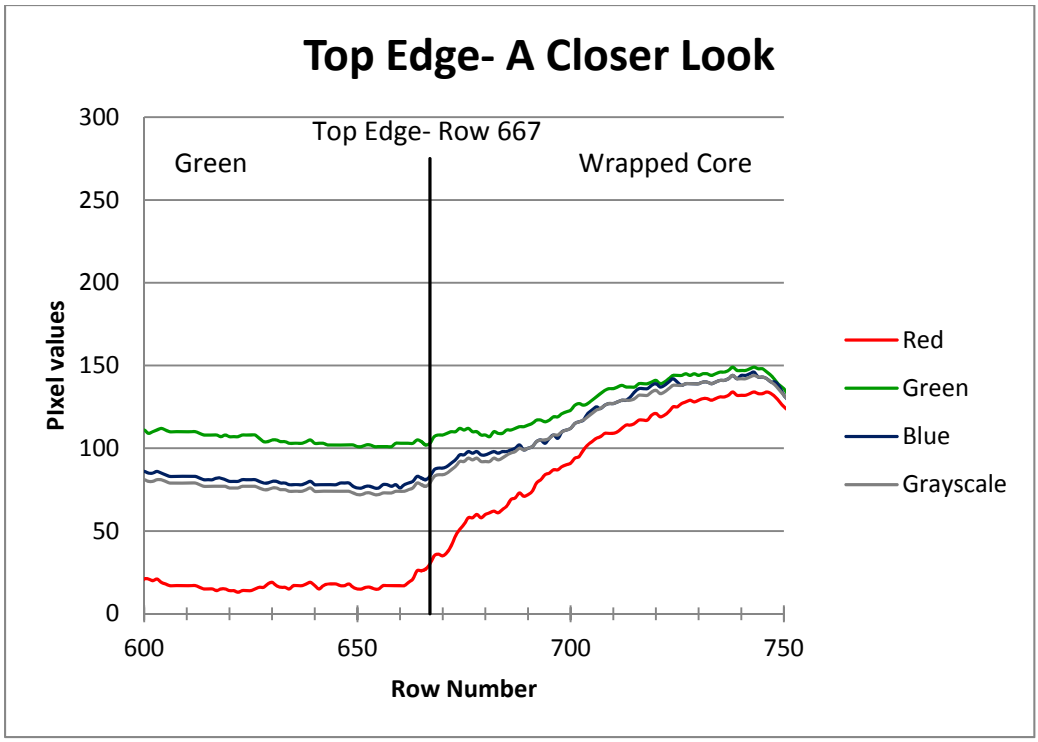


Figure 36: Pixel values around the top edge of the core in image

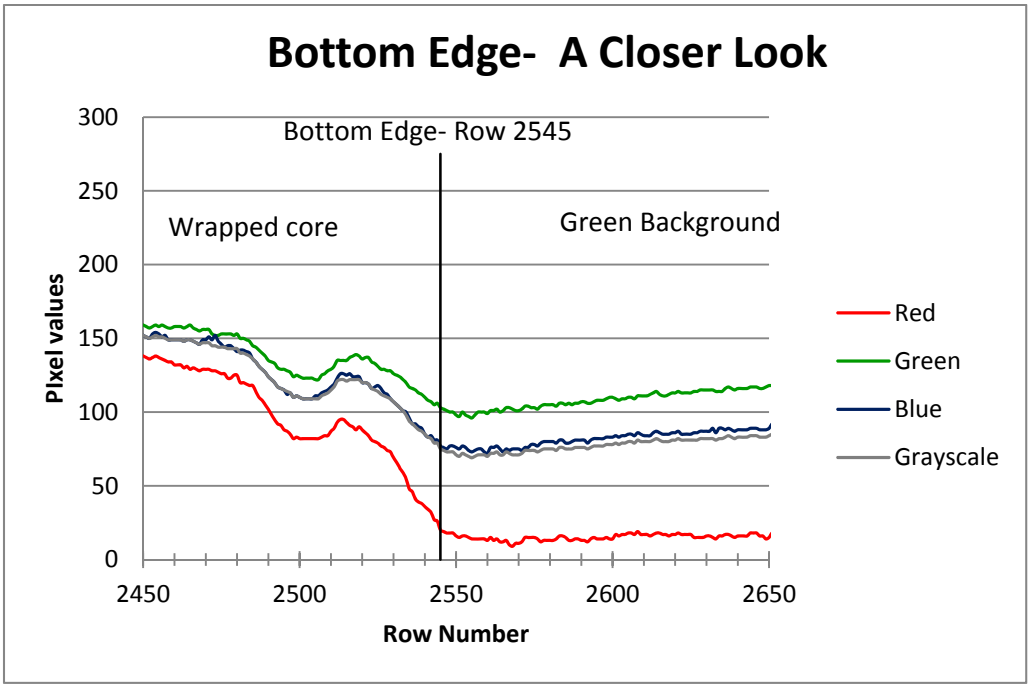


Figure 37: Pixel values around the bottom edge of the core in image

The following figures show that the RGB and grayscale values follow a similar trend and there may exist a local pattern of pixel values that can be used to identify the edges of the core. The pixel values around the edges tend to gradually decrease as they approach the core and then they experience a relatively sharp increase as they reach the core. In the next section this trend will be used to identify the edges of the core using a calculated gradient of the pixel values. This gradient edge detection will be automated and coded to find the edges of the core.

Gradient edge detection

The step value from one pixel to the next in the image is not constant. While one pixel may have a value of 115 an adjacent pixel, in any direction, is not necessarily 116 or 114. There may be an abrupt change in pixel values or the next adjacent pixel may be exactly the same. The gradient edge detection method looks for a unique increase or decrease in pixel values around the core edge to identify the edges of the core.

The gradient is calculated as $Y_{j+1} - Y_j$. Where Y_j is a pixel value from column j and Y_{j+1} is the next pixel value of the column j. The entire calculated gradient of column 2567 is plotted below in Figure 38. The edge locations determined from the visual inspection are plotted and labeled on the figure for reference. The gradient values before and after the edge markers show relatively constant positive and negative values with only occasional spikes in gradient values. Some of these spikes can be seen around row 500 and row 3000. Obviously, the large spikes are the gridlines present in the image.

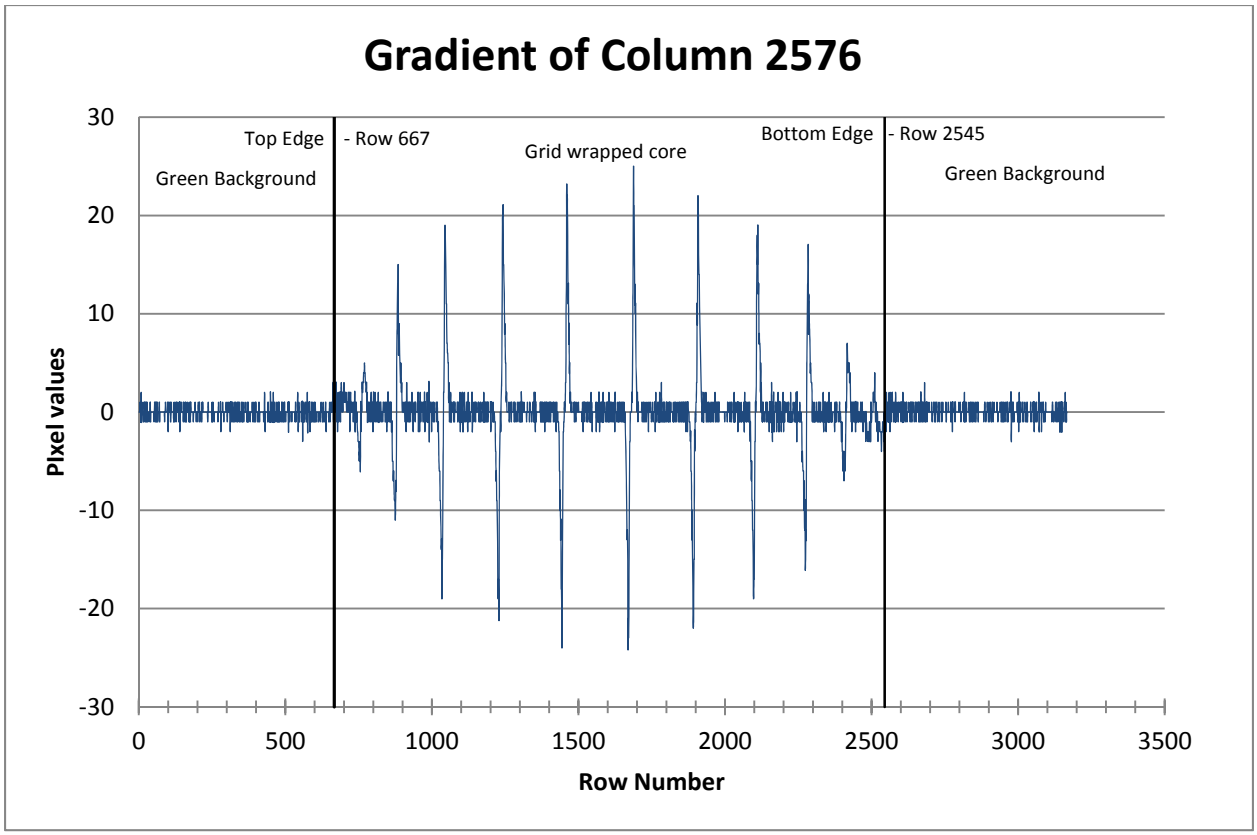


Figure 38: Graph of gradient of column 2567

If the first and last 500 gradient values are sampled, separately, there exists a local minimum and maximum spike in the gradient values for each data set. These ranges of gradients represent areas of the image that include the green background and not the core in the image.

If the gradient values are scanned starting from the left, X = 1, there exists two more spikes in the gradient values that will be the first values to exceed the local minimum and maximum values determined by sampling the first 500 gradient values. The same procedure can be repeated by starting from the right side of the gradient, X=3167, and scanning the gradient values to the left. This procedure will also produce two gradient values that exceed the maximum and minimum values determined by sampling the last 500 gradient values.

The gradient of the pixel values of column 2567 are calculated and the data are scanned for local max/min values. The data are scanned from each end a second time for values that exceed these local min/max values. The results are shown below.

Table VI: Gradient Edge Detection for Column 2567

Gradient Edge Detection	Top Edge	Bottom Edge
Local Min/Max	-2,2	-3,3
(First Value, Row #) > max	3, 663	4, 2511
(First Value, Row #) < min	-3, 559	-4, 2544
Visual Edge Detection Row #	667	2545

The gradient, by its nature, is the difference between two pixel values and does not have a whole integer row coordinate location in the image. The value may be plotted at a point half way between pixels but for convenience, the gradient value is assumed to occur at Y_j, where Y_j is the first value in the gradient calculation, $Y_{j+1} - Y_j$.

Remember that the visual inspection and location of the core edges had an estimated variance of ± 15 rows. The top and bottom edge min/max row numbers from Table VI are within ± 15 rows of the edges determined by visual inspection.

The following two figures show a closer look at the gradient values around the top and bottom edges of the core.

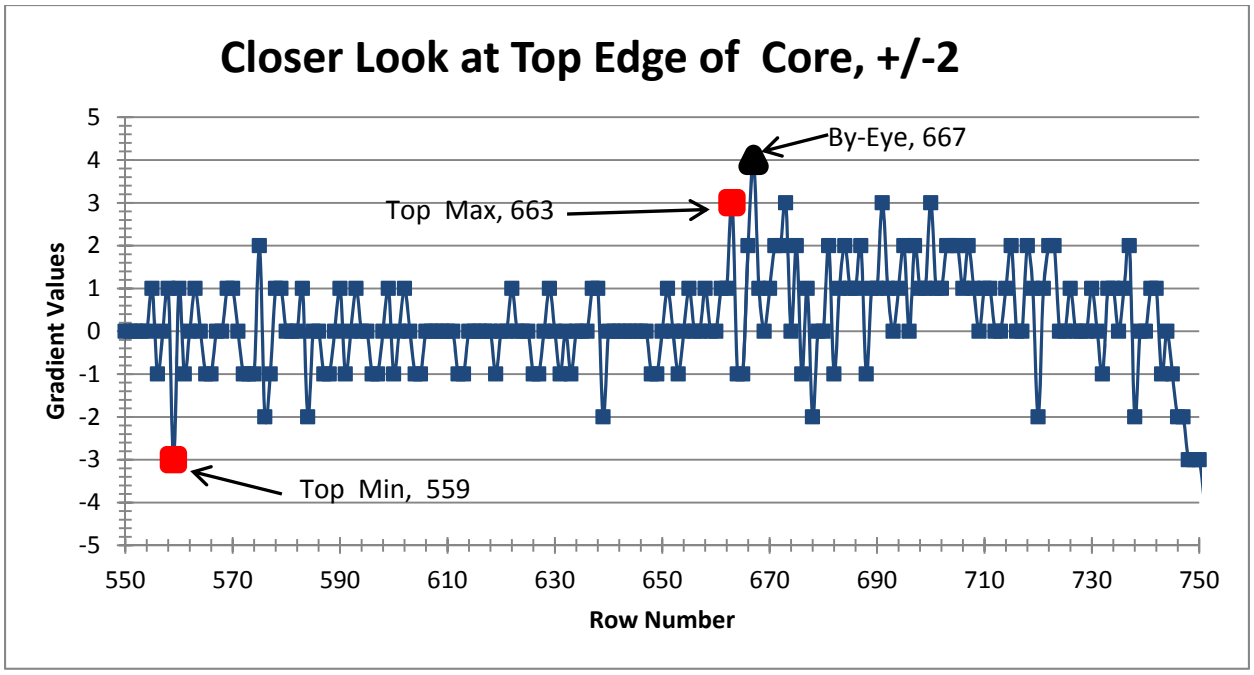


Figure 39: A closer look at gradient values around the top edge of core in the image

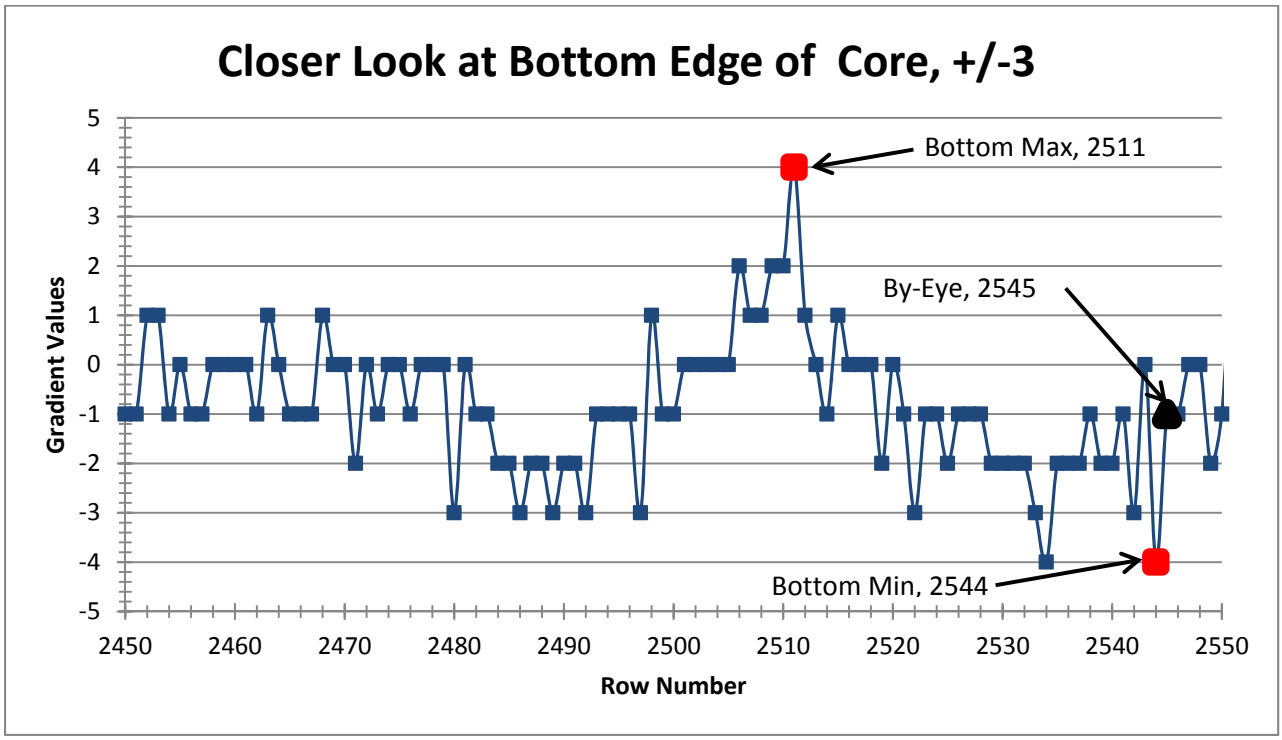


Figure 40: A closer look at gradient values around the bottom edge of core in the image

Inspection of Figure 38 and Figure 39 above, show that the Top max. and Bottom min. value are closest to the visually determined edges of the core. These are the results of an analysis of a single column from the image. Before a decision is made to use the location of these two values as the top and bottom edges of the core a survey of multiple columns is introduced.

The analysis is carried out for 1000 columns between columns 1500 and 2500. These limits are chosen to assure that columns selected for this analysis will intersect the core and not peripheral objects in the image, like the core roller. Each of the 1000 iterations of the analysis returns the row location of the top and bottom min/max occurrences. These values are categorized and recorded as Top Min, Top Max, Bottom Min, and Bottom Max. There are now 1000 values for each of the categories. The mode value for each of these categories is calculated along with the standard deviation of each category. The results of this analysis are shown in the table below.

Table VII: Analysis of Multiple Columns

Category	Mode Row #	Visual Inspection Row #	Standard Deviation	(Range Row #), Delta	Average Row #
Top Max (+)	666	667	60.7	(201-1056), 855	679
Top Min (-)	746		64.0	(502-1026), 502	721
Bottom Max (+)	2508		62.4	(1800-2666), 866	2502
Bottom Min (-)	2542	2545	74.4	(1600-2666), 1066	2529

Within the statistical summary of data in Table VII, the range of values returned by the analysis of multiple columns is large which shows that this gradient detection cannot be applied for just a single column of data.

The first conclusion was to use the Top Max and Bottom max values as the location of the top and bottom edges of the core. The mode values returned for these categories had the lower of the two standard deviations min/max. The lower standard deviation indicated that the variation of the values in these categories is smaller so they must be a "better" trend in the change of the gradient. It is obvious that while the Top Max mode row number agrees closely with value determined by visual inspection, the Bottom Max mode row number of 2508 is more than ±15 pixels from the value of 2545, determined by visual inspection. This indicates that the positive gradient value found in the lower portion of the image, near the bottom edge of the core, is not an accurate indicator of the edge of the core in the image, even though the category produced a lower standard deviation of data than the Bottom Min.

The comparison of standard deviation between the min/max categories for each region of the core may not be a suitable tool to compare the min/max.

The second conclusion starts with the fact that the gradient is directional. The sign of the gradient value is dependent on the direction which the gradient was calculated. Looking again at a plot of the pixel values, it is expected that the gradient will change signs from negative to positive as they transition around the top of the core while the opposite is true for the bottom of the core, (-) to (+). The top edge of the core is defined by positive gradients while the bottom edge is defined by negative gradient values. The Top Max (+), and Bottom Min. (-) categories will be used as an indicator of the edges of the core. The mode value from multiple column analyzes will be used for the coordinate to define the core edges.

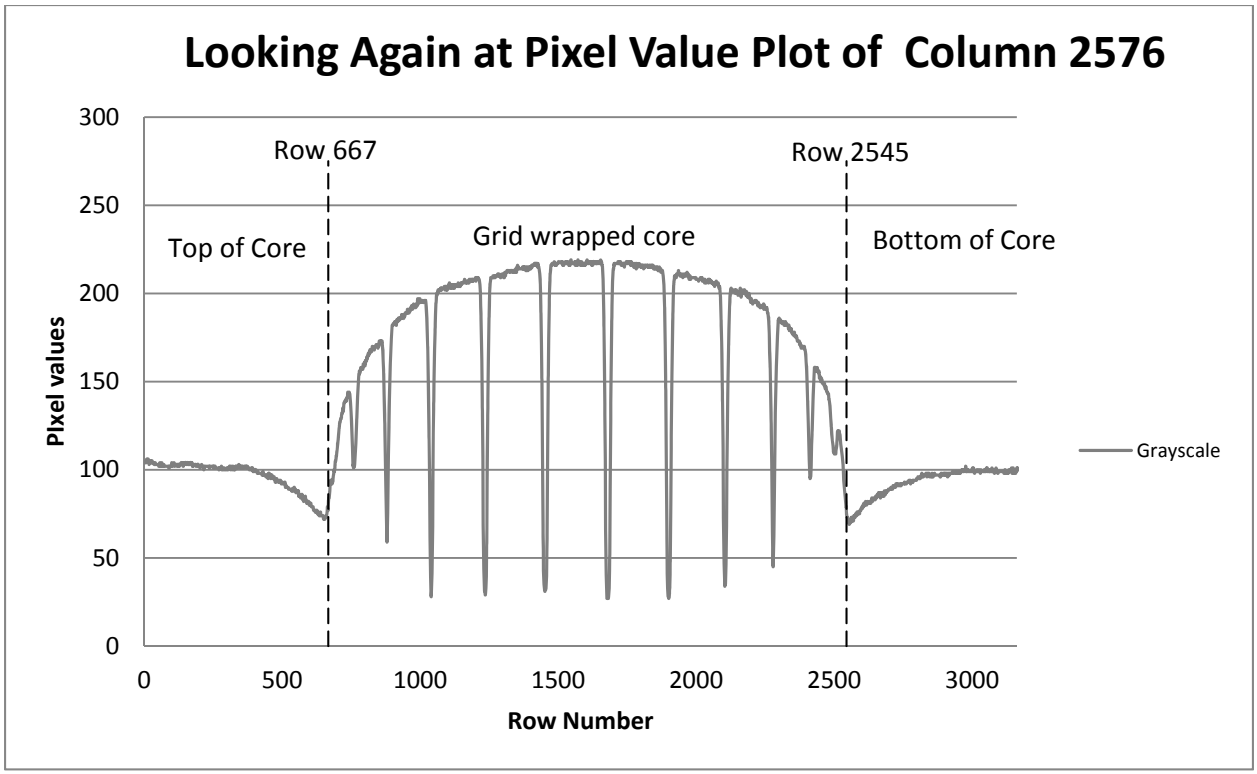


Figure 41: Grayscale values of column 2567 graphed as before

Conclusion

Table VIII shows the final edge coordinates determined by taking the Top Max row and Bottom Min. row coordinate from the gradient analysis. The values from the visual analysis have been provided for reference. The goal of edge detection is ultimately to find the center row of the cylindrical core in the image. The center row has been calculated as: Top Row + (Bottom Row- Top Row)/2. The calculated center row for each method is within two rows. There seems to be a good agreement between the two methods. An analysis of the final unwrapped image will be necessary to understand the effects of moving the center row by two pixels.

Table VIII: Gradient Edge Detection

Edge	Gradient	Visual Inspection
Top- Row	666	667
Bottom-Row	2542	2545
Center Row	1604	1606

The gradient method is outlined in the following steps.

1. Calculate the gradient of pixel values for a single column.
2. Scan the first and last 500 rows for local min./max
3. Scan the gradient data from the first row on until a gradient value exceeds the local minimum. This is the Top edge for a single column.
4. Reverse the gradient scan starting at the last row until a gradient value exceeds local maximum. This the bottom edge for a single column.
5. Repeat the entire process for 1000 columns between 1500 and 2500.
6. Take the mode value for all Top edge and Bottom edge rows returned by the multiple scans.
7. The mode Top row and Bottom row are the edges of the core.

The one fallback of automating this method is the initial scanning ranges for local min/max. The sample interval was set at 500 rows since that is a portion of the aluminum billet image that definitely contains only the green background. This is true for this image but as the camera height over the core changes this may no longer be true. If the camera height over the core is decreased enough so that only the core is present in the image and the green background is no longer visible, a sample interval of the first and last 500 rows may sample portions of the core when scanning for local min/max. At this point it is unknown how this would affect edge detection and there is no proposed solution to automatically define the initial sample interval.

Canny filtering automatic edge detection

The previous two sections showed that the edge of the core can be identified visually or automatically by sampling the gradient values of multiple columns of the image. Another

approach is to filter the image using canny edge detection. A color image is converted to a gray scale image and the canny edge detection is used to create a filtered black and white image. (Gonzalez et al, 2009)

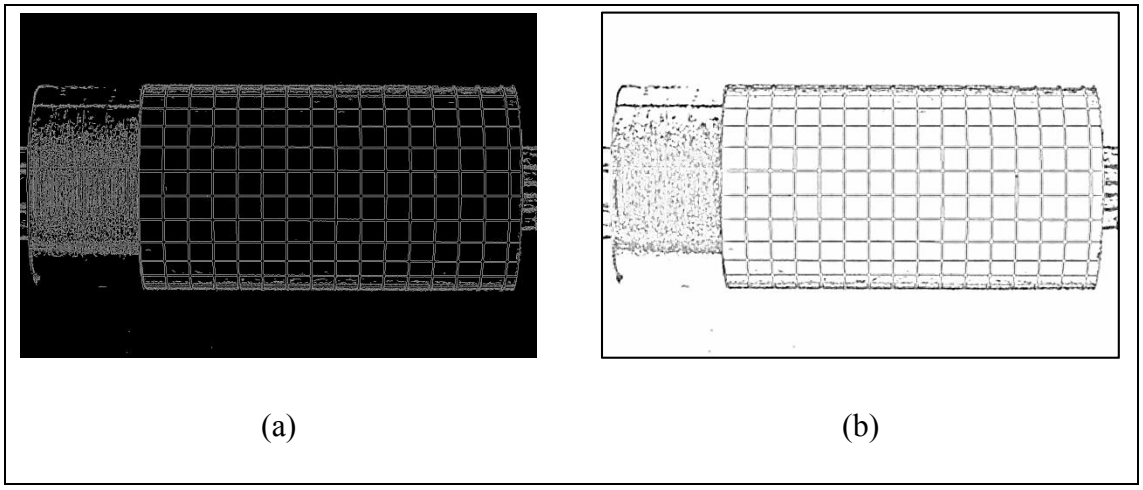


Figure 42: (a) Canny Filtered, (b) Complement Image

The binary image (a) is returned after applying the filter. The pixel values of the image are either 0 or 1. The complemented image (b) is provided for easier viewing. At first glance the lines in the images seem to be continuous but a closer look at the top edge of the core reveals that the lines are fragmented and not necessarily continuous, Figure 43.

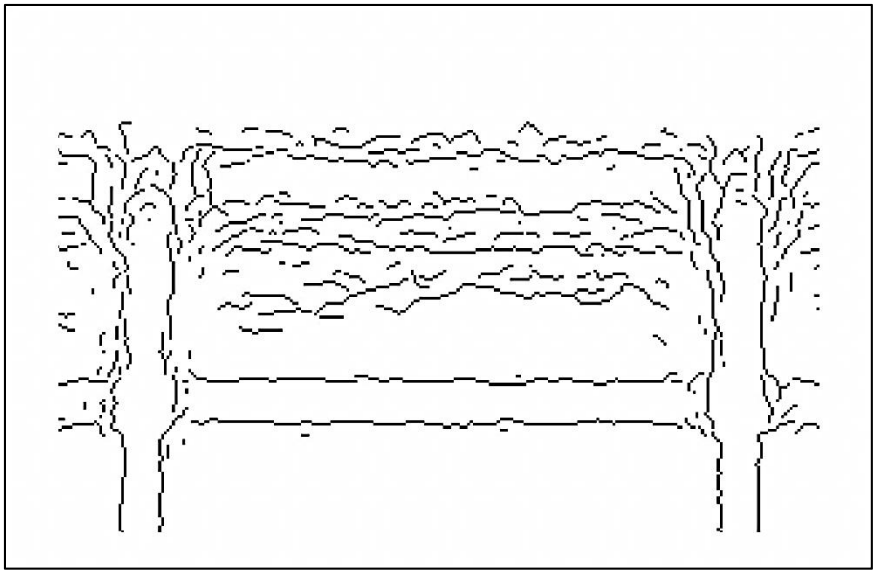


Figure 43: Resulting Canny Filter of core around the upper edge of the core in image

A single column of pixel values is extracted from the binary image Figure 38 (a). The binary column of data is a string of ones and zeros. A pixel value of one represents an edge produced by the filtering. The pixel values are plotted, below, on a graph where the Y-axis is the pixel value and the X-axis is the row number of the pixel.

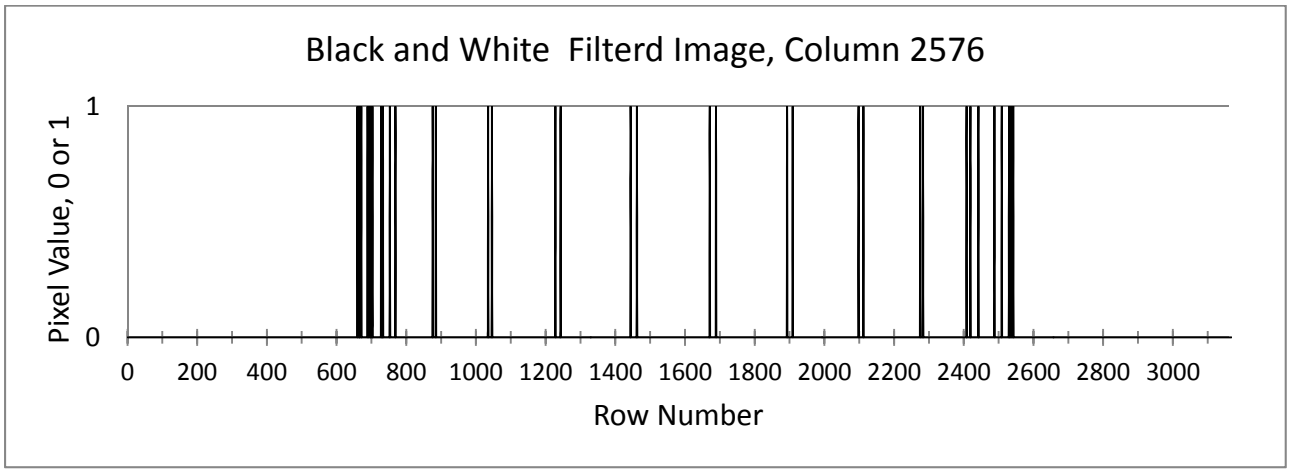


Figure 44: Graph of binary pixel values for a single column after Canny Edge Detection

The vertical black lines in the figure are pixels within the column with a value of 1. These pixels represent an edge. Looking at the figure it is apparent that there is a first and last pixel

with a value of 1. These two locations correspond to the top and bottom edges of the core for a single column. These values occur at row 683 and 2542 respectively. Remember that the edges are fragmented and these coordinate pairs are unique for this particular column. In order to get a better estimate of the location of the edges, all of the columns of the binary image are scanned and the top and bottom edges are recorded. These locations have been plotted on the gray scale image below, Figure 45.

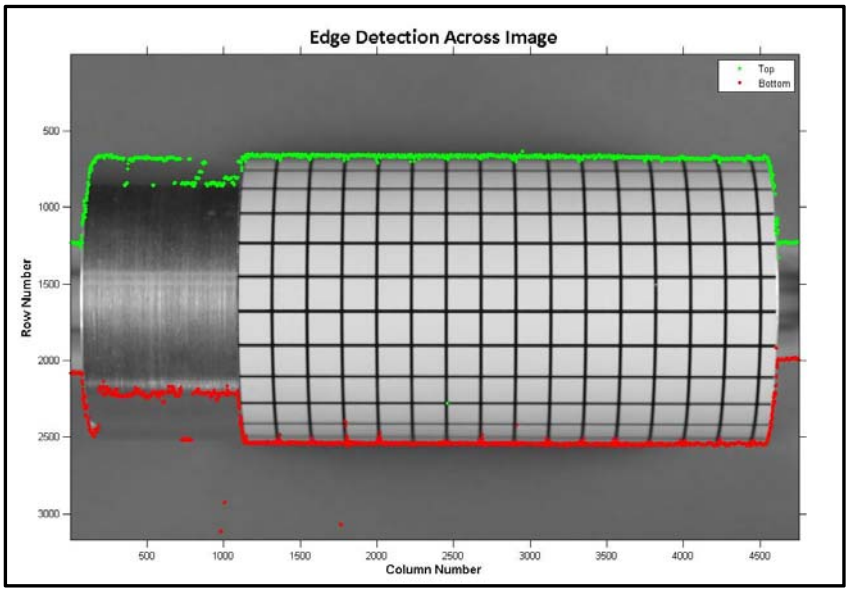


Figure 45: Red and green dots represent apparent edge of core as determined by Canny filter method

It is obvious that the edge detection suffers around the left side of the image where the aluminum surface is present in the image. The portion of the image wrapped with the core shows better results but the values still have a range and outliers are present. Scanning each of the 4,572 columns of the image produced two data sets. The first data set contains 4,572 row numbers identified as the top edge and a second data set that also contains 4,572 row numbers for the bottom edge. The following table shows the statistical summary of the each of these data sets. The edges determined by visual inspection have been provided for reference.

Table IX: Statistical Summary

Edge	Visual Inspection	Mode	Median	Mean	Range
Top- Row	667	683	670	714.24	1643
Bottom-Row	2545	2542	2540	2454.8	1199

Recall that the edge is probably within ±15 rows of the value determined by visual inspection. Looking at the values in the statistical summary of each data set it would appear that the median value of the top row data set is closest to the value determined by visual inspection. For the bottom row data set the mode value of the edge values would be closest to the visually determined value. The first thought is to use these two values determined by different statistical means as the top and bottom edge coordinates because they match a value that was determined by user judgment. This seems like an arbitrary approach that may not be very robust.

The two data sets contained all of the edge rows from every column of the image. In Figure 45, it is apparent that the aluminum surface produced some undesirable results and those values from that region are likely skewing the statistical summary. The next idea is to reduce the range of columns scanned in order to find a range that may not be skewed by the outliers seen in the aluminum surface region of the image.

Table X: Concentric Scanning

Center - 2400	Top Edge -667			Bottom Edge- 2545		
Column Scan Range	Mean	Median	Mode	Mean	Median	Mode
2300-2500	673	664	662	2542	2542	2542
2200-2600	669	664	666	2542	2542	2542
2100-2700	667	664	662	2542	2542	2542
2000-2800	666	664	661	2541	2542	2543
1900-2900	666	664	661	2541	2542	2543
1800-3000	666	664	661	2541	2542	2543
1700-3100	666	664	661	2543	2542	2542
1600-3200	665	664	661	2542	2542	2542
1500-3300	665	664	661	2542	2542	2542
1400-3400	665	664	661	2542	2542	2542
1300-3500	665	664	662	2542	2542	2542
1200-3600	665	664	661	2542	2542	2542
1100-3700	667	664	661	2538	2542	2541
Average	**667**	**664**	**662**	**2542**	**2542**	**2542**
1000-3800	673	665	661	2529	2542	2541
900-3900	679	666	661	2520	2542	2542
800-4000	682	666	661	2513	2542	2542
700-4100	685	667	661	2510	2542	2544
600-4200	685	667	661	2502	2541	2543
500-4300	685	667	661	2496	2541	2543
400-4400	685	668	683	2490	2541	2542
300-4500	685	669	683	2485	2541	2542
200-4600	685	669	683	2480	2541	2542
100-4700	698	669	683	2469	2541	2542
Entire Image	714	670	683	2454	2540	2542

A series of concentric scans is performed starting around the center of the image at column 2400. Each iteration increases the scanning range by 200 columns. For example, the first scan searches columns 2300 -2500, the next search scans columns 2200 -2600, etc. Each scan returns two data sets and the mean, median and mode have been calculated for each set. The results are shown in the table below. In the original image of the gridded core the aluminum surface has a column range of 1 to 1100 while the grid pattern is present in columns 1100 to 4600.

There are several interesting trends in the data of Table . The effects of edge detection in the area of the aluminum surface tend to skew the mean, median and mode values for both the top and bottom edge data sets. This is seen as the range grows in size and exceeds the range 1000-3800.

Another interesting trend is that the median row value does not change for either the top or bottom edges, as the scanning range remains within the portion of the image that contains only the grid pattern. This trend will be capitalized on and used to determine the final edge rows for Canny edge detection.

Since the median value does not change over a wide scan range, a scan range of ±1000 rows from the center of the image is chosen for the final version of this method. Since the core is always centered in the image, this range should only intercept the core and not peripheral objects like the core rollers present in some of the images.

The final results of the Canny edge detection returned median edge row coordinates for a scan range of ± 1000 columns around the middle of the image. The following table compares the results of the Canny edge detection with the other two previous methods discussed.

Table XI: Canny Edge Detection

Edge	Canny	Visual Inspection	Gradient
Top- Row	664	667	666
Bottom-Row	2542	2545	2542
Center Row	1603	1606	1604

The top and bottom edges returned by the canny filtering have been superimposed over the original image and are highlighted in red in Figure 46.

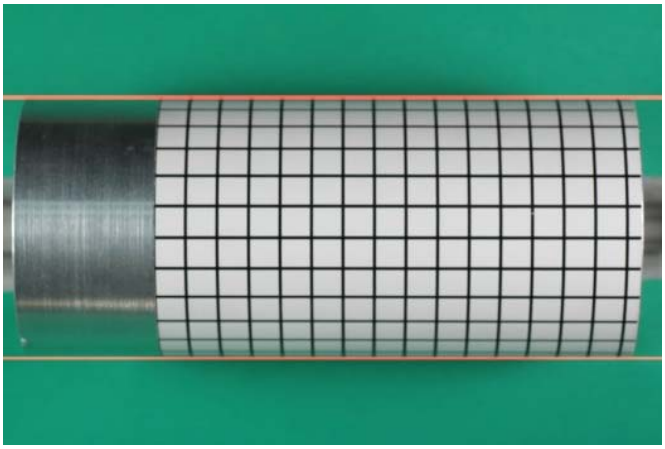


Figure 46: Edges of core, red lines, as determined by Canny Edge Detection

It is remarkable to see that the edge rows returned by all three methods are in such close proximity. The center row values are within +/- 3 rows. The effects of center row location will be determined later in an analysis of the unwrapping.

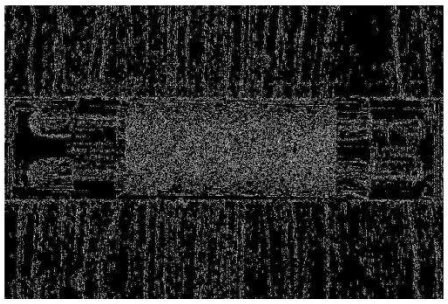
The final procedure for determining the edge coordinates by Canny filtering is outlined below:

1. Convert RGB image to grayscale
2. Apply canny edge filtering to produce a binary image.
3. Scan a single column and record the row location for the first and last white pixel. This will be the top and bottom edges for this column.
4. Repeat the scan for multiple columns at a range of ±1000 columns from the center of the image. The result is two data sets of row locations. These are the top and bottom row data sets.
5. Calculate the median value for each data set. These two values will be the top and bottom edge row coordinates.

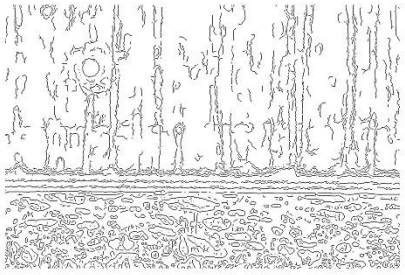
While the Canny filtering appears to be a robust edge detection method, there are instances where it will produce unrealistic edge coordinates. This is obvious when the background in the

image is not uniform or smooth. The following figures show what happens when the background is not uniform.

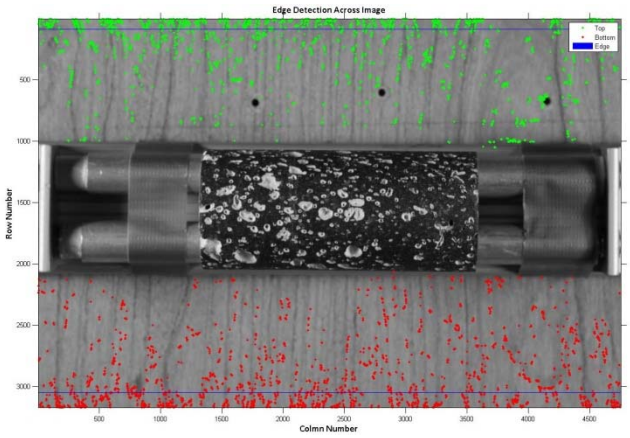
(a)- Original Image



(b)- Binary Filtered Image



(c)- Compliment image, Close up of Top Edge of figure (b)



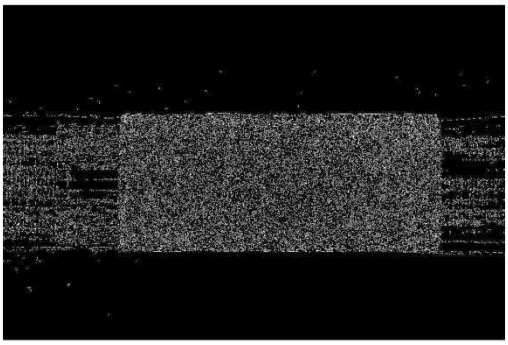
(d)- Median Edge Coordinates

Figure 47: Results of edge detection with a non-uniform background

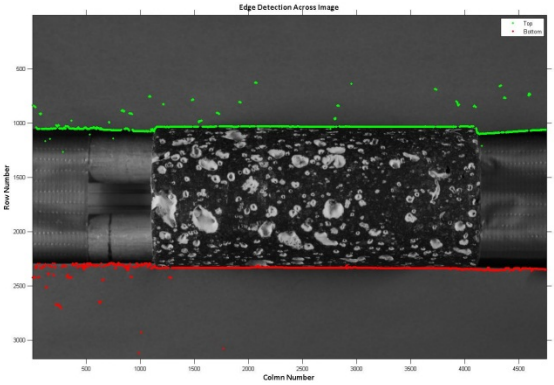
In Figure 47 (c) it is obvious that the non-uniform background contains many fragmented points and lines that have been identified as edges by the canny filter. The scatter of the incorrect edges is apparent in figure (d). The median row coordinates of the edges are highlighted by blue lines just at the top and bottom edge of the image and they are obviously nowhere near the edges of the core in the image.

Now let us look at a typical image of a rock core with a uniform background and apply the canny filter.

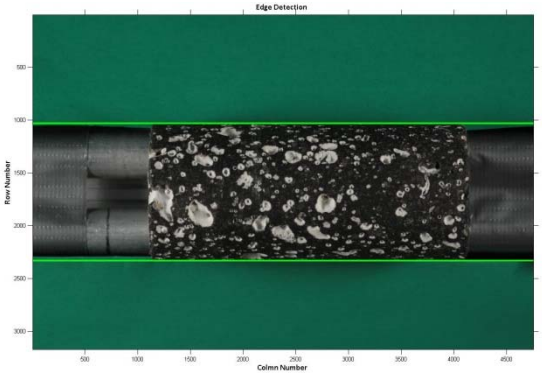
(a)- Original Image



(b)- Binary Image



(c)- Median Edge Coordinates



(d)- Final Edges

Figure 48: Edge detection of a typical rock core with a uniform background

The median edge coordinates have a small range with few outliers, Figure 48(c). The median edge is outlined in blue in figure (d). This image has a uniform background and the filtering appears to find suitable edges that could be used to find the center of the image in unwrapping operations.

Edge Detection Methods Conclusion

The lack of an absolute stark contrast between the edges of the core and the background makes edge detection an approximation. A comparison of the edge coordinates determined by the three edge detection methods is presented in Table XII.

Table XII: Comparison of Edge Detection Methods

Edge detection Method	Top Edge- Row	Bottom Edge- Row	Center Row
Visual Inspection	667	2545	1606
Gradient	666	2542	1604
Canny Filtering	664	2542	1603

All three methods appear to be in close agreement. Ultimately the purpose of the edge detection is to find the centerline of the core in the image. The centerline row coordinates are very close, +/- 3 rows, for all three methods. In order to evaluate these three methods they will each be applied to an image during the unwrapping process. The resulting unwrapped image will be analyzed and a single measure of the accuracy produced as a result of using each method will be compared to further weigh the edge detection methods.

Unwrapping

In digital image processing a spatial, or geometric, transformation takes a pixel's location in the current image, input space, and translates it to a new image, output space, based on a geometric relationship. The reader may be familiar with resizing an image, or even a computer window. This is a form of geometric transformation where pixels in the input space are mapped to output space based on a scaling factor like 2X.

In order to find the value of a pixel in output space, inverse mapping is used as opposed to forward mapping (Wolberg, 1990). In the Matlab environment, a forward mapping function is used to find the bounds or borders of the final image. This is called the output space. Once the

shape and size of the output space is defined, inverse mapping is used to “look back” to the original image, input space, and interpolates the final value of the output space pixel. Nearest neighbor bi-linear interpolation will be used to find the unwrapped pixels values. Bi-linear and bi-cubic interpolation are also available to the user (Eddins,2014).

For the unwrapping of the cylinder, the unwrapping equation is the forward function used to find the bounds of the final image. The new bounds of the unwrapped image will only increase the number of rows in the output image; no new columns will be added to the unwrapped image. The inverse of the unwrapping equation is used to evaluate the value of each pixel in the new output space. The unwrapping transform does not really move pixels, it is the inverse equation that interpolates new pixels based on pixel values closest to the inverse point in the original wrapped image, or input space.

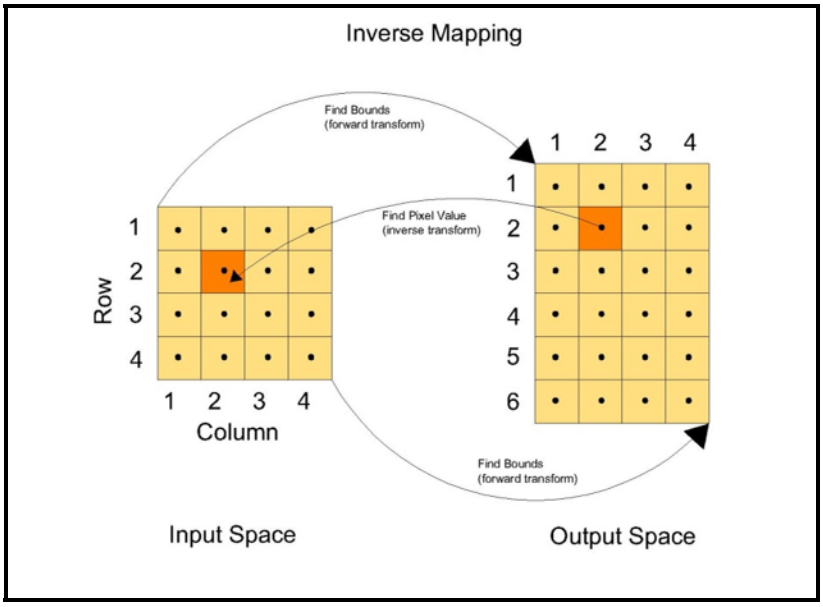


Figure 49: Illustration of inverse mapping

Custom geometric transform- unwrapping

Two custom transforms, forward and inverse, have been derived to perform the unwrapping of the cylindrical core. The transforms are based on a pixel's calculated arc length away from the center of a cylinder. The derivation of the transform is shown below. The diagram is of a cross-section of a cylinder, which is a circle.

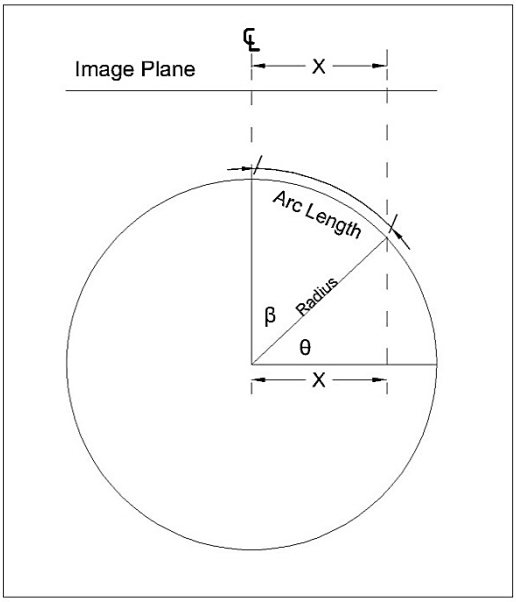


Figure 50: Diagram outlining formula variables for unwrapping

$$\beta = \frac{\pi}{2} - \theta; where\ \theta = acos(X/Radius)$$

$$Arc\ Length(X) = Radius \times \beta$$

$$Arc\ Length(X) = Radius \times \left(\frac{\pi}{2} - acos(X/Radius)\right) \quad \text{(Forward Function)}$$

$$X(Arc\ Length) = Radius \times cos\left(\frac{\pi}{2} - Arc\ Length\ /\ Radius\right) \quad \text{(Inverse Function)}$$

The equations above are in radians so the radius, measured in inches, is converted to # of rows using the pixel scale. The distance X to a point in the image plane is measured from the top

centerline of the circle. In the image of the cylinder the distance X is a pixels position from the horizontal centerline of the core.

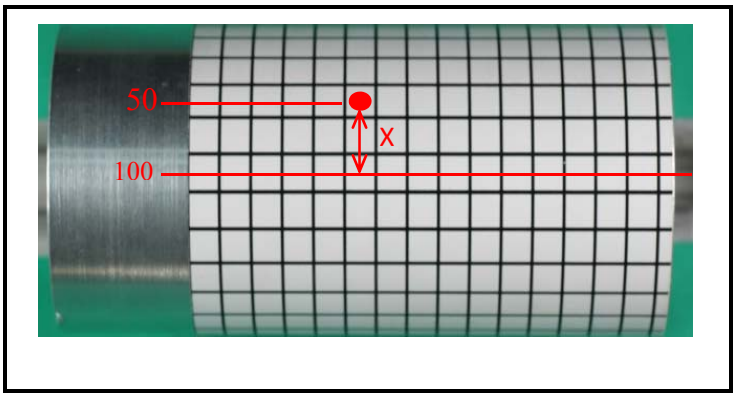


Figure 51: Distance X of a pixel away from the center of the core in the image

In Figure 51, a point at distance X is arbitrarily assigned a row value of 50. The center of core is assigned a value of 100. The distance X is calculated by subtracting the row number of the point by row location of the point. For the hypothetical case, above, the calculation for distance X is 100-50 = 50. This is a simple calculation that gives a pixel's distance in rows from the center of the core. The arc length or forward function can be easily modified to include this calculation.

$$Arc\ Length(X) = Radius\ \times \left(\frac{\pi}{2} - \text{acos}(X/Radius)\right)$$

$$Arc\ Length(X) = Radius \times \left(\frac{\pi}{2} - acos((Center\ Row - Pixel\ Row)\ /\ Radius)\right)$$

That would be the end of it, except that finding the inverse equation is not so straightforward. The arc length is a function of a given pixels distance from the center row of the unwrapped image, or output space.

$$X(Arc\ Length) = Radius \times cos\left(\frac{\pi}{2} - (Arc\ Length/Raidus)\right)$$

$$X(Arc\ Length) = Radius\ \times cos\left(\frac{\pi}{2} - \ (Center\ Row - Pixel\ Row)/Raidus\right)$$

In order to find the arc length, the center row of the unwrapped image would have to be known in advance of using the inverse transform to calculate a pixel value in the output space. Rather than going through the effort to find the center of the output image, matrix translation is used to reassign a pixel row number to reflect its location relative to the center of the image regardless of whether the pixel is in the input or output image matrices.

In an original image matrix, the rows adjacent to the center row have a distance of X = 1, yet the row number might be some number like 2 or 3, as seen in Figure 52. In order to reposition the pixels closest to the center row so that their row number equals the distance from the center row, the original matrix is split into two halves along the center of the 4x4 matrix by copying the pixel values from each half to a new matrix. This 4x4 matrix represents a portion of an image between the edges of the core. This particular matrix has an even number of rows. The case for odd numbered images will be presented much later.

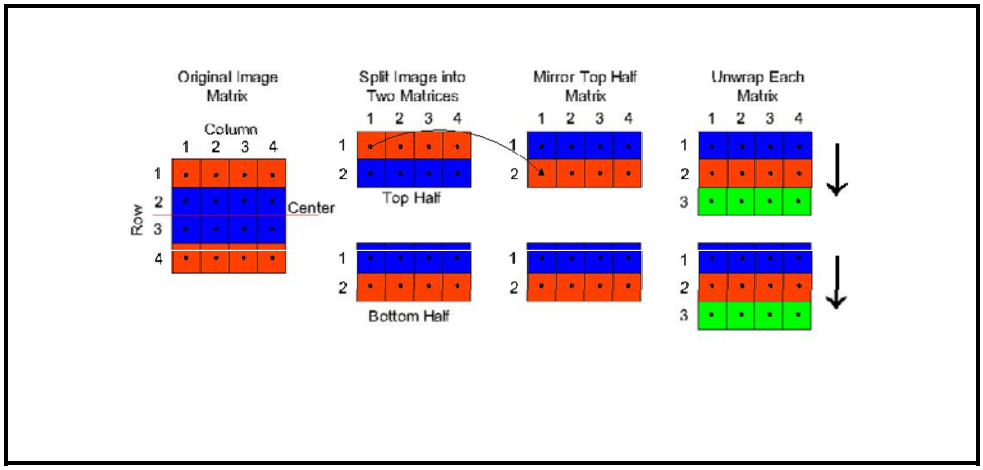


Figure 52: Image transformation methodology

The row numbers of the bottom half of the image now reflect the pixel distance from the center of the image. The row numbers of the top half of the image need to be manipulated in order to re-position the pixels so that their row number reflects their position relative to the center. To do this, the top half of the image is mirrored or inverted so that pixels that were once nearest to the center of the original image are now at the top of the new top half matrix and have a row number of 1 which is equal to that rows distance from center.

Now that the pixel rows reflect their position relative to the center of the image, the forward and inverse equations can again be modified to reflect the new pixel row coordinates.

The forward function remains almost unchanged except the pixel row number is substituted for the calculated distance of the pixel from the center of the image.

$$Arc\ Length(X) = Radius\ \times \left(\frac{\pi}{2} - acos((Center\ Row - Pixel\ Row)/Radius)\right)$$

$$Arc\ Length(X) = Radius\ \times \left(\frac{\pi}{2} - acos(Pixel\ Row\ Number/Radius)\right)$$

The inverse function can also be rewritten using the exact same substitution since a pixels row in the output space will also be its distance away from the center of the unwrapped image. This saves having to calculate the center row of the unwrapped image before applying the transform formulas.

$$X(Arc\ Length) = Radius \times cos\left(\frac{\pi}{2} - \ ((Center\ Row\ Unwrapped - Pixel\ Row)/Radius)\right)$$

$$X(Arc\ Length) = Radius \times cos\left(\frac{\pi}{2} - \ ((Pixel\ Row\ Number)/Radius)\right)$$

The development of the completed unwrapped matrix is shown in the steps below. Notice that the top half of the image is mirrored twice, first, to re-position pixels prior to unwrapping and again after unwrapping. The two unwrapped matrices are combined to form the final unwrapped image.

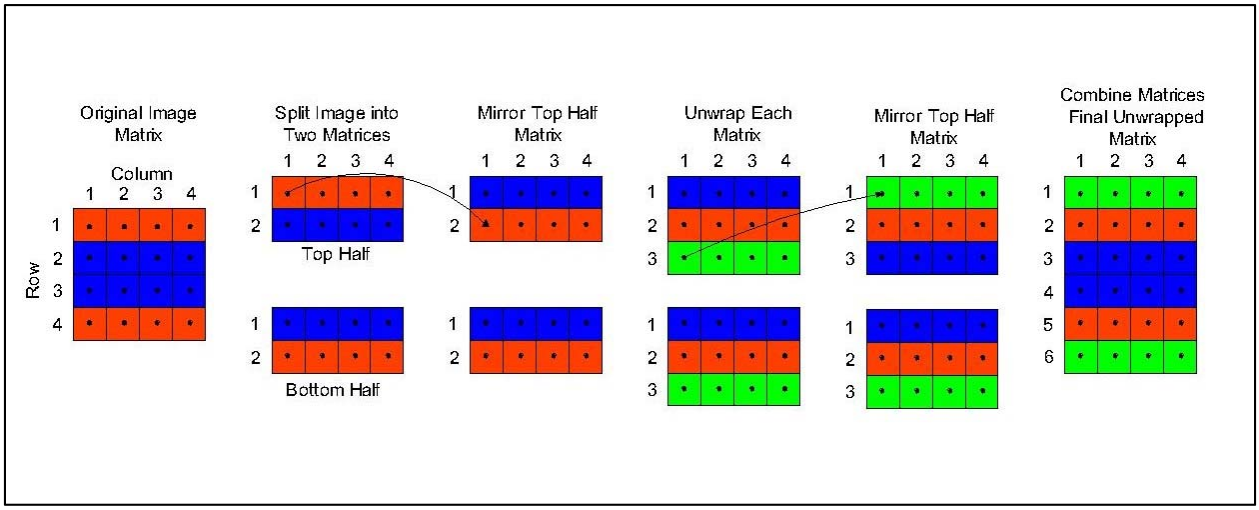


Figure 53: Unwrapping of an image by pieces

Inverse mapping

In Figure 53, the point in the process that illustrates the unwrapping of each matrix was simplified. The following discussion will explain in more detail how the final form of the transformation equations are used in inverse mapping to unwrap the image.

In inverse mapping the forward function defines the bounds or borders of the new image. For the cylinder the forward, or unwrapping, function has been modified so that it is individually applied to each half of the image. It is important to reiterate that the forward function only finds

the bounds of the new image and not the pixel values. The inverse equation will be used to find the actual pixel values.

Forward Function - Arc length(X) = Radius x (π/2 – acos(Pixel Row Number)/Radius)

Let us look at a depiction of one half of a typical image with the following parameters.

- Scale = 0.00110132 inches/pix
- Radius = 1.062 inches
- Radius = 1.062 in. / 0.00110132 in/pix = 964.29 rows (or pixels)
- Dimensions: 940 rows , 4752 columns
    - 940 rows visible in one half of image indicates that only 175.5 deg. of the whole core was visible in the image.
    - Row 940 is the furthest row from the center of the core.

The original bounding box of the wrapped image has four cells at the corners of the image with coordinates shown below in Figure 54.

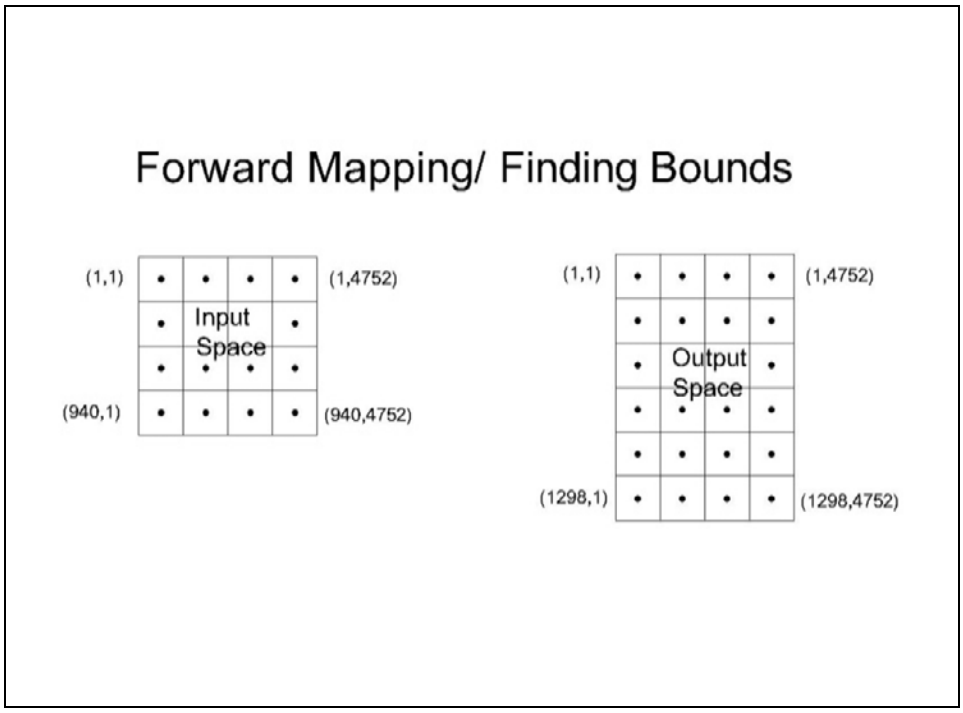


Figure 54: Forward function illustration

Only the new row values of the output space bounds are calculated using the forward function, since the columns will remain the same for unwrapping the image.

$$\text{Arc length}(1,1) = 964.29 \times (\pi/2 - \text{acos}(1/964.29) = 1.0000002;$$

Round to nearest whole row number (1.0000002,1) = (1,1)

$$\text{Arc length}(940,4752) = 964.29 \times (\pi/2 - \text{acos}(940/964.29) = 1297.78;$$

Round to nearest whole row number (1297.78,4752) = (1298,4752)

The final bounding row coordinates are rounded to the nearest whole integer. The output space bounds define a new empty matrix with 1298 rows and 4752 columns.

The inverse mapping function is used to populate the cells of the output space matrix with color values**.**

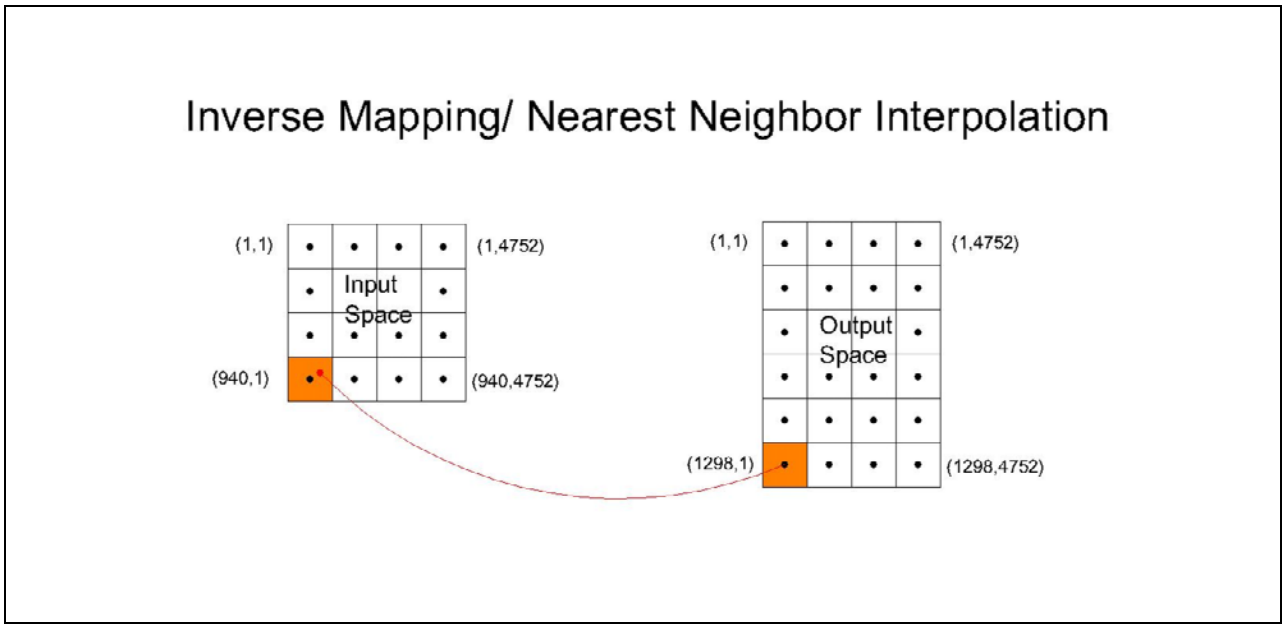


Figure 55: Inverse mapping illustration

Using the inverse equation and the cell row number in the output space, a location in the input space can be defined from which the output space cell color value is determined.

$$X(\text{Arc Length}) = \text{Radius} \times \cos(\pi/2 - (\text{Pixel Row Number})/\text{Radius}))$$

$$X(1298,1) = 964.29 \times \cos(\pi/2 - (1298/964.29) = 940.047.$$

Remember that the column number does not change, so the location of the output space cell (1298,1) in input space is (940.047,1). Nearest neighbor, or point sampling, interpolation returns the closest pixel value to the inverse mapped location. For this case, a pixel value at row (940,1) would be closest to sampling point (940.047, 1).

The modified versions of the unwrapping equations are:

$$\text{Forward Equation } (X) = \text{Radius} \times (\pi/2 - \text{acos}(X)/\text{Radius}) \quad \text{(eqn. 2)}$$

Where X is the row number of a pixel in input space.

$$\text{Inverse Equation } (X) = \text{Radius} \times \cos(\pi/2 - (X/\text{Radius})) \quad \text{(eqn. 3)}$$

Where X is the row number of an empty cell in output space.

Even and odd number of rows in an image

The previous discussion on matrix re-positioning prior to unwrapping presented a case where the image had an even number of rows. The input image can have either an even or odd number of rows. The following discussion will explain how and why the number of rows needs to be accounted for when the image is split into two halves prior to and during the unwrapping transformation.

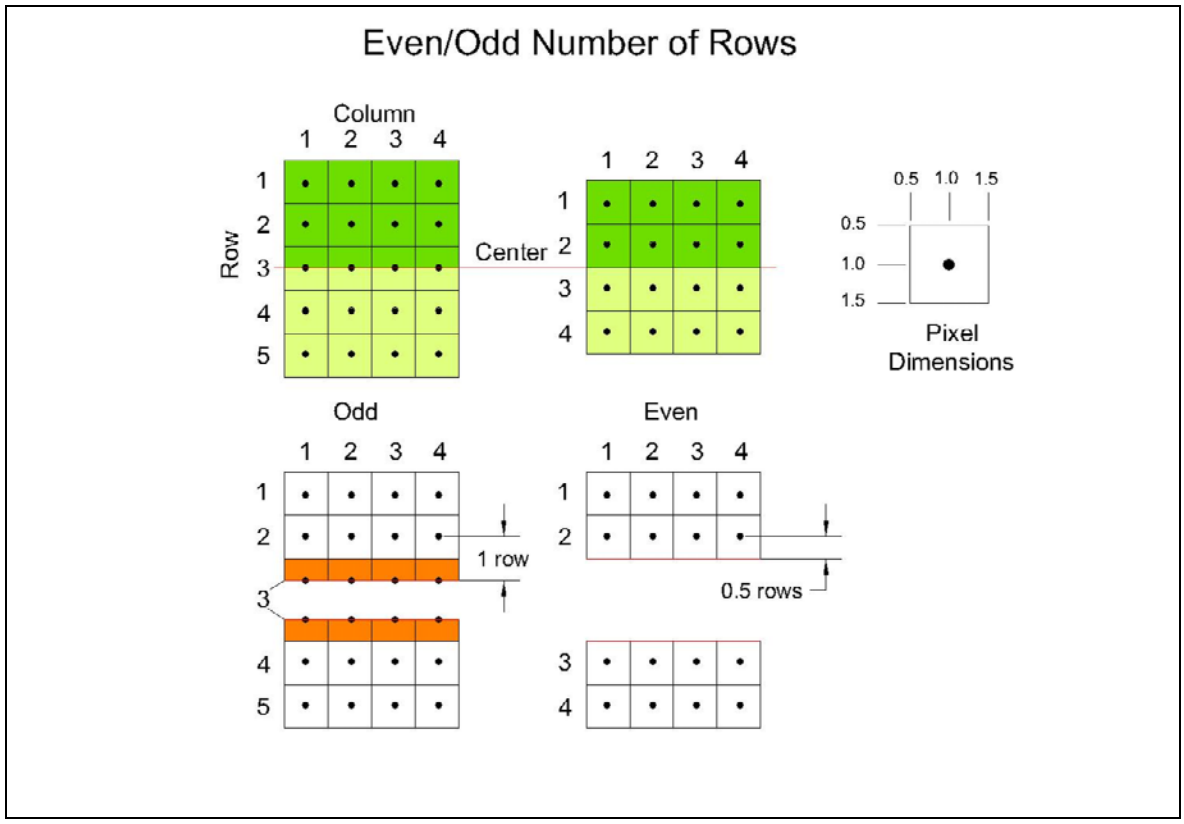


Figure 56: Illustration of even and odd # of rows

Splitting an image with an odd number of rows in two equal parts means splitting the image along the center of a center row, Figure 56. In this figure the distance of the 2nd row from the center is one row. The distance of the 1st row from the center is 2 rows. Obviously, a cell cannot be split in half so in order to compensate for this, the center row or the 3rd row is appended to each half of the image before the unwrapping sequence is applied. This ensures that

when the forward and inverse transforms are applied the row distances from center will be reflected in the row number of a pixel. Just before the two matrices are appended together to form the final unwrapped image, the original center row is removed from the top half matrix. The entire process for unwrapping images with an odd number of rows is shown below.

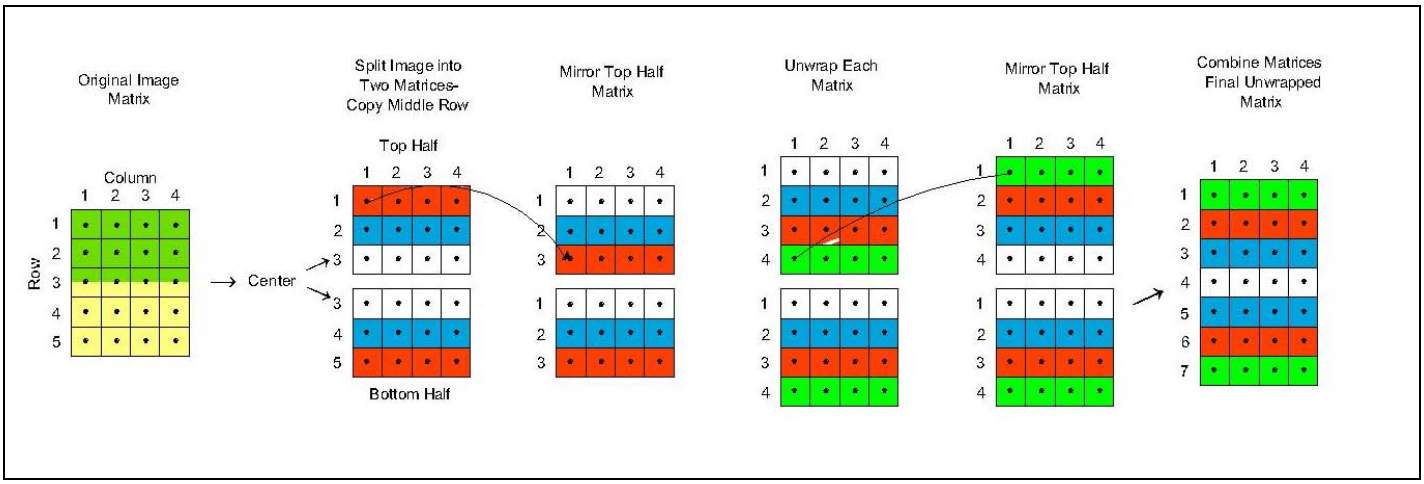


Figure 57: Piecewise unwrapping of image with odd number of rows

Splitting an image with an even number of rows creates some complications. The distance of the second row from the center of the image is 0.5 rows. The distance of the first row from the center is 1.5 rows. There is no way to geometrically transpose the cell locations so that their row number is any fraction of a whole number. In other words the row numbers must be whole numbers. In order to compensate for the half row distances the transform equations are modified as follows:

$$\text{Forward Equation (X)} = \text{Radius} \times (\pi/2 - \arccos((X-0.5)/\text{Radius}))$$

Where X is the row number of a pixel in input space minus 0.5 rows.

$$\text{Inverse Equation (X)} = \text{Radius} \times \cos(\pi/2 - ((X+0.5)/\text{Radius}))$$

Where X is the row number of a pixel in input space plus 0.5 rows.

In the previous discussions, it was stated that the forward function is used only to find the bounds of the output space but the forward equation can also be used to map individual input space pixels to the output space. The forward function produces row numbers that are in fractions of a row and must be rounded to the closest whole number in order to be mapped in the output space. Conversely the inverse function, used to interpolate the pixel value of a cell in output space, will always begin a calculation with a whole row number from the output space, which produces a fractional row number in input space. The inverse function "looks" for the closest whole row number in input space in order to sample the pixel value for the output space, using nearest neighbor interpolation. The best way to show that this fractional modification to the row location works correctly is to follow the inverse mapping, mathematically. The following calculations will use the parameters from one half of an image with an even number of rows. These are the same parameters used in the earlier discussion of inverse mapping.

- Scale = 0.00110132 inches/pix
- Radius = 1.062 inches
- Radius = 1.062 in. / 0.00110132 in/pix = 964.29 rows (or pixels)
- Dimensions: 940 rows , 4752 columns
- Row 940 is the furthest row from the center of the core.

Using the forward function, we can map, not only the bounds of the output space, but also any row of the image input space to the output space. Since there are an even number of rows in the image halve, a value of 0.5 rows will be subtracted from the current row value in input space to compensate for the location of the center.

A demonstration calculation is shown using the row furthest away from the center, 940, before presenting the results of multiple calculations in table form.

Forward Equation (X) = Radius x (π/2 – acos ((X-0.5)/Radius)

Forward Equation (X) = 964.29 x (π/2 – acos ((940-0.5)/964.29) = 1295.56.

The row coordinate from input space must be mapped to a whole row coordinate in output space. The row number 1296.55 is rounded to become row 1296. The inverse function is expected to map row 1296 back from the output space to row 940.0, and not row 940.5. Since the inverse mapping uses nearest neighbor interpolation it is only necessary that the inverse mapping produces a coordinate that is not greater than 940 +/- 0.50.

Inverse Equation (X) = Radius x cos(π/2 –(X+0.5)/Radius)

Inverse Equation (X) = 964.29 x cos(π/2 – 1296+0.5)/964.29) = 939.71

Since 939.71 is greater than 939.5, the nearest neighbor interpolation will select a pixel value from row 940 for use in the output space. This confirms that with careful accounting the transform functions will successfully handle partial row values. A selection of other row values is evaluated in the table below to confirm the modification of the mapping functions for images with an even number of rows.

Forward- Function					Inverse- Function				Check if < 0.5	
Original Row- Input Space	Subtract -0.5 rows	Output Space Location	Round		Output Space Row	Add +0.5 rows	Input Space Location		(Original Row - Input Space Location)	
940	939.5	1295.56	1296		1296	1296.5	939.71		0.39	ok
939	938.5	1291.16	1291		1291	1291.5	938.58		0.39	ok
938	937.5	1286.85	1287		1287	1287.5	937.65		0.38	ok
937	936.5	1282.62	1283		1283	1283.5	936.71		0.38	ok
936	935.5	1278.46	1278		1278	1278.5	935.51		0.38	ok
935	934.5	1274.37	1274		1274	1274.5	934.53		0.38	ok
934	933.5	1270.35	1270		1270	1270.5	933.54		0.37	ok
933	932.5	1266.39	1266		1266	1266.5	932.53		0.37	ok
932	931.5	1262.50	1262		1262	1262.5	931.50		0.37	ok
931	930.5	1258.66	1259		1259	1259.5	930.72		0.37	ok
930	929.5	1254.87	1255		1255	1255.5	929.67		0.37	ok
929	928.5	1251.14	1251		1251	1251.5	928.60		0.37	ok
928	927.5	1247.46	1247		1247	1247.5	927.51		0.36	ok
927	926.5	1243.83	1244		1244	1244.5	926.68		0.36	ok
926	925.5	1240.25	1240		1240	1240.5	925.57		0.36	ok
925	924.5	1236.71	1237		1237	1237.5	924.72		0.36	ok
924	923.5	1233.21	1233		1233	1233.5	923.58		0.36	ok
923	922.5	1229.76	1230		1230	1230.5	922.72		0.35	ok
922	921.5	1226.35	1226		1226	1226.5	921.55		0.35	ok
921	920.5	1222.97	1223		1223	1223.5	920.66		0.35	ok
920	919.5	1219.63	1220		1220	1220.5	919.76		0.35	ok

These calculations have been carried out for all 940 rows of the image half, and every single calculation returned a check less than 0.5, which indicates that the inverse mapping worked throughout the image.

The entire unwrapping process for an image with an even number of rows is shown in the figure below.

The depiction of the unwrapping in Figure 58, obviously does not depict the inverse mapping calculations that were described in detail above.

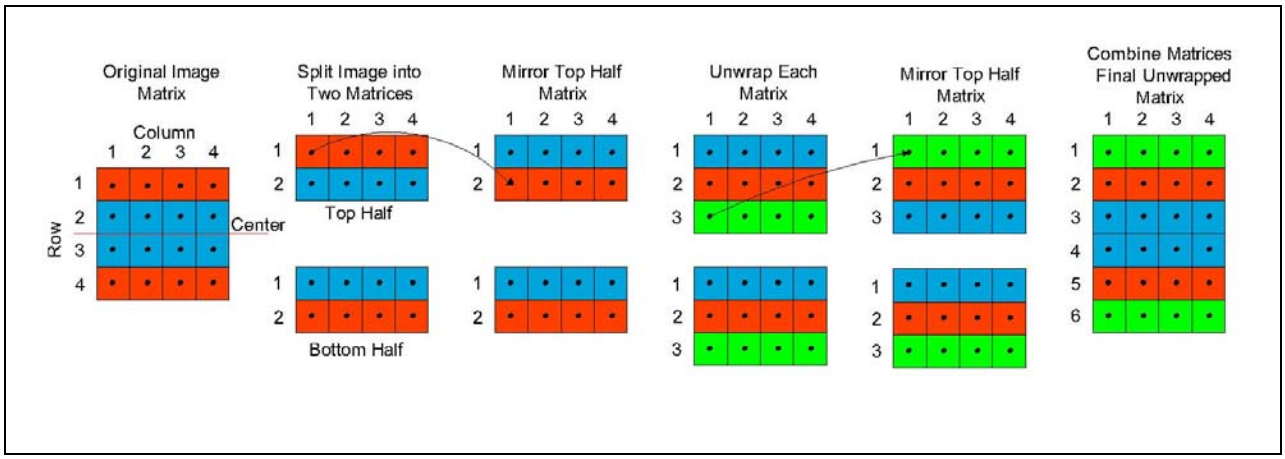


Figure 58: Piecewise unwrapping of image with an even number of rows

The final form of the transform equations are again modified to their final form, which is found in the Matlab code. The forward and inverse equations are as follows:

$$Forward\ Equation(X) = Radius \times \left(\frac{\pi}{2} - acos\left(\frac{(X-C)}{Radius}\right)\right);\ \begin{cases} C{=}0; Odd\ number\ of\ Rows \\ C{=}0.5; Even\ Number\ of\ Rows \end{cases}$$

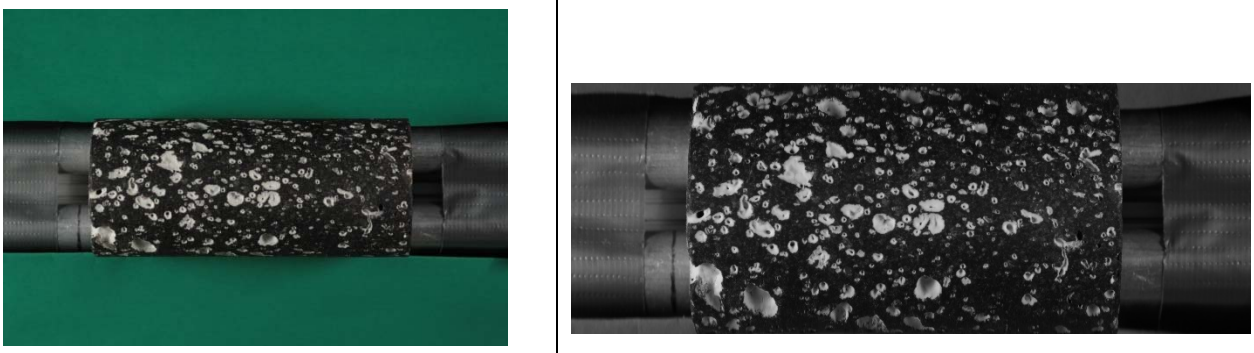
$$Inverse\ Equation(X) = Radius \times cos\left(\frac{\pi}{2} - \frac{(X+C)}{Radius}\right);\ \begin{cases} C = 0; Odd\ number\ of\ Rows \\ C = 0.5; Even\ Number\ of\ Rows \end{cases}$$

Application of unwrapping transforms

The order of operations for unwrapping is as follows:

1. Crop the original image at the edges of the core using one of the three edge detection methods described earlier. (Visual, Gradient, or Canny)
2. Determine if there are an odd or even number of rows in the cropped image.
3. Odd-Rows: Split the image into two matrices along the center row. Copy the center row to both halves.
4. Even- Rows: Split the image along the centerline of the image to create two new matrices.
5. Take the top half and mirror it over the top row. This will put pixels in the correct position for unwrapping.
6. Apply the inverse mapping to the top and bottom half of the image separately. C = 0, for odd number of rows, C = 0.5 for even number of rows.
7. Mirror unwrapped top half of the image to restore it to its original orientation.
8. Concatenate or append the top half and bottom half back together for a complete unwrapped core.

The steps outlined above are applied to an image of the core wrapped with a grid pattern first, in order to establish a pixel scale for a given camera set up and diameter core. Once the grid pattern is removed subsequent images of the core can be unwrapped by applying the same unwrapping procedure. Figure 59, below, shows the results of unwrapping an actual rock core.



(a)- Original Image | (b)- Unwrapped image

Figure 59: Unwrapped Rock Core

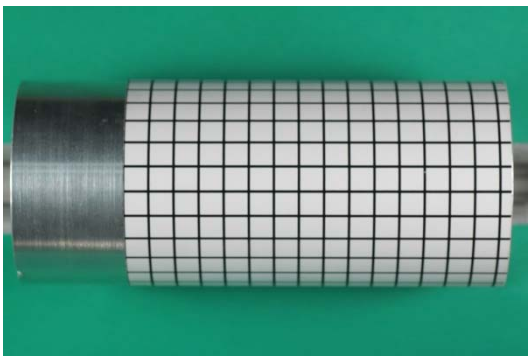
It is difficult to see the results of unwrapping transformation on the rock core in Figure 59 without careful and close scrutiny of each void present in the image and even then these are only subjective observations. This is why all of the measurable effects of the transformation need to conducted using the grid pattern wrapped around the original rock core.

ANALYSIS

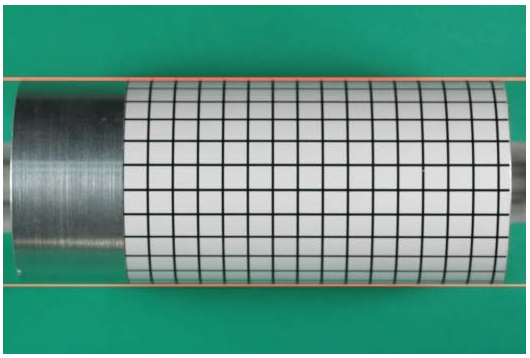
The simplest analysis is to look at the effects of unwrapping on an image that contains a gridded core wrapped around a cylindrical object. For the following analysis an image was selected that contained a grid printed on paper and wrapped around a cylindrical aluminum billet with the following parameters:

- Scale = 0.0011 inches/ pixel or row
- Radius = 1.062 inches
- Radius = 1.062 in / 0.0011032 in/row = 964.29 rows
- Unwrapping Gradient method

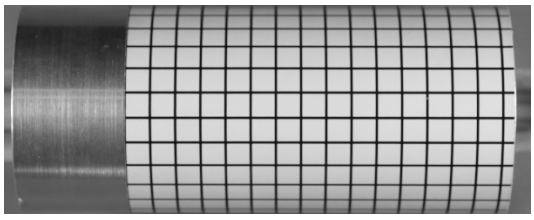
The images below show the original image at various stages in the unwrapping process. Visually there is an improvement to the distortion of the grid squares in the vertical direction from figure (a) to figure (d). The following procedure will outline a method to quantitatively measure improvements to the image through unwrapping.



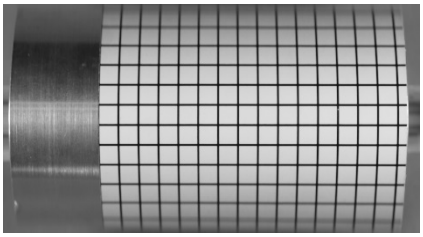
(a) Original Image



(b) Edge Detection- Canny



(c) Cropped Image



(d) Unwrapped Image

Figure 60: Images form each step of unwrapping workflow

Measuring error

In an earlier section it was shown that the location and spacing of horizontal gridlines can be measured using the cut-off method. The original printed-paper gridline pattern has gridline spacing of 0.25 inches. Ideally the gridlines in the final unwrapped image should all have a spacing of 0.25 inches. By determining the location of the gridlines, the spacing, or number of

rows between the lines and the scale of a row/pixel in the image, the spacing of the gridline can be calculated and compared to the original spacing of 0.25 inches. With this information, a percent error is calculated for the distance between each horizontal gridline. The percent error of gridline spacing is calculated as:

$$Measured\ Spacing(inches) = MeasuredSpacing\ (rows) \times Scale\ (in/pixel)$$

$$\%\ Error = ((Measured\ Spacing(in.) - Known\ Spacing(in.))/Known\ Spacing(in.)) \times 100$$

A negative % error indicates that the spacing between two gridlines is less than the known spacing. A positive value indicates that the spacing is greater than the known spacing.

Error Calculation Process

The percent error is calculated between horizontal gridlines at several locations across the image using several iterations of the cut-off method. For each iteration of the cut-off method, the user is prompted to visually select a column between two vertical gridlines in the image.

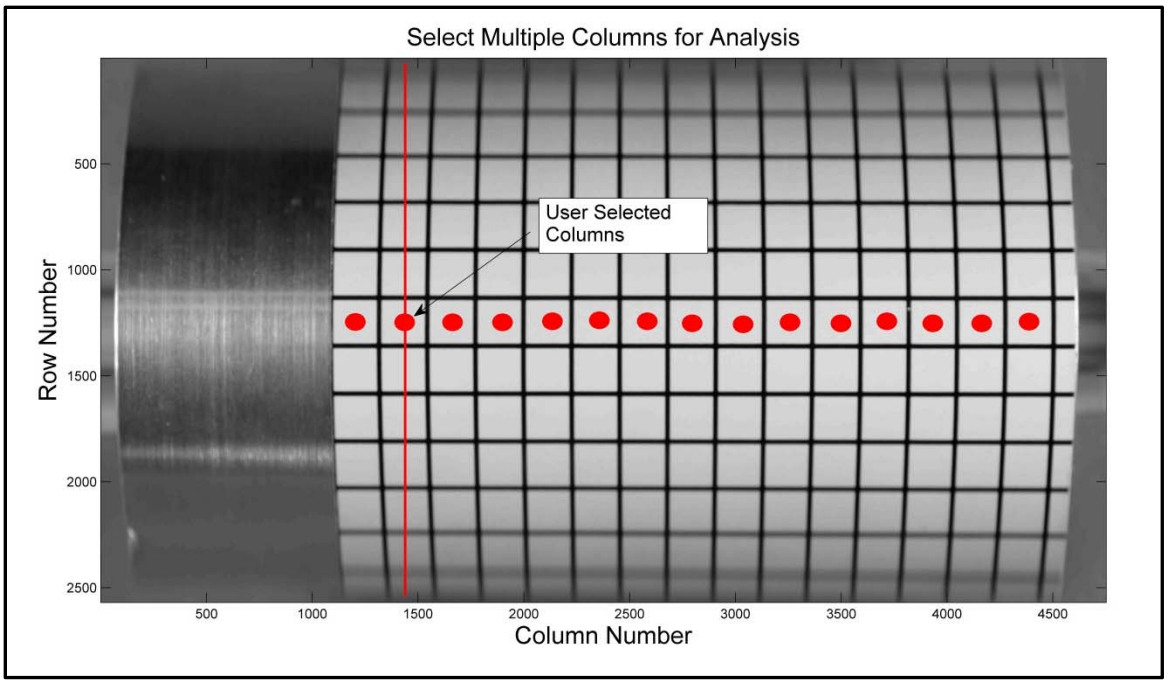


Figure 61: User Selected Column

The selected column is plotted as a bar chart and the user is prompted to select the cut-off value and search range by visually moving a horizontal line within the bar chart, Figure 62. The ends of the red line in the figure symbolize the extents of the search range. Once the cut-off value is selected, the location of the midpoint of each line is found and the spacing between adjacent gridlines is calculated. These spacing values are converted from units of pixels to inches using the scale, and the error in the gridline spacing is calculated. The process then returns the user to select another column for analysis and the process is re-iterated.

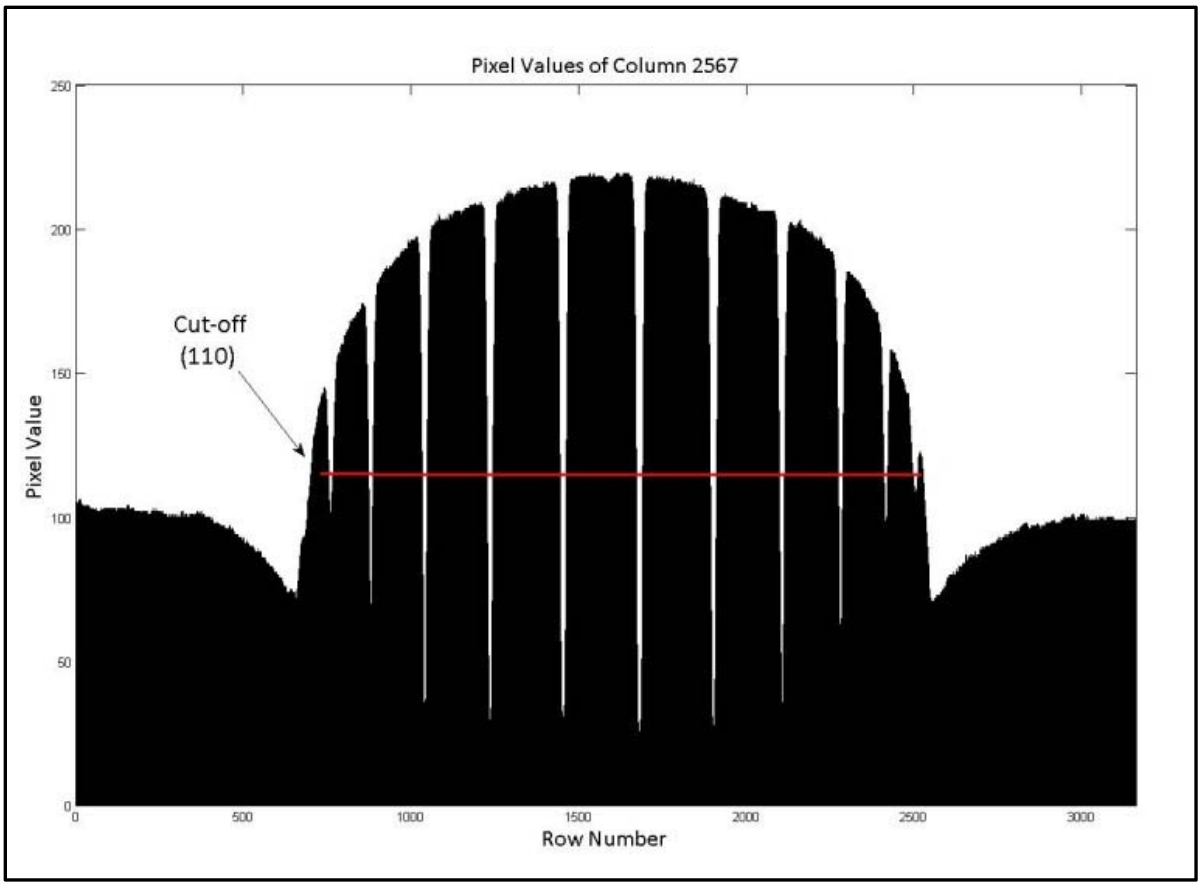


Figure 62: User selected cut-off and range for locating gridlines in an image

After the code has scanned each of the users selected columns the results of the error calculations are plotted on the image. Each value is plotted along the user-selected column at the midpoint between each line. The figure below is provided to the user as a quick visual check to assure that the calculations successfully found the spacing between each gridline at a selected column. The figure is automatically saved to an image file. Several key values from the iterative process are written to a data base and saved for later use. These values include the location of the each user selected column, cut-off value, search range, midpoint of each line at the selected column, midpoint between lines, and % error.

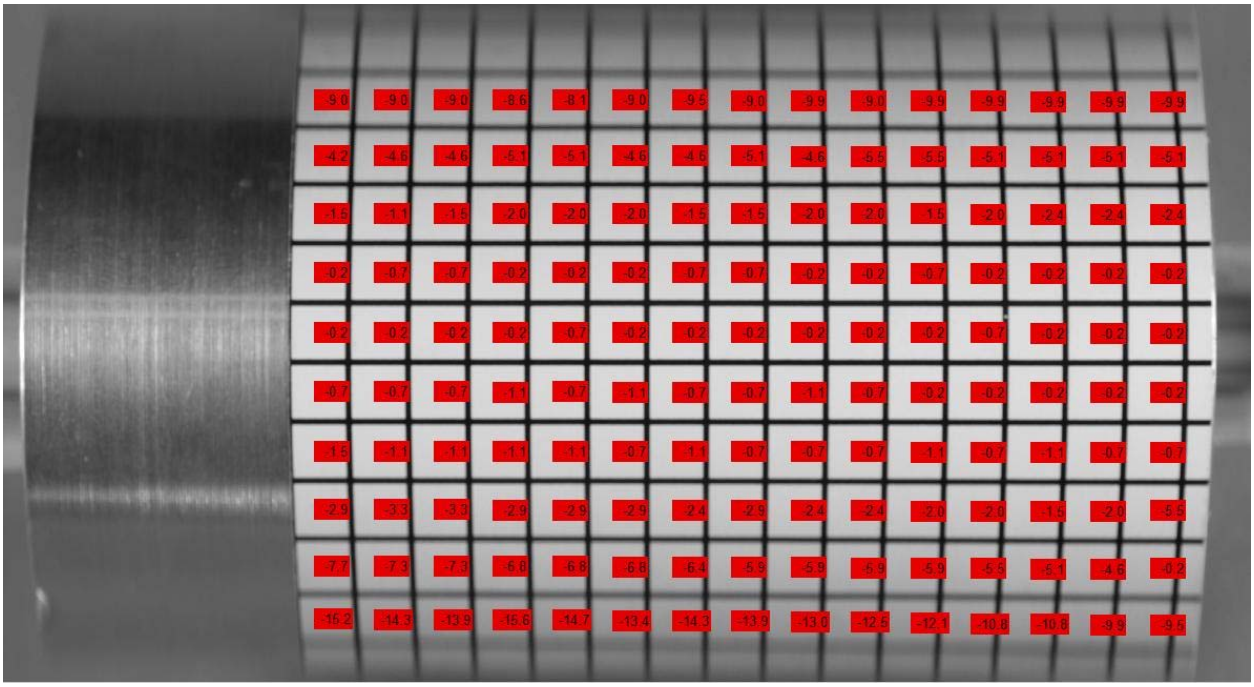


Figure 63: Error in gridline spacing after unwrapping image using Canny Edge Detection

The entire iterative process has been coded so that the user can rapidly calculate the percent error in gridline spacing at multiple points across an image with the grid wrapped core. For this project the percent error in gridline spacing is measured for three unwrapped images and the original wrapped image. Each of the three unwrapped images is unwrapped using one of the three edge detection methods; visual, gradient and canny filtered.

Error Database and Visualization

The database contains a separate file for the analysis of each of the three unwrapped images. The user can access a file and plot the error data using one of several figures to show the distribution of error across the image. These figures are a 3D bar chart, contour plot, and a scatter plot.

Each data point in a population of errors has an XYZ coordinate where X is the column #, Y is the row #, and Z is the percent error. During the error estimation process, the user selected a

single column of which several data points were calculated. Data points along this column will obviously share a single column coordinate, while the row coordinate is unique for adjacent data points from one column to the next column.

	User Selected Column - 1	**User Selected Column - 2**
Row Number	300	302
	506	508
	834	836

The 3D bar chart plots the percent error , in the z-axis, along each user selected column at a common row. In the 3D bar chart the row number from adjacent points across multiple columns cannot be unique values. In order to find a common row number for points in adjacent columns the row numbers are averaged to find a unique row number for plotting.

	User Selected Column - 1	**User Selected Column - 2**	**Average Row Number**
Row Number	300	302	301
	506	508	507
	834	836	835

For the typical unwrapped image seen through this paper there are 15 user-selected column where each has column has 11 gridlines which produce 10 error values. The 3D bar chart below has 15 columns and 10 rows. The code will generate the figure in a plot editor window where the user can edit and annotate the figure before saving it to an image file.

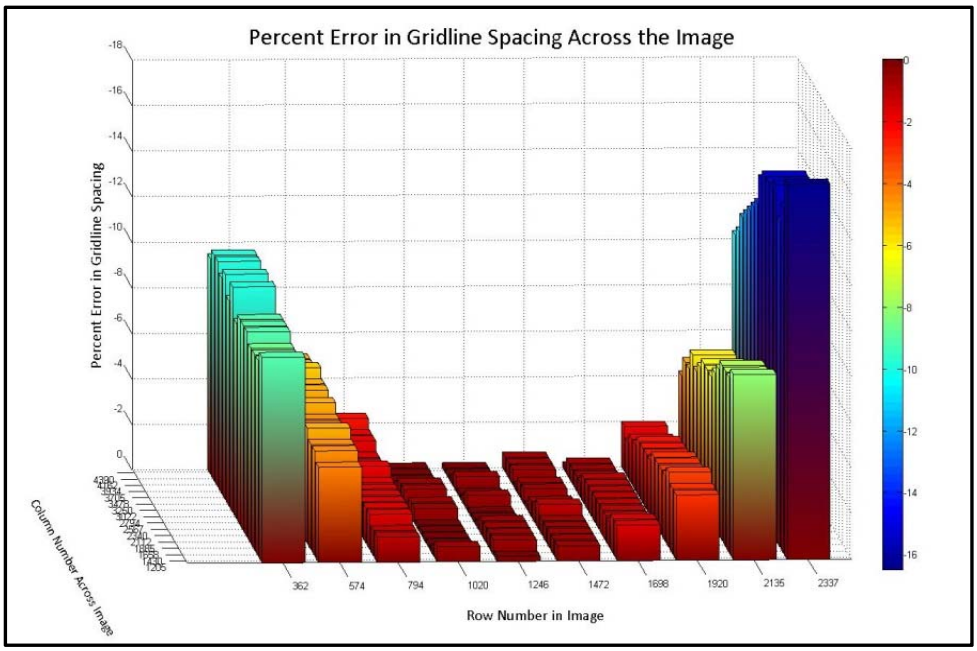


Figure 64: 3D bar chart of error in gridline spacing, negative indicates gridline spacing is less than expected

Another format for presenting the distribution of error data across the image is as a filled contour plot. Unlike the 3D bar chart the unique XYZ coordinates for each data point can be used to create the shaded contour plot.

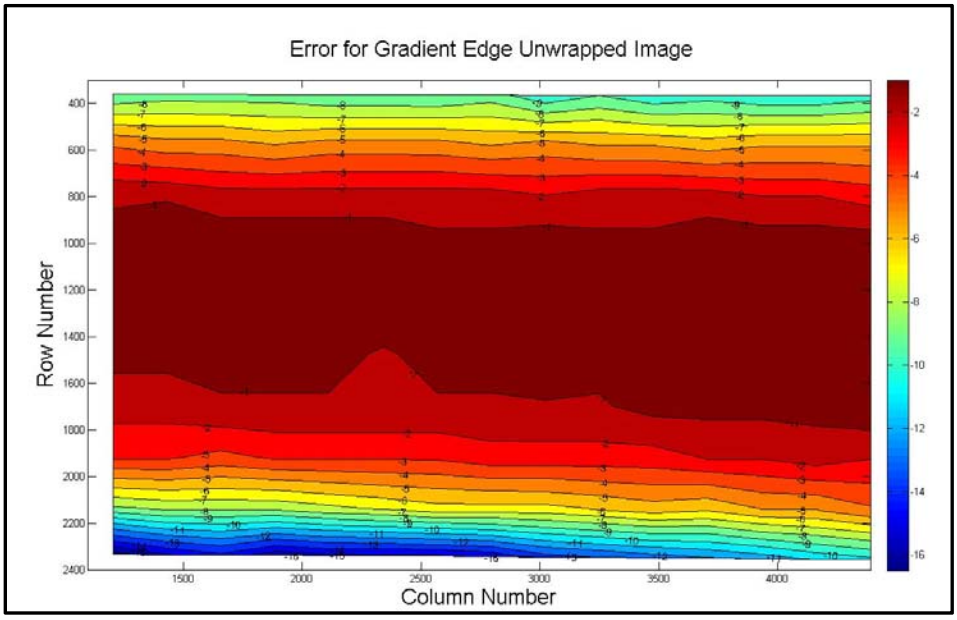


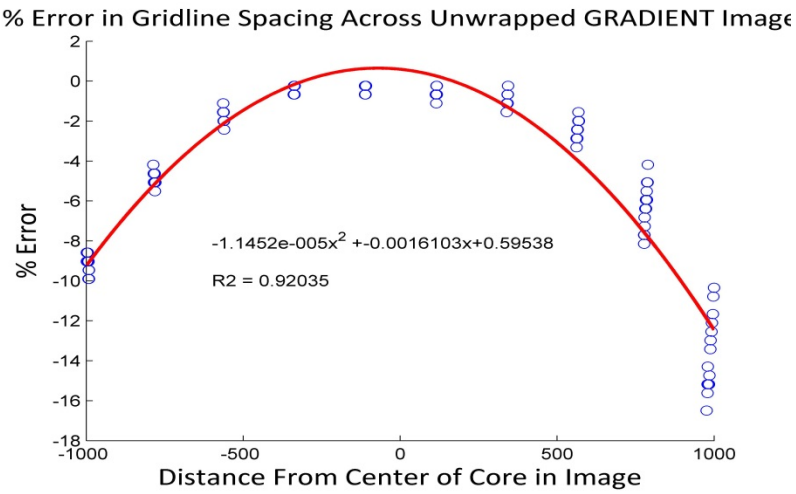
Figure 65: Filled Contour Plot of gridline spacing error, neg. indicates gridline spacing is less than expected

Again, the code will generate the contour figure in a plot editor window were the user can edit and annotate the figure before saving it to an image file.

Each of these figures allows the user to easily visualize the distribution of error in gridline spacing across the image. The plots show an area near the center of the image where the error is less than -1. The negative value indicates that the distance between adjacent horizontal gridlines is less than the known distance. The error tends to increase as the distance from the center of the image increases. These plots may allow the user to quickly identify an area of the image with a user defined acceptable level of error.

Quantifying Error

The figures above give the user a quick visual inspection of the distribution of error across the unwrapped image. The general trend of the error values is that they increase as a function of distance from the center of the image. By normalizing the location of all of the error data point with respect to the center of the unwrapped image, a scatter plot can be created. A polynomial trend line can be fit to the data and the coefficients of the trend line can be used to create a function. The function of the trend line can be used to quantitatively estimate the locational error of any point in the vertical direction of the image as a function of the point from the center of the unwrapped image. The negative value indicates that the distance between adjacent horizontal gridlines is less than the known distance.



Each data point in a population of errors has an XYZ coordinate where X is the column #, Y is the row #, and Z is the percent error. The row number is measured from the top left of the image. In order to normalize the row numbers with respect to the center of the unwrapped image, the row number of each data point is subtracted by the location of the center of the unwrapped image. A negative row number indicates that a data point is in the upper half of the image.

Center of Image Row- 475	**Original Row Coordinate**	**Normalized Row Coordinate**
Row Number	300	(300-475) = -175
	506	(508-475) = 33
	834	(836-475) = 361

Once the distribution of errors has been normalized with respect to the center of the image the entire population of row numbers and error values is used to calculate a 2nd order polynomial equation using a least square fit. (Matlab-Documentation, polyfit) The coefficient of determination (R^2) is also calculate using the original magnitude of the error across the normalized row locations, for reference.

The coefficients of the polynomial equation can be used to calculate the expected error in a measure of distance to a point or region in the unwrapped image from the center of the image.

$$Error\ at\ a\ Point = ax^2 + bx + C;$$

Where x is the distance of a point from the center of an image, and abc are the coefficients of the second order polynomial. The following table shows the results of applying this formula to a sample set of data from the image.

Image Size	2578x 4752 (r,c)	% Error=ax2 +bx+c	
Row Center	1290	R2 = 0.92	
Coefficients	a= -1.452e-5, b = -0.0016103, c=0.59538		
Row	Normalized	Error	% From Center
1	-1284.5	-21.3	50
642	-643.5	-4.4	25
1221	-64.5	0.6	5
1285.5	0	0.0	0
1349	63.5	0.4	5
1928	642.5	-6.4	25
2569	1283.5	-25.4	50

The row numbers in the table above were purposefully selected so that they are mostly symmetrical about the center row of the image. It is shown that by selecting some percentage of the image around the center of the image a region can be defined such that the distance between two points measured within that region can be assured to have an error equal to or less than some value. The table above represents an image of the core unwrapped using the gradient edge detection method. For example, the table shows that the region between rows 642-1928 represents 50% of the image around the center of the image, (25% +25%). For this region the user should expect that there will be a maximum error of 6.4%. This percentage means that there

will be +/- value in inches to the final measure. To find the percentage as a value of inches the original error formula will be used to back calculate the know measure at the row 1928, normalized to row 642.5.

$$Measured\ Spacing(inches) = MeasuredSpacing\ (rows) \times Scale\ (in/pixel)$$

$$\%\ Error = ((Measured\ Spacing(in.) - Known\ Spacing(in.))/Known\ Spacing(in.)) \times 100$$

Recall that the measured row value of 1928 is normalized to row 642.5 and error at the point is 6.4. Converting the row value of 642.5 to inches, using the scale, yields a value of 0.706 inches. This measured value is used to solve for the known spacing to get a value of 0.664 inches. The difference in the measured and true, or known, value is 0.04 inches. This says that the distance between the center of the image and a point at the edge of the bounds that encompasses 50% of the image should be off by 0.06 inches. There will be distances between points inside this region with much smaller error but this region is said to have a maximum error value of 6.4%. A rating table can be set up to show the percentage of the image as a function of error and this error is back calculated to show the potential error in the distance measure in inches for the image used in the analysis.

Image Size	2578x 4752 (r,c)	Row Center	1290	Unwrapped
Row	Normalized	Error	% of Image	Error (Inches)
1	-1289	-21.45	100	1.17
645.5	-644.5	-4.40	50	0.68
967.75	-322.25	-0.39	25	0.35
1225.5	-64.5	0.64	5	0.07
1290	0	0.00	0	0.00
1254.5	-35.5	0.63	5	0.04
1612.25	322.25	-1.43	25	0.35
1934.5	644.5	-6.47	50	0.67
2578	1288	-25.57	100	1.13

The table shows that for the image unwrapped using the gradient edge detection the maximum error in any vertical distance measurement using the entire image would be no more than 1.13 inches. There error could be reduced by using a percentage of the entire image, say 25% around the center of the image and the error would be 0.35 inches. These values were generated for a single unwrapped image the used the gradient edge detection method.

A quick comparison between the three edge detection methods can be done by generating three polynomial equations and a rating table to compare the results of the three edge detection methods.

Table XIII: Comparison of edge detection methods on unwrapping

		Coefficients			
Unwrap-Method	Size of Image	**a**	**b**	**c**	R^2
Gradient	(2578x4752	-1.45E-05	-0.0016103	0.59538	0.92
Canny	(2578x4752	-1.11E-05	-0.0012657	0.53465	0.91
Visual	(2578x4752	-1.14E-05	-0.0020531	0.56121	0.92
			Error		
% of Image	Row	Normalized	Visual	Gradient	Canny
100	1	-1289	-15.8	-21.45	-16.2
50	645.5	-644.5	-2.9	-4.4	-3.3
25	967.75	-322.25	0.03	-0.39	-0.2
5	1225.5	-64.5	0.6	0.64	0.6
0	1290	0	0	0	0
5	1254.5	-35.5	0.6	0.63	0.6
25	1612.25	322.25	-1.3	-1.43	-1.0
50	1934.5	644.5	-5.5	-6.47	-4.9
100	2578	1288	-21.1	-25.57	-19.5

Table XIII shows that the gradient edge detection produced consistently higher error value in the final unwrapped image, but the values may not be significantly greater than values produced by the other two methods.

Wrapped vs. Unwrapped

The error data was calculated across several rows and columns of the image. This data can be plotted in two directions using a box plot. The first direction is across several rows of data where adjacent points across a row have a different unique column numbers. The second direction is down several columns of data were adjacent data points have the same column numbers, but a unique row number.

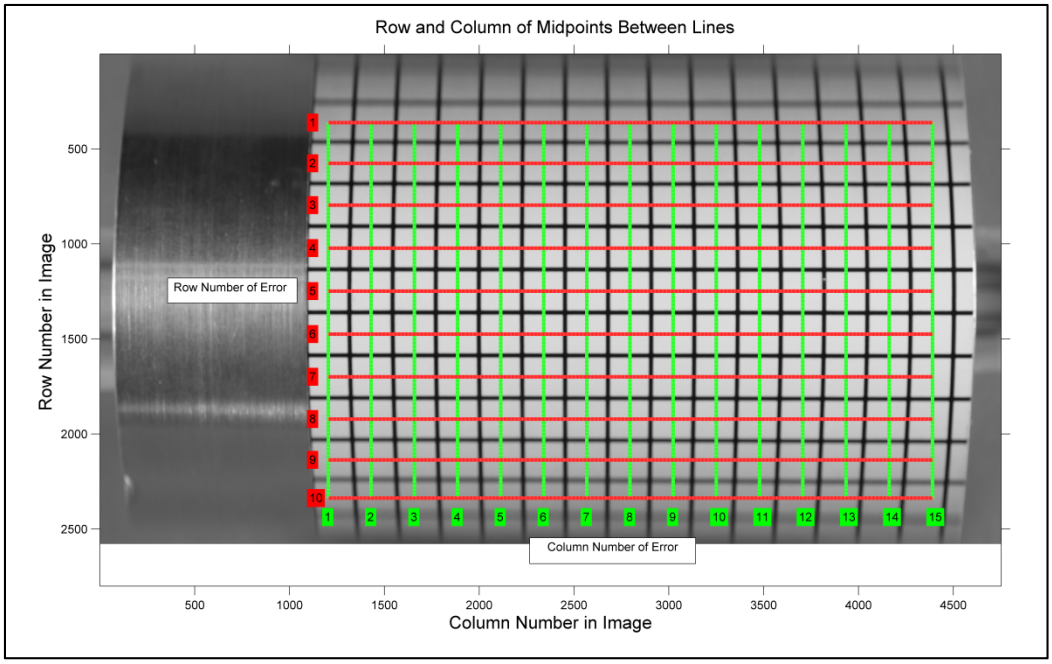


Figure 66: Location of each gridline spacing calculation

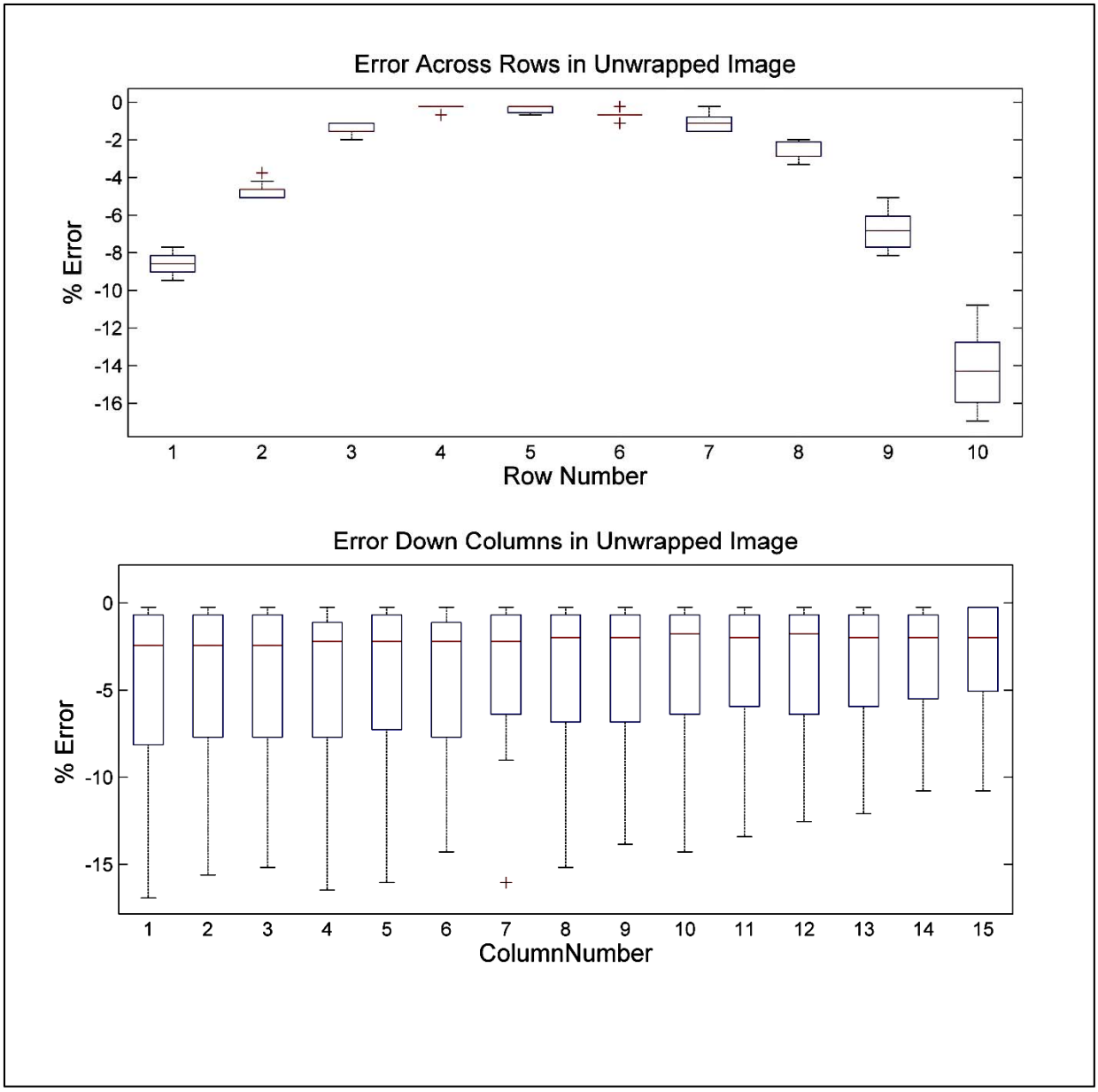


Figure 67: Box and whisker plot of error for unwrapped image

The box plots above in Figure 67 are from an image that was unwrapped using visual inspection to determine the edges of the core before unwrapping the original image. The red line of each box indicates the mean error for the entire column or row. The blue box outlines the 95% of the error and the dashed lines indicate the range of the error values across a column or row. Before commenting on the plots, it is useful to generate another set of box plots showing the original wrapped image side-by-side with the unwrapped box plot.

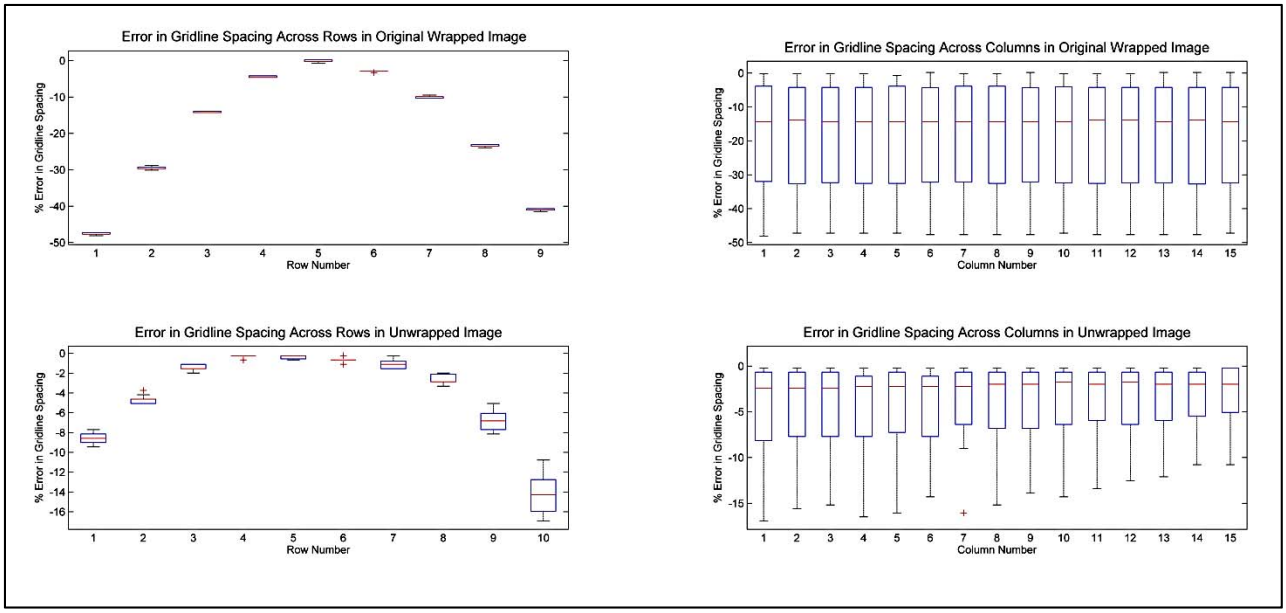


Figure 68: Box and whisker plots of gridline spacing error for original and unwrapped image

Conclusion

The procedure to evaluate the unwrapped image starts by locating the distance between horizontal lines at several user-selected columns. These distances are compared to a, known or true distance to compute the error in the measured distance. This error is expressed as a percentage with negative values indicating that the distance between gridlines in the image are less than a known value. The negative, values indicate that the entire area of the unwrapped image is not a true planar representation of the outer surface of the cylinder.

The population of errors is then saved to a database from which three figures can be produced along with a scatter plot that shows the coefficients of a 2^{nd} order polynomial function that can be used to calculate the error at any point in the unwrapped image, and not between gridlines. This equation can be used to find bounds of areas around the center of the image that may have an acceptable error for the user. The user can create a rating table to further refine the

decision. Further work could use the polynomial equation to create a correction for any two points measured at any point of the image.

A brief analysis of the three edge detection methods showed that all three methods produce comparable results and any of the three methods could be used to successfully unwrap an image.

CONCLUSION

This paper presents ideas, concepts, and techniques that can be used to unwrap an image of a cylindrical object. Specifically, these methods were developed with the idea that they could be applied to images of rock core in a controlled setting. Several pieces of code have been developed that can be used in succession to unwrap an image.

The procedure for unwrapping an image starts with the direct measurement of the diameter/radius of a cylindrical core. A grid pattern with known spacing is applied to core and an image is taken such that the camera is directly over the core. Next, the radius of the core and the known spacing of the pattern are entered into a piece of code which aids the user in determining the scale of a row, or pixel in the image. Once this scale is found the user has the option of unwrapping the cylinder using one of three edge detection methods. After the image is unwrapped it is saved as an image file. After the unwrapping is complete the user can call another piece of code that will determine the percent error in gridline spacing across the unwrapped image at discrete user specified points. The population of errors along with their location within the image, are saved to a data base. This database can then be queried to produce several figures showing the distribution of error across the image. Along with the figures, the distribution of errors is fitted with a 2^{nd} order polynomial equation that can be used to predict

error at any point within the image. This equation could be used to create a rating table to demarcate areas of the image with an acceptable level of error for future uses.

The goal of this project was to create an inexpensive method for unwrapping images of cylindrical rock cores in the lab. The procedures described above could be used to unwrap images if at least one image contains the calibrating gridline pattern, as the unwrapping procedure only requires that the radius and pixel scale be known. The radius and scale of a pixel is not expected to change between images as long as the camera remains in a fixed position over the core.

All in all the procedure is relatively inexpensive to implement, and it is relatively easy to run the associated code on a personal computer.

Recommendations for future work

This project was originally envisioned as just the first part in a series of software modules that would eventual provide the user with a set of tools for quantifying the distribution of voids present on the outer surface of a rock core. In order to capture the entire surface of the core, several images would need to be first unwrapped and then stitched together to form a continuous image of the outer surface. Once a continuous image was created, thresh-holding techniques could be applied to the image to isolate voids from the matrix material of the rock. From there statistical tools available in Matlab could be applied to quantify the shape, size, proximity and distribution of voids on the outer surface of the core.

Ultimately it was envisioned that this information could be used to develop a non-destructive method for quantifying the strength and stiffness of a given material without the need for further destructive testing like unconfined compression, or tri-axial compression testing.

Bibliography

Apostal, T. M., & Mnatsakanian, M. A. (2007). Unwrapping Curves from Cylinders and Cones. *The Mathematical Association of America Monthly, 114*, 388-416.

Eddins, S. (n.d.). *Blogs*. Retrieved January 14, 2014, from Mathworks: Steve Eddins, http://blogs.mathworks.com/steve/2006/01/31/spatial-transformations-terminology-and-notation/

Erfourth, B. S. (2006). Charecterization of the imapct of shperical voids-regarding size, spatial distribution, and porosity- on engineering properties of rock and rock-like materials. *Thesis*. Butte, MT: Montana Tech of The University of Montana.

Gonzalez, R. C., Woods, R., & Eddins, S. I. (2009). *Digital Image Processing Using Matlab, 2nd ed.* Gatesmark publishing.

Hemmleb, M., & Weidman, A. (1997). Digital Rectification and Generation of Orthoimages in Architectural Photogrammetry. *Proc. of the CIPA Int. Symposium '97,Photogrammetry in Architecture, Archaeology and Urban Conservation, International Archives for Photogrammetry and Remote Sensing (IAPRS).*, (pp. 261-267). Göteborg,Schweden.

Karras, E. G., Oatias, P., & Ketipis, K. (1997). Raster Projection and Development of Curved Surfaces. *International Archives of Phtotgrammetry & Remote Sensing*.

Karras, G. E., Patias, P., & Petsa, E. (1996). Digital Monoplotting and Photo-Unwrapping of Developable Surfaces in Architectural Photogrammetry. *International Archives of Photogrammetry & Remote Sensing, 31*(5), 290-294.

Pavelka, K., Ruzicka, S., & Bila, Z. (2013). Photo-Plan Creation of Cylindrical Objects. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume II-5/W1* (pp. 229-234). Strasborg,France: XXIV International CIPA Symposium,.

Schepers, R., Rafat, G., Gelbke, C., & Lehmann, B. (2001). Application of borehole logging, core imagin and tomography to geotechnical exploraton. *International journal of Rock Mechanics and Mining Sciences*, 867-876.

Schepers, R., Rafat, G., Gelbke, C., & Lehmann, B. (2001). Application of borehole logging, core imaging, and tomography to geotechnical exploration. *International Journal of Rock Mechanics and Mining Sciences*, 867-876.

Tanner, J. (2014). *Visible_fraction_of_surface*. (PHP Science Labs , Producer) Retrieved April 30, 2014, from Neoprogrammics: http://www.neoprogrammics.com/spheres/visible_fraction_of_surface.php

Wolberg, G. (1990). *Digital Image Warping.* Los Alamitos, CA: IEEE Computer Society Press.

Appendix

MATLAB CODE

```
SCALE FINDER CODE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This code produces a pixel scale in inches/pixel for use in
the unwrapping.
% The user needs to enter the image name by editing the
filename: i.e.
%
%     filename = '0025'; Modify to: filename = 'Your Image File
Name Here';
%
% The user also needs to enter original girdline spacing inches:
i.e.
%
%     spacing = 0.25;  Modify to:  spacing = Your Gridline
Spacing in inches;
%
% The code will automatically save pixel scale and gridline
spacing as .mat with appended image file name:
%
%      Pixel Scale for (Your Image File Name).mat
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


clear all


filename =uigetfile('*.jpg'); %User input File Name
spacing = 0.25;                 % User input, original gridline
spacing, inches
Radius = 1.062;

Theta = (spacing/(2*pi*Radius))*360;
Cord = 2*Radius*sind(Theta/2);

fmt = 'jpg';
I = imread(filename);       %Read in image
if size(I,3) == 3;              %Check to see if image is RGB
I = rgb2gray(I);                %Convert RGB image to grayscale
end
```

```
imshow(I)                          %Display grayscale image
cutoff = 115;
Search1 =  160;
Search2 = 2410;

hp = impixelinfo;                          % Tool Prompts the user
for the column to be analyzed
set(hp,'Position',[150 150 300 20]);     % Shows Current
Position of curser
H = impoint;                               % Get Current position
of cursor
Point = wait(H);                           % Wait for user to
double click cursor

% Text Box prompts user to either change or accept column
position, and
% asks if user wants to proceed. User enters N to cancel code or
OK to
% proceed.
prompt = {'Enter Column for analysis:', ' Do you want to
continue Analysis: (Y/N)'};
dlg_title = 'Column Number';
num_lines = 1;
P = num2str(round(Point(1)));
def = {P,'Y'};
answer = inputdlg(prompt,dlg_title,num_lines,def);
column = str2num(answer{1});
choice = answer{2};
i = 1;

% Gets User Selected Column and plots as bar chart. Waits for
user to manipulate search
% area line.User double clicks line when satisfied with
position.
Column = I(:,column,:);
Column = cast(Column, 'double');
hparent = bar(Column,'DisplayName','I');figure(gcf)
h = imline(hparent,[Search1 cutoff; Search2 cutoff]);
setPositionConstraintFcn(h,@(pos) [pos(:,1)
repmat(mean(pos(:,2)),2,1)])
accepted_pos = wait(h);
pause(1)

Search1 = round(accepted_pos(1,1));
Search2 = round(accepted_pos(2,1));
cutoff = round(accepted_pos(1,2));
```

```
%Calculate the cutoff positions for Top and Bottom of each
gridline.
valve = 0;                        %Valve is the switch variable
l = 1;
r = 1;

for y = Search1:Search2
   if Column(y)>= cutoff && valve == 0 && Column(y+1) < cutoff
% I(y) is 0:255 value. If controls when we go down one of the
valleys
      Left(l) = y+1;                    %Really just storing Y
coordinate of point
      l=l+1;                          %increment i
      valve = 1;                      %Close this if until we get
to other side of valley

       end
   if Column(y) >= cutoff && valve >= 1  && Column(y-1) < cutoff
%Other side of valley
      Right(r)= y-1;                     %Record Y coor of point
before > cutoff
      r= r+1;                           %Increment i
      valve = 0;                        %Reset switch for next
valley
   end

end

counter = 0;

% Calculate the distance between each gridline, find max
distance
% between gridlines and use this value to calculate the pixel
scale

Data(1,:) = [ column  Right];
Data(2,:) =[ cutoff Left ];

MidpointLines = Left+ round((Right- Left+1)/2);
Data(3,:) = [Search1 MidpointLines];

Delta = diff(MidpointLines);
Data(4,:) = [ Search2 0 Delta];

DD = round(Delta/2);
MidptBetweenGridlines = bsxfun(@plus,MidpointLines(:,1:end-
1),DD);
```

```
Data(5,:) = [ 0 0 MidptBetweenGridlines];
FinalData{i} = Data;

[Max Location] = max(Data(4,3:end));
Scale = Cord/Max;

% Plot value of scale at location between gridlines
RROW = Data(5,Location+2);
imshow(I)
hold on
text(column,RROW, sprintf('%d',Scale),'BackgroundColor',[.9 .0
.0]);
hold off

%Save gridline scale and gridline spacing to variable Scale.
%Append original filename to end of .mat variable name.
header = { 'Scale', 'Spacing', 'Radius_in';};
records = [  Scale  spacing  Radius ];
ds = dataset({records,header{:}});
ds.Filename={filename};
ds.Column = column;
ds.Cutoff = cutoff;
ds.Gridline_Left = [Left]';
ds.Gridline_Right = [Right]';
ds.MidPoints = [MidptBetweenGridlines 0]';
ds.Max = [Max];
ds.Search_Range = [Search1 Search2]';
DATA{1} = ds;

[VV filename Vv] = fileparts(filename);
SaveDataAs = ['Dataset for Image ' filename ];
save(SaveDataAs,'DATA')
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This code unwraps the cylindrical core Given USER DEFINED CORE
EDGES
% and saves a lossless jpg image.
% This particular version of  unwrapping code requires several
user inputs.
%
%  The user needs to enter the image name by editing the
filename: i.e.
%
%     filename = '0025'; Modify to: filename = 'Your Image File
Name Here';
%       Do not include '.jpg' in filename
%
% The user also needs to define the pixel scale in inches/pixel
%
%     scale = 0.00110132; Modify to: scale = Your pixel scale
Here;
%
% The user needs to enter the core radius in inches.
%
%     Radius = 1.062; Modify to: Radius = Your core radius Here;
%
% The user needs to enter the Top and Bottom Row number to
define the edges
% of the core in the original image. Recommend using command:
%     imshow('Your Image Name.jpg'); Use Data Tip Tool and
record Row number
%     of horizontal edges of core.
%
%     Top = 667;  Modify to Top = You Row Number Here;
%     Bottom = 2545; Modify to Bottom = Your Row Number Here;
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


filename1 = uigetfile('*.mat');
load(filename1,'-mat');
ds = DATA{1};
filename =uigetfile('*.jpg'); %User input File Name

scale = ds.Scale(1);
radius = ds.Radius_in(1);
Radius = radius/scale;
Top = 667;
```

```
Bottom = 2545;

% Read in user defined image file. Convert file to Grayscale if
neccesary
Image = imread(filename);
if size(Image,3) == 3;
Image = rgb2gray(Image);
end

Cropped = Image(Top:Bottom,:);  % Crop out Core based on user
imputs


% Check to see if Cropped image has even or odd number of rows,
divide
% image into two parts for unwrapping

is_image_odd_or_even_sized = mod(size(Cropped,1),2);

if is_image_odd_or_even_sized  ==0;  %0 means no remander so
image is even
   TopHalf = Cropped(1:(floor(size(Cropped,1)/2)),:);
   BottomHalf = Cropped((round(size(Cropped,1)/2)):end,:);
   iseven = 1; %1 Means Image has even # of rows
   C = 0.5;
else  %Image has ODD number of rows
   TopHalf = Cropped(1:((size(Cropped,1)/2)),:);
   BottomHalf = Cropped(((size(Cropped,1)/2)):end,:);
   iseven = 0; %0 Means Image has odd # of rows
   C = 0;
end


%Perform unwrapping seperatly to top and bottom half of image
and
% recombine halves to create full unwrapped image

% Define Transforms
T = [ 1 0 0; 0 -1 0; 0 0 1];           % This mirrors top half
tform_mirror = maketform('affine',T);
forward_fcn = @(xy,tdata)[xy(:,1),(Radius * (pi/2-
acos(((xy(:,2)-C)/Radius))))]; %Inverse Unwrapping
inverse_fcn = @(xy,tdata)[xy(:,1), ((Radius)*cos(pi/2-
((xy(:,2)+C)/Radius)))];     %Forward unwrapping
tform_unwrapp = maketform('custom',
2,2,forward_fcn,inverse_fcn,[]);                      %Combine
forward and Reverse to Create Uwnrapping Transform
```

```
% Apply transforms to Top and Bottom Half of Images seperatly
TopHalf = imtransform(TopHalf,tform_mirror);    % Mirror Top
half
TopHalf = imtransform(TopHalf,tform_unwrapp);   % Unwrapp Top
Half
TopHalf = imtransform(TopHalf,tform_mirror);    % Mirror Top
half Again
BottomHalf = imtransform(BottomHalf,tform_unwrapp); %Unwrapp
Bottom Half

%Recombine Top half and Bottom Half to produce Full Unwrapped
Image
if iseven == 1;                          %Original Image had even
# of rows
    temp= vertcat(TopHalf,BottomHalf);
else if iseven == 0;                     %Original Image had odd
# of rows
    TopHalf = TopHalf(1:(end-1),:);
    temp= vertcat(TopHalf,BottomHalf);
end
end
[VV filename Vv] = fileparts(filename);
fmt ='jpg';
%Write Unwrapped Image to new Lossless jpg File
temp3 = cast(temp,'uint8');
temp4 = cast(temp,'uint8');
filename3 = ['Unwrapped Visual ' filename '.' fmt];
filename4 = ['LL Unwrapped Visual ' filename '.' fmt];
imwrite(temp3,filename3);
imwrite(temp4,filename4,'Mode','lossless');


%Write data to file
Edge1 = 'Visual';
header = { 'Filename' ,'Edge Detection' };%Scale', 'Spacing',
'Radius_in';};
records = { filename, Edge1 };% ds.Scale  ds.Spacing
ds.Radius_in ];
ds2.Number = 3;
ds2 = dataset({records,header{:}});
ds2.Scale = ds.Scale;
ds2.Spacing = ds.Spacing;
ds2.Radius_in = ds.Radius_in;
ds2.Radius_row = Radius;
ds2.Size_Wrapped = size(Image);
ds2.Is_Even = iseven;
```

```
ds2.Edge = 'Gradient';
ds2.Top_Bottom_Canny = [Top Bottom];
ds2.Size_Cropped = [(Bottom- Top+1) size(Image,2)];
ds2.Size_Unwrapped = size(temp);
DATA{2} = ds2;

SaveDataAs = ['Dataset for Image ' filename ];
save(SaveDataAs,'DATA')
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%EDGE DETECTION: CANNY EDGE DETECTION METHOD
%This code unwrapps the cylindrical core and saves a lossless
jpg image.
% This particular version of  unwrapping code requires several
user inputs.
%
%  The user needs to enter the image name by editing the
filename: i.e.
%
%     filename = '0025'; Modify to: filename = 'Your Image File
Name Here';
%       Do not include '.jpg' in filename
%
% The user also needs to define the pixel scale in inches/pixel
%
%     scale = 0.00110132; Modify to: scale = Your pixel scale
Here;
%
% The user needs to enter the core radius in inches.
%
%     Radius_inches = 1.062; Modify to: Radius_inches = Your
core radius Here;
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all

filename1 = uigetfile('*.mat');
load(filename1,'-mat');
ds = DATA{1};
filename =uigetfile('*.jpg'); %User input File Name
[VV filename Vv] = fileparts(filename);
fmt = 'jpg';
scale = ds.Scale(1);
radius = ds.Radius_in(1);
Radius = radius/scale;

% Read in user defined image file. Convert file to Grayscale if
neccesary
Image = imread(filename,fmt);
if size(Image,3) == 3;
Image = rgb2gray(Image);
end
```

```
% Cropping with Canny edge detection filter
bw = edge(Image,'canny');
%imshow(bw)

for i = 1:size(bw,2)
    X = bw(:,i);
    [row, col] = find(X>0);
  TF = isempty(row);
  if TF ==0
   Amax(i) = max(row);
   Amin(i) = min(row);
  end
end

Center = round(size(bw,2)/2);
Range = 1000;
Top = median(Amin(1,Center-Range:Center+Range));
Bottom = median(Amax(1,Center-Range:Center+Range));

Cropped = Image(Top:Bottom,:);

% Check to see if Cropped image has even or odd number of rows,
divide
% image into two parts for unwrapping

is_image_odd_or_even_sized = mod(size(Cropped,1),2);

if is_image_odd_or_even_sized  ==0;  %0 means no remander so
image is even
   TopHalf = Cropped(1:(floor(size(Cropped,1)/2)),:);
   BottomHalf = Cropped((round(size(Cropped,1)/2)):end,:);
   iseven = 1; %1 Means Image has even # of rows
   C = 0.5;
else  %Image has ODD number of rows
   TopHalf = Cropped(1:((size(Cropped,1)/2)),:);
   BottomHalf = Cropped(((size(Cropped,1)/2)):end,:);
   iseven = 0; %0 Means Image has odd # of rows
   C = 0;
end


%Perform unwrapping seperatly to top and bottom half of image
and
% recombine halves to create full unwrapped image

% Define Transforms
```

```
T = [ 1 0 0; 0 -1 0; 0 0 1];           % This mirrors top half
tform_mirror = maketform('affine',T);
forward_fcn = @(xy,tdata)[xy(:,1),(Radius * (pi/2-
acos(((xy(:,2)-C)/Radius))))]; %Inverse Unwrapping
inverse_fcn = @(xy,tdata)[xy(:,1), ((Radius)*cos(pi/2-
((xy(:,2)+C)/Radius)))];     %Forward unwrapping
tform_unwrapp = maketform('custom',
2,2,forward_fcn,inverse_fcn,[]);                      %Combine
forward and Reverse to Create Uwnrapping Transform

% Apply transforms to Top and Bottom Half of Images seperatly
TopHalf = imtransform(TopHalf,tform_mirror,'nearest');    %
Mirror Top half
TopHalf = imtransform(TopHalf,tform_unwrapp,'nearest');   %
Unwrapp Top Half
TopHalf = imtransform(TopHalf,tform_mirror,'nearest');    %
Mirror Top half Again
BottomHalf = imtransform(BottomHalf,tform_unwrapp,'nearest');
%Unwrapp Bottom Half

%Recombine Top half and Bottom Half to produce Full Unwrapped
Image
if iseven == 1;                         %Original Image had even
# of rows
    temp= vertcat(TopHalf,BottomHalf);
else if iseven == 0;                    %Original Image had odd
# of rows
    TopHalf = TopHalf(1:(end-1),:);
    temp= vertcat(TopHalf,BottomHalf);
end
end

%Write Unwrapped Image to new Lossless jpg File


temp3 = cast(temp,'uint8');
temp4 = cast(temp,'uint8');
filename3 = ['Unwrapped Canny ' filename '.' fmt];
filename4 = ['LL Unwrapped Canny ' filename '.' fmt];
imwrite(temp3,filename3);
imwrite(temp4,filename4,'Mode','lossless');

%Write data to file
Edge1 = 'CANNY';
header = { 'Filename' ,'Edge Detection' };%Scale', 'Spacing',
'Radius_in';};
```

```
records = { filename, Edge1 };% ds.Scale  ds.Spacing
ds.Radius_in ];
ds4 = dataset({records,header{:}});
ds4.Number = 5;
ds4.Scale = ds.Scale;
ds4.Spacing = ds.Spacing;
ds4.Radius_in = ds.Radius_in;
ds4.Radius_row = Radius;
ds4.Size_Wrapped = size(Image)';
ds4.Is_Even = iseven;
ds4.Top_Bottom_Canny = [Top Bottom];
ds4.Size_Cropped = [(Bottom- Top+1) size(Image,2)]';
ds4.Size_Unwrapped = size(temp)';
DATA{4} = ds4;
SaveDataAs = ['Dataset for Image ' filename ];
save(SaveDataAs,'DATA')
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%EDGE DETECTION: GRADIENT METHOD
%This code unwrapps the cylindrical core and saves a lossless
jpg image.
% This particular version of  unwrapping code requires several
user inputs.
%
%  The user needs to enter the image name by editing the
filename: i.e.
%
%     filename = '0025'; Modify to: filename = 'Your Image File
Name Here';
%       Do not include '.jpg' in filename
%
% The user also needs to define the pixel scale in inches/pixel
%
%     scale = 0.00110132; Modify to: scale = Your pixel scale
Here;
%
% The user needs to enter the core radius in inches.
%
%     Radius = 1.062; Modify to: Radius = Your core radius Here;
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


filename ='0025'; %User input File Name
fmt = 'jpg';


filename1 = uigetfile('*.mat');
load(filename1,'-mat');
ds = DATA{1};
filename =uigetfile('*.jpg'); %User input File Name

scale = ds.Scale(1);
radius = ds.Radius_in(1);
Radius = radius/scale;
%scale = 0.00110132;  % User input , image scale, inches per
pixel.
%Radius = 1.062;
```

```
% Read in user defined image file. Convert file to Grayscale if
neccesary
Image = imread(filename,fmt);
if size(Image,3) == 3;
Image = rgb2gray(Image);
end

j = 1;
for Column = 1500:2500

x1 = Image(:,Column);
x1 = cast(x1,'double');

k = diff(x1)';
SampleSize = 500;
%%%Top edge detection

LeftSideNoiseMin = min(k(:,1:SampleSize));
LeftSideNoiseMax = max(k(:,1:SampleSize));

RightSideNoiseMin= min(k(:,(end -SampleSize):end));
RightSideNoiseMax= max(k(:,(end -SampleSize):end));

%%%Left Side Min Max Detection
valve = 1;
valve2 = 1;
for i = 1:round(size(k,2)/2)

    if valve == 1 && (k(i) > LeftSideNoiseMax)
      TopMax = i;
      valve = 0;
    end
    if valve2 == 1 && k(i) < LeftSideNoiseMin
       TopMin = i;
       valve2 = 0;
    end
end
valve = 1;
valve2 = 1;
for i = size(k,2): -1:round(size(k,2)/2)

    if valve == 1 && k(i) > RightSideNoiseMax
      BottomMax = i;
      valve = 0;
    end

    if valve2 == 1 && k(i) < RightSideNoiseMin
```

```
            BottomMin = i;
            valve2 = 0;
    end
end

c(j, 1:4) = horzcat(TopMax,TopMin,BottomMax,BottomMin);

for i = 1:4
    m(i,1) = mode(c(:,i));
    m(i,2) = std(c(:,i));
end
j = j+1;
end

%%Cropping
Top= m(1,1);
Bottom = m(4,1);
Cropped = Image(Top:Bottom,:);


% Check to see if Cropped image has even or odd number of rows,
divide
% image into two parts for unwrapping

is_image_odd_or_even_sized = mod(size(Cropped,1),2);

if is_image_odd_or_even_sized  ==0;  %0 means no remander so
image is even
   TopHalf = Cropped(1:(floor(size(Cropped,1)/2)),:);
   BottomHalf = Cropped((round(size(Cropped,1)/2)):end,:);
   iseven = 1; %1 Means Image has even # of rows
   C = 0.5;
else  %Image has ODD number of rows
   TopHalf = Cropped(1:((size(Cropped,1)/2)),:);
   BottomHalf = Cropped(((size(Cropped,1)/2)):end,:);
   iseven = 0; %0 Means Image has odd # of rows
   C = 0;
end


%Perform unwrapping seperatly to top and bottom half of image
and
% recombine halves to create full unwrapped image

% Define Transforms
T = [ 1 0 0; 0 -1 0; 0 0 1];           % This mirrors top half
tform_mirror = maketform('affine',T);
```

```
forward_fcn = @(xy,tdata)[xy(:,1),(Radius * (pi/2-
acos(((xy(:,2)-C)/Radius))))]; %Inverse Unwrapping
inverse_fcn = @(xy,tdata)[xy(:,1), ((Radius)*cos(pi/2-
((xy(:,2)+C)/Radius)))];     %Forward unwrapping
tform_unwrapp = maketform('custom',
2,2,forward_fcn,inverse_fcn,[]);                        %Combine
forward and Reverse to Create Uwnrapping Transform

% Apply transforms to Top and Bottom Half of Images seperatly
TopHalf = imtransform(TopHalf,tform_mirror);    % Mirror Top
half
TopHalf = imtransform(TopHalf,tform_unwrapp);   % Unwrapp Top
Half
TopHalf = imtransform(TopHalf,tform_mirror);    % Mirror Top
half Again
BottomHalf = imtransform(BottomHalf,tform_unwrapp); %Unwrapp
Bottom Half

%Recombine Top half and Bottom Half to produce Full Unwrapped
Image
if iseven == 1;                          %Original Image had even
# of rows
    temp= vertcat(TopHalf,BottomHalf);
else if iseven == 0;                     %Original Image had odd
# of rows
    TopHalf = TopHalf(1:(end-1),:);
    temp= vertcat(TopHalf,BottomHalf);
end
end

%Write Unwrapped Image to new Lossless jpg File
temp3 = cast(temp,'uint8');
temp4 = cast(temp,'uint8');
filename3 = ['Unwrapped Gradient ' filename ];
filename4 = ['LL Unwrapped Gradient ' filename ];
imwrite(temp3,filename3);
imwrite(temp4,filename4,'Mode','lossless');


%Write data to file
Edge1 = 'GRADIENT';
header = { 'Filename' ,'Edge Detection' };%Scale', 'Spacing',
'Radius_in';};
records = { filename, Edge1 };% ds.Scale  ds.Spacing
ds.Radius_in ];
ds6 = dataset({records,header{:}});
ds6.Number = 7;
```

```
ds6.Scale = ds.Scale;
ds6.Spacing = ds.Spacing;
ds6.Radius_in = ds.Radius_in;
ds6.Radius_row = Radius;
ds6.Size_Wrapped = size(Image);
ds6.Is_Even = iseven;
ds6.Edge = 'Gradient';
ds6.Top_Bottom_Canny = [Top Bottom];
ds6.Size_Cropped = [(Bottom- Top+1) size(Image,2)];
ds6.Size_Unwrapped = size(temp);
DATA{6} = ds6;
SaveDataAs = ['Dataset for Image ' filename ];
save(SaveDataAs,'DATA')
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%  This code aids the user in locating horizontal gridlines in a
%  user
%  defined image file. There ae three user inputs that MUST be
%%   defined in
%  the code.
%
% The user needs to enter the image name by editing the
filename: i.e.
%
%     filename = '0025'; Modify to: filename = 'Your Image File
Name Here';
%       Do not include '.jpg' in filename
%
% The user also needs to enter original girdline spacing in
inches: i.e.
%
%     spacing = 0.25;  Modify to:  spacing = Your Gridline
Spacing Here;
%
% The user also needs to define the pixel scale in inches/pixel
%
%     scale = 0.00110132; Modify to: scale = Your pixel scale
Here;
%
% The code will display the selected image and wait for the user
to double
% click the pointer on the image in order to select a column
that only
% intersects horizontal gridlines. The program will then display
a bar
% chart of the selected column. The user should then stretch or
move the
% horizontal line superimposed on the bar graph to definea
search area and
% cutoff value. When the user is satisfied with the location of
the line
% the user needs to double click on the line to proceed.
%
% The code will continue iterations until the user enters N in
the dialogue
% box, when prompted.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
```

```
clear all
counter = 0;

filename1 = uigetfile('*.mat');
load(filename1,'-mat');
ds = DATA{1};
filename2 =uigetfile('*.jpg'); %User input File Name


scale = ds.Scale(1);
spacing = ds.Spacing(1);
filename3 = sprintf('%s',ds.Filename{1});
[VV filename Vv] = fileparts(filename3)
% Read in user defined image file. Convert file to Grayscale if
neccesary
Image = imread(filename2);
if size(Image,3) == 3;
Image = rgb2gray(Image);
end

% Pre-Seed some variables
cutoff = 115;
Search1 =  160;
Search2 = 2410;
i = 1;



%This section prompts user to select column in order to find
grilines
%User should double click mouse if nothing is happening



imshow(Image)


%Displays current position of cursor over image, waits for
Double Click to
%get position of cursor
hp = impixelinfo;
set(hp,'Position',[150 150 300 20]);
H = impoint;
Point = wait(H);

%Prompts the user to confirm column choice from Double Click.
```

```
%Enter 'N' to end Analysis
prompt = {'Enter Row for analysis:', ' Do you want to continue
Analysis: (Y/N)'};
dlg_title = 'Column Number';
num_lines = 1;
P = num2str(round(Point(2)));
def = {P,'Y'};
answer = inputdlg(prompt,dlg_title,num_lines,def);
row = str2num(answer{1});
choice = answer{2};

%Displays Bar graph. Allows user to stretch and move horizontal
line to
%define search area for gridlines in image.
%Waits for user to DOUBLE CLICK line to set user choice and move
on.

Column = Image(row,:);
Column = cast(Column, 'double');
hparent = bar(Column,'DisplayName','I');figure(gcf)
h = imline(hparent,[Search1 cutoff; Search2 cutoff]);
setPositionConstraintFcn(h,@(pos) [pos(:,1)
repmat(mean(pos(:,2)),2,1)])
accepted_pos = wait(h);
pause(1)

Search1 = round(accepted_pos(1,1));
Search2 = round(accepted_pos(2,1));
cutoff = round(accepted_pos(1,2));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The following section Calculates the gridline location and %
Error
%Saves data as cell array FinalData. Each cell array has format:
%
%   Row 1: Column 1;                Selected Column for Analysis
%   Row 1: Column(2 through End);   Top location of each
Gridline
%   Row 2: Column 1;                Cutoff Value
%   Row 2: Column(2 through End);   Bottom location of each
Gridline
%   Row 3: Column 1;                Begining point for gridline
search
%   Row 3: Column (3 through end)   Midpoint of each gridline.
%   Row 4: Column 1                 Ending Point for gridline
search
```

```
%   Row 4: Column (3 through end)   Distance between each
gridline
%   Row 5: Column (3 through end)   Error in gridline spacing
%   Row 6: Column (3 through end)   Midpoint between each
gridline
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%

valve = 0;
l = 1;
r = 1;

%Search for gridlines in user defined search area, from bar
graph
for y = Search1:Search2
   if Column(y)>= cutoff && valve == 0 && Column(y+1) < cutoff
      Left(l) = y+1;
      l=l+1;
      valve = 1;

       end
   if Column(y) >= cutoff && valve >= 1  && Column(y-1) < cutoff
      Right(r)= y-1;
      r= r+1;
      valve = 0;
   end

end

MidpointLines = Left+ round((Right- Left+1)/2);
Delta = diff(MidpointLines);
DD = round(Delta/2);
MidRow = bsxfun(@plus,MidpointLines(:,1:end-1),DD);
First = Image(:,MidRow(1));
clear MidpointLines MidptBetweenGridlines Delta Data Left Right
Column column Left Right
for j = 1:size(MidRow,2)
Bigcolumn(j,:) = Image(:,MidRow(j));
end
ARRRGH = min(Bigcolumn);
Column = min(Bigcolumn);
%column = First;
Column = cast(Column, 'double');
hparent = bar(Column,'DisplayName','I');figure(gcf)
h = imline(hparent,[Search1 cutoff; Search2 cutoff]);
```

```
setPositionConstraintFcn(h,@(pos) [pos(:,1)
repmat(mean(pos(:,2)),2,1)])
accepted_pos = wait(h);
pause(1)
Search1 = round(accepted_pos(1,1));
Search2 = round(accepted_pos(2,1));
cutoff = round(accepted_pos(1,2));
valve = 0;
l = 1;
r = 1;

for j = 1:size(MidRow,2)
    column = MidRow(j);
    Column = Image(:,MidRow(j));
%Search for gridlines in user defined search area, from bar
graph
for y = Search1:Search2
   if Column(y)>= cutoff && valve == 0 && Column(y+1) < cutoff
      Left(l) = y+1;
      l=l+1;
      valve = 1;

       end
   if Column(y) >= cutoff && valve >= 1  && Column(y-1) < cutoff
      Right(r)= y-1;
      r= r+1;
      valve = 0;
   end

end

% Record data and calculations to variable FinalData and
Data(1,:) = [ column  Right];
Data(2,:) =[ cutoff Left ];

MidpointLines = Left+ round((Right- Left+1)/2);
Data(3,:) = [Search1 MidpointLines];

Delta = diff(MidpointLines);
Data(4,:) = [ Search2 0 Delta];

Error = ((Delta*scale - spacing)/spacing)*100;
Data(5,:) = [ 0 0 Error];

DD = round(Delta/2);
MidptBetweenGridlines = bsxfun(@plus,MidpointLines(:,1:end-
1),DD);
```

```
Data(6,:) = [ 0 0 MidptBetweenGridlines];
FinalData{i} = Data;

clear MidpointLines MidptBetweenGridlines Delta Data Left Right
Column column Left Right
valve = 0;
l = 1;
r = 1;
i = i+1;
v = 1;
end
% Plot marker on image to indicate the last column(s) selected
% for analysis
imshow(Image)
hold on
 for k = 1:size(FinalData,2)
     cn = FinalData{k}(1,1);
     text(cn,round(3/5*size(Image,1)),
sprintf('%.3d',v),'BackgroundColor',[.9 .0 .0]);
     v = v+1;
end
hold off

ShowImage = 1;


  % Ends gridline identification and calculations

% Plot original image, superimpose all of the % error calcs on
image at
% midpoint between gridlines
imshow(Image)
hold on
 for k = 1:size(FinalData,2)  %Plot Percent Error Gridline
Spacing
     counter = 1;
     cn = FinalData{k}(1,1);
     row = FinalData{k}(6,3:end);
     Error = FinalData{k}(5,3:end);

    for v = 1:size(row,2)
     if row(v) > 0
         text(cn-10,row(v),  sprintf('%6.1f',
Error(v)),'BackgroundColor',[.9 0 0]);
         counter = counter+1;
     else
```

```
    end
    end

 end
hold off




%Save figure with Error data superimposed over image.
f = getframe(gca);
im = frame2im(f);


SaveFigureAs = ['Error Data for ' filename5, ' ', filename
'.jpg'];
imwrite(im,SaveFigureAs);
%Save the cell array data to .m file. Append 'Error Data for' in
front of
%original filename.

DATA{2} = FinalData;
SaveDataAs = ['Dataset for Original Image ' filename ];
save(SaveDataAs,'DATA')




%% This is the bar Chart Plotting Section

j = 1;

for k =1:size(FinalData,2)
    SizeData(k) = size(FinalData{k}(5,3:end),2);
end
M = max(SizeData);
ErrorData = zeros(k,M);
YData = zeros(k,M);
for k =1:size(FinalData,2)
    S = size(FinalData{k}(5,3:end),2);
    ErrorData(j,1:S) = FinalData{k}(5,3:end);
    YData(j,1:S) = FinalData{k}(6,3:end);
    XData(j,:) = FinalData{k}(1,1);
    j = j+1;
end
```

```
%XData = XData';

f = num2str(XData);

ErrorData = ErrorData';
YData = round(mode(YData))';

h =bar3(YData,ErrorData)

colorbar
for k = 1:length(h)
    zdata = get(h(k),'ZData');
    set(h(k),'CData',zdata,...
             'FaceColor','interp');
end

title('Percent Error in Gridline Spacing Across the
Image','FontName','Calibri', 'FontSize',24)
set(gca,'Zdir','reverse')
ylabel('Row Number in Image','FontName','Calibri',
'FontSize',12);
set(gca,'YTickLabel',YData); %'rotation',-40
xlabel('Column Number Across Image','FontName','Calibri',
'FontSize',12,'rotation',-60)
%set(gca,'ActivePositionProperty','Outerposition');%set(get(gca,
'xlabel'),'rotation',20);
set(gca,'XTickLabel',XData);
%set(gca,'Xticklabel',{'2', '78'});
zlabel('Percent Error in Gridline Spacing','FontName','Calibri',
'FontSize',16)
view([-93.5 6]);
SaveBarGraphAs = ['Bar Graph of Error Data for ' filename
'.jpg'];
saveas(gcf,SaveBarGraphAs);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
% This code allows the user to access a data base of information
% from the error analysis
% The user has the option to plot three different figures and
get the
% a plonyomial equation to fit the data
%
% load Error Data from cell array
% Select Data Set fo Errors;  Dataset = 3, 5, or 7
% 1-Scale Data
% 2- VisualEdge Detection Unwrapping
% 3-Visual Error
% 4- Canny Edge Detection Unwrapping
% 5-Canny Error
% 6-Gradient Edge Detection Unwrapping
% 7-Gradient Error

Dataset = 7;
FinalData = DATA{Dataset};
% Get Error Data and location in image
for i = 1:size(FinalData,2)
    A = FinalData{i}(5,3:end);
   Sb(i) = size(A,2);
end
MaxSize = max(Sb);

Error_Z = zeros(size(FinalData,2),MaxSize);
Row_Y = zeros(size(FinalData,2),MaxSize);
Column_X = zeros(size(FinalData,2),MaxSize);

for i = 1:size(FinalData,2)
    S = size(FinalData{i}(5,3:end),2);
 Error_Z (i,1:S)=FinalData{i}(5,3:end);
 Row_Y(i,1:S) =FinalData{i}(6,3:end);
  Column_X(i,1:S) =FinalData{i}(1,1);
end

% Get Statistical Data
M = mean2(Error_Z);
ModifiedVariance = sum(sum( Error_Z.^2))/(nnz(Error_Z)-1);
ClassicalVAR = sum(sum( Error_Z.^2-M))/(nnz(Error_Z)-1);
ClassicalSTD = ClassicalVAR^.5;
Variance1 =var(reshape(Error_Z,1,numel(Error_Z)));
StandardD = Variance1^.5;
RangeMin = min(min(Error_Z));
RangeMax = max(max(Error_Z));
```

```
ds = DATA{Dataset-1};
name = ds.EdgeDetection{1};
EdgeMethod = sprintf('%s',name);

%%
% Scatter Plot
Center = mean2(Row_Y);
Y2 = Row_Y-Center;
Y = reshape(Y2,1,numel(Y2));
Z1 = Error_Z;
Z = reshape(Z1,1,numel(Z1));

% Get polynomiaal and R2
p1 = polyfit(Y,Z,2);
Zfit2 = polyval(p1,Y);
Zresid = Z-Zfit2;
ssresid = sum(Zresid.^2);
sstotal= (length(Z)-1)*var(Z);
rsq=1-ssresid/sstotal;
% Plot Scatter
scatter(Y,Z,'b')
hold on
title(['% Error in Gridline Spacing Across Unwrapped '
EdgeMethod ' Image'],'FontName','Calibri', 'FontSize',12)
set(gca,'Zdir','reverse')
ylabel('% Error','FontName','Calibri', 'FontSize',12);
xlabel('Distance From Center of Core in
Image','FontName','Calibri', 'FontSize',12)

tfit = (min(Y):0.1:max(Y));
yfit = polyval(p1,tfit);
plot(tfit,yfit,'r','LineWidth',2)
a = num2str(p1(1));
b = num2str(p1(2));
c = num2str(p1(3));
str1 = [ a 'x^2 +' b  'x'  '+' c ];
R = num2str(rsq);
str2 = [ 'R2 = ' R];
text(-600,-8,str1)
text(-600,-10,str2)
hold off
%%
%% This is the bar Chart Plotting Section

XData = Column_X(:,1)';
```

```
f = num2str(XData);

ErrorData = Error_Z';
YData = round(mode(Row_Y))';

h =bar3(YData,ErrorData)

colorbar
for k = 1:length(h)
    zdata = get(h(k),'ZData');
    set(h(k),'CData',zdata,...
             'FaceColor','interp');
end

title('Percent Error in Gridline Spacing Across the
Image','FontName','Calibri', 'FontSize',24)
set(gca,'Zdir','reverse')
ylabel('Row Number in Image','FontName','Calibri',
'FontSize',12);
set(gca,'YTickLabel',YData); %'rotation',-40
xlabel('Column Number Across Image','FontName','Calibri',
'FontSize',12,'rotation',-60)
%set(gca,'ActivePositionProperty','Outerposition');%set(get(gca,
'xlabel'),'rotation',20);
set(gca,'XTickLabel',XData);
%set(gca,'Xticklabel',{'2', '78'});
zlabel('Percent Error in Gridline Spacing','FontName','Calibri',
'FontSize',16)
view([-93.5 6]);
SaveBarGraphAs = ['Bar Graph of Error Data for ' filename
'.jpg'];
saveas(gcf,SaveBarGraphAs);




%%
%%
%I = Image;
%Irgb = cat(3,I,I,I);
%imagesc(Irgb)
%colormap(Gray)
%hold on
%[c,h] =contour(Column_X,Row_Y,Error_Z,10);
[c,h] = contourf(Column_X,Row_Y,Error_Z,10);
colormap(jet)
%title('Contours of Error Values')
```

```
%colormap autumn
clabel(c,h)
%%
Irgb = cat(3,I,I,I);

imagesc(Irgb)
F = getframe;
im0= F.cdata;
clf
contourf(Column_X,Row_Y,Error_Z,10);
F = getframe;
im1 = F.cadata;
clf
image(im0)
hold on
h = image(im1);
set(h,'AlphaData',0.5)
hold off
%%
Irgb = cat(3,I,I,I);
image(Irgb)
hold all
contourf(Column_X,Row_Y,Error_Z,10);

colormap('default');
xlim([1, 4752]); ylim([1, 3162]);
xlabel('x (pixels)', 'FontName', 'Arial', 'FontSize', 14);
ylabel('y (pixels)', 'FontName', 'Arial', 'FontSize', 14);
set(gca,'XAxisLocation', 'top', 'YDir', 'reverse', 'FontName',
'Palatino Linotype', 'FontSize', 13, 'DataAspectRatio', [10000
10000 1]);
colorbar
set(gca,'clim',[min(displacement_x(:)) max(displacement_x(:))]);
```