2015

Processing Pre-Existing Connect-The-Dots Puzzles For Educational Repurposing Applications

Shelby Elizabeth Kilgore

North Carolina Agricultural and Technical State University

Processing Pre-Existing Connect-the-Dots Puzzles for Educational Repurposing Applications

Shelby Elizabeth Kilgore

North Carolina A&T State University

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Department: Electrical and Computer Engineering

Major: Electrical Engineering

Major Professor: Dr. Corey A. Graves

Greensboro, North Carolina

2015

The Graduate School
North Carolina Agricultural and Technical State University

This is to certify that the Master Thesis of

Shelby Elizabeth Kilgore

has met the thesis requirements of
North Carolina Agricultural and Technical State University

Greensboro, North Carolina
2015

Approved by:

_____	_____
Dr. Corey A. Graves Major Professor	Dr. Christopher Doss Committee Member
_____	_____
Dr. John Kelly Committee Member and Department Chairperson	Dr. Sanjiv Sarin Dean, The Graduate School

Biographical Sketch

Shelby Elizabeth Kilgore was born May 5, 1991 in Syosset, New York. She is the third and final child of her parents, and she has two older sisters. She received her high school diploma from Westbury High School of Westbury, New York in 2009. She received her Bachelor of Science in Computer Engineering at North Carolina Agricultural and Technical State University in 2013. In Fall of 2013 she became a candidate for a Master's of Science in Electrical Engineering with a concentration in Computer Engineering.

Dedication

This thesis is dedicated to my loving family; my father, my mother, and my two older sisters. My parents have instilled in me the values of having faith in God, hard work, strength, perseverance and determination. I truly believe that without their support I would not have been able to achieve this great accomplishment. I love and appreciate my family for how they have molded me and how they continue to support me as an adult.

Acknowledgments

I first would like to honor my Lord and Savior Jesus Christ who is the all-knowing director my life. He has been my strength and stability throughout my graduate school matriculation. The Bible says in the eighth chapter of the book of Romans, "And we know that all things work together for good to them that love God, to them who are the called according to His purpose". From this scripture comes my motivation to stay focused and encouraged during times of failure and obstacles because I know that even the negative aspects of life will yield a positive result. God is the center of my life and without Him nothing would be possible.

I would also like to extend my sincerest gratitude towards my graduate advisor, Dr. Corey A. Graves and my committee members, Dr. John C. Kelly and Dr. Christopher C. Doss for their support and assistance.

I would like to thank my parents, Rufus and Debra Kilgore for their love, support and prayers throughout my graduate matriculation. They have taught me morals, values and standards that have birthed a drive in me that will not allow me to fall short of any of my goals.

I must thank my two older sisters, Heather and Ashley, for all of their support and prayers. They have been perfect role models, by obtaining degrees from Morgan State University in Psychology and Biology, respectively. They continued to demonstrate the value of education by continuing on to receive advanced degrees in Psychology and Veterinary Technology. They conduct their lives in ways that I have always been able to look to as guidance for my own.

I would also like to thank my friends, roommate, and sorority sisters for their support. They have always been there to keep me grounded, motivated, and encouraged. They also provided a means for me to relax and relieve stress at times when I did not realize I needed the break. I would

like to thank my roommate, Ciarra Cornelius, additionally, for being my occasional alarm clock after long nights of hard work.

Finally, to my alma mater, North Carolina Agricultural and Technical State University, I appreciate you providing the place for me to learn and grow as a young professional. Here is where I have met some lifelong friends who have helped develop me intellectually as well as emotionally. I did not understand the loving nature of A&T when I entered as a freshmen. However, now that I am leaving with a Master's degree, I fully understand the value of this institution and the proud slogan, AGGIE PRIDE!

Table of Contents

List of Figures

Abstract

Connect-the-Dots puzzles are puzzles which contain labeled dots in a sequence. These puzzles are mostly designed as a way for children to hone in on their counting skills, while having fun. These same puzzles, which are available in abundance online and with modification, can be used to aid students in other areas of education such as spelling. Research shows that the addition of visual imagery provides a significant impact in spelling performance.

The objective of this research is to develop an algorithm for processing Connect-the-Dots puzzles to assist in the replacement of the original numbers in the puzzle with characters that will help to facilitate an alternative educational purpose. In particular, the use of Optical Character Recognition (OCR) and image processing algorithms to process pre-existing Connect-the-Dots puzzles is explored. An algorithm was developed to locate and identify the numbers in the puzzles. The system is comprised of five components, namely, an Image Preprocessing component, a Dot Locator component, a Number Locator component, a Number Recognition component, and a Post-Processing component.

To test the accuracy of the algorithm an experiment was conducted using 20 hand selected puzzles from an online source. The accuracy of the algorithm was evaluated, component by component, as well as overall, by visually capturing the make-up of the puzzles and comparing them to the results generated by the algorithm. Results show that the algorithm performed at an overall accuracy rate of 66%. However, the Dot Locator component performed at a rate of 100%, the Number Locator at a rate of 86%, and the Number Recognition at a rate of 76%. This research will aid in the development of an application that may provide educational benefits to children who are exposed to using technology for learning, at a young age.

CHAPTER 1

Introduction

"Reading ignites a hunger for knowledge and facilitates education" [1], which is an essential part of human development. Students who are solid readers perform better in school, have a healthy self-image, and become lifelong learners, which add to their viability in a competitive world [1]. Becoming a solid reader is dependent upon spelling ability, which is an integral part of literacy.

In recent years, there appears to have been significant effort focused on improving education in a realm where technology is becoming more embedded in everyday life. It seems as though an emphasis has been put on sustaining the interests of students who are constantly stimulated by technology in a non-educational manner. In a study concerning tablet PCs in education, it was shown that tablet PCs increase student focus and attentiveness [2]. The use of tablets allows students to take a more independent and active role in their education, inside and outside of the classroom. Tablets have the potential to facilitate learning in important areas of education, such as spelling and reading.

Studies have shown that visual imagery is a significant factor in spelling performance and retention, especially when coupled with appropriate imagery [3]. The importance of imagery in spelling performance, as well as the growing interest in using technology for education, is the motivating factor behind the proposed tablet-based Connect-the-Dots puzzle application.

Connect-the-Dots puzzles, also known as Dot-To-Dot puzzles, are puzzles that contain a sequence of labeled dots. These dots are marked with numbers or letters that create a picture when connected in successive order. These puzzles are ordinarily used to aid in counting or learning the alphabet; however, the proposed application seeks to exploit the well-known puzzle by using

letters, which comprise a word, to label the dots. The word, when properly spelled, will bridge the dots to complete an image that embodies the word that is spelled. Prior to now, no research has been done on the use of these puzzles for reading and spelling education.

In order for this application to function, there is a need for customized Connect-the-Dots puzzles. Traditionally, designing and constructing brand new electronic versions of puzzles require artistic skill and time. However, now due to technology there are large amount of fully developed puzzles that are available for free online. These puzzles can be downloaded and saved as image files and recycled in the development of the proposed spelling application. While these puzzles may be readily available for use, they require some modification to fit the purpose of the system.

In this Thesis, we present an algorithm that will identify numbers in Connect-the-Dots puzzles for the purpose of substituting the existing numbers in a puzzle with any other characters. The proposed algorithm makes use of an existing open source Optical Character Recognition engine, Tesseract [4], along with the Image Processing Toolbox found in MATLAB [5] in order to accurately detect numbers in grayscale GIF images of Connect-the-Dots puzzles.

CHAPTER 2

Digital Image Manipulation Techniques

A digital image is a representation of a two-dimensional image as a finite set of digital values, called pixels [6]. Digital images are formed by a process called digitization which samples gray values at a discrete set of points and stores them as a matrix [7]. This implies that a digital image is represented in a coordinate plane and can be referred to as a function of x and y [6]. The amplitude of the function at any pair of coordinates is the gray level at that point [6]. A pixel occupies a small rectangular region on the screen and displays one color at a time [8]. A pixel not only has a particular value but also a location which corresponds to the previously mentioned x and y coordinates.

Digital Images can be saved in many file formats, such as: TIFF, PNG, GIF, JPG, RAW, BMP, PSD, and PSP. The vast majority of image files that can be found on the Internet according to [8] are of the formats: BMP, XBM, JPEG, GIF, and PNG. This Thesis deals only with GIF, PNG, and JPEG formats.

GIF, PNG, AND JPEG are all considered bitmap image formats meaning that they are represented as 2-dimensional arrays [8]. One drawback with bitmap images is the amount of data required to hold them. It is mentioned in [8], that the size of an image in bytes (not counting overhead) is represented by this equation:

$$\frac{w \times h \times b + 7}{8}$$

In this case w is the width of the image in pixels, h is the height of the image in pixels, and b is the bits per pixel.

Digital Images have been manipulated since as early as the 1920s [6]. Digital Image Manipulation can be broken into three categories: Image Processing, Image Analysis, and Image

Understanding [7]. Image Processing takes in an image and produces another version of that image through some form of technique [7]. Such manipulations of image processing include noise removal, image sharpening, image type conversion, and others. Image Analysis takes in an image and outputs data that represents information about the image [7]. Image Analysis processes include segmentation, object recognition, localization, and measuring object properties. Yet a higher level of image manipulation is Image Understanding, which involves inputting an image and outputting a high-level description of the image, with inferences that mirror human comprehension. Such processes include optical character recognition, scene understanding, autonomous navigation, and facial recognition. This thesis utilizes all three of these types of manipulations.

2.1 Digital Image Processing

Digital Image Processing is defined as the acquisition and processing of visual information by a computer [9]. Image Processing is a low-level process under Digital Image Manipulation which encompasses applications such as noise reduction, contrast enhancement, image sharpening, image resizing, and image type conversion [6]. The image processing techniques that this thesis utilizes include image resizing, image type conversion, and graphics insertion.

2.1.1 Image resizing. MATLAB Image Processing Toolbox uses a function named imresize() to scale an image. This function will return an image that is the size of the specified 'scale' value times the original size of the image [5]. This function both enlarges and shrinks the image; if the 'scale' value is greater than 0 and less than 1 then the image will shrink, if the 'scale' value is greater than 1 then the image will be enlarged [5].

2.1.2 Image type conversion. MATLAB also has a function which converts an indexed image, grayscale image, or truecolor image into a binary image. An indexed image is one that consists of a data matrix, and a colormap matrix [5]. The rows in the colormap specify the red,

green, and blue components of each pixel color [5]. The x value in the image matrix corresponds to the row index of the colormap, which determines the RGB value for that pixel location. Grayscale images are stored as a data matrix with each element of the matrix corresponding to one image pixel [5]. These images do not need an associated colormap because the gray levels are represented in values ranging from 0 to 255 [5]. A truecolor image is one that is stored in a data array that defines the red, green, and blue color components for each pixel in the image [5]. In these types of images, each color (red, green, and blue) are stored as 8 bits each, providing about 16 million color options [5]. A binary image is represented by a Boolean matrix only consisting of 0s and 1s which represent black and white pixels [5]. The im2bw() function converts the input image to a binary image using the threshold level.

The threshold level can be calculate using the function graythresh(). This function computes the global threshold that can be used to convert the image to a binary image [5]. The output of this function is an intensity value that fits in the range of 0 to 1 [5].

2.1.3 Graphics Insertion. MATLAB's toolbox has the functionality to fuse graphics into an image [5]. The insertShape() function takes the image, the shape, position and size of the shape, as inputs and outputs a truecolor image with the shape inserted [5]. This action is completed by overwriting pixel values [5].

2.2 Digital Image Analysis

Digital Image Analysis is the extraction of meaningful information from digital images [7]. Image analysis is a mid-level process under the Digital Image Manipulation umbrella, which involves applications such as segmentation, classification, localization [10], object analysis, and region/image properties [5]. The categories under image analysis that will be discussed in this

paper are localization (more specifically, text localization), object analysis, and measuring region properties.

2.2.1 Text Localization. Text Localization is defined as the process of determining the location of text in the image and generating bounding boxes around the text [11]. The authors in [12] state that the difficulty of text localization can be attributed to text-like background objects, which lead to false alarms in text detection. Methods for text localization can be broken into two categories: region-based and component-based [13].

Region-Based. Region-based methods are designed around the concept that text regions have distinct characteristics from non-text regions such as distinctive gradient distribution, texture and structure [13]. According to [13], region-based methods are sensitive to the text orientation and cluster number, meaning that they can only localize texts containing many characters in horizontal alignment. These methods rely heavily on the contrast of the color or gray scale of a text region to those of the background [13].

Component-Based. Component based methods of text localization are based on observations that texts can be seen as sets of separate connected components, each of which has a distinct intensity, color distribution and enclosed contour [13]. These methods generally contain three stages: Connected Component (CC) extraction to segment CCs from images, CC analysis to determine whether or not they are text components by heuristic rules or classifiers, and post-processing to group text components into text region [13]. According to [13], for Component-based methods, text components are hard to segment accurately without prior information of text position and scale. Designing fast and reliable CC analysis method is also difficult when there are too many text-like components in images [13].

2.2.2 Object analysis. MATLAB uses the Hough Transform to find circular objects in an image. The Hough Transform is a transform designed to identify lines and curves within an image [5]. A circle with radius R and center (a, b) can be described with the parametric equations:

$$x = a + R\cos\theta$$

$$y = b + R\ \sin\theta$$

The angle sweeps through the full 360 degree range and the points (x, y) trace the perimeter of a circle. The algorithm seeks to find (a, b, R) to describe each circle with a and b being the coordinates of the center of the circle and R being the radius. The Hough Transform can be a very computationally expensive algorithm when using it for circle finding. However, having an exact radius value or range of values will reduce the computation time and memory for storage. For each edge point, a circle (perimeter circle) is drawn with that point as the origin and R as the radius. The algorithm makes use of an array that contains the coordinates of the circles and the radii. The values in this array (accumulator array) are used to determine the true center point, which will be common to all the perimeter circles as shown in Figure 1.

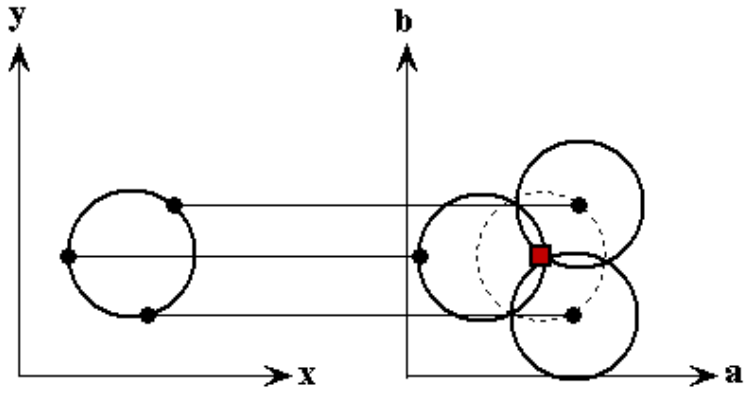


Figure 1. Circular Hough Transform.

2.2.3 Measuring region properties. Measuring region properties is a subset of image analysis. The set of functions that belong to this category serve the purpose of retrieving information about the objects in an image.

Connected components. In MATLAB there is a function that returns the connected components (CCs) found in a binary image, or specified region of an image [5]. The connected components are determined using a default connectivity of 8. Two pixels are said to be an 8-neighbor of each other if they share an edge or a vertex. A set of black pixels is an 8-connected component if each pixel in the set is an 8-neighbor to another black pixel [14]. Figure 2 below displays an example of an 8-connected component as represented in a binary image with its pixel values given.

MATLAB's algorithm for finding the connected components is [5]:

1. Search for the next unlabeled pixel, p.
2. Use a flood-fill algorithm to label all the pixels in the connected component
3. Repeat steps 1 and 2 until all the pixels are labelled.

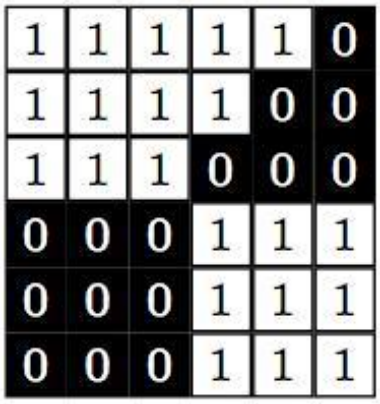


Figure 2. Binary representation of 8-connected component.

Bounding box. After a connected component is found, MATLAB also provides functions that can be used to gain information about the objects found, called regionprops(). Within this

function one can obtain the bounding box of the component. The bounding box is the smallest rectangle containing the region [5]. An example of a bounding box can be found in Figure 3 below where the green box represents the bounding box.

Figure 3. Example of a bounding box.

2.3 Digital Image Understanding

Image Understanding, otherwise considered computer vision, is defined as the construction of explicit meaningful descriptions of the structure and the properties of the 3-dimensional world from 2-dimensional images [15]. Optical Character recognition falls under the umbrella of Computer Vision otherwise known as Digital Image Understanding.

2.3.1 Optical character recognition. Optical Character Recognition (OCR) is the process used to convert scanned or printed images into images that are machine-encoded and editable [16]. Data entry, text entry, process automation, aid for the blind, automatic number-plate readers, automatic cartography, form readers, signature verification and identification are just a few of the many applications that benefit from OCR systems [16]. OCR is performed on images off-line after the writing or printing has already been completed, as opposed to on-line recognition where the computer recognizes the characters as they are drawn. Both handwritten as well as computer printed text can be recognized by OCR. OCR systems date back to the 1950s where they were implemented as very expensive hardware machines and now have evolved to commercially available software packages. A lot of potential seems to lie within the exploitation of existing

methods, by mixing methodologies and making more use of context [17]. Methods of OCR have been exploited in various research projects such as: License plate recognition, and breaking CAPTCHA images. There are many OCR tools available, however there are few that are open source and free [18]. One of those open source free tools is called Tesseract [4].

CAPTCHA (Completely Automated Public Turing Test to Tell Computers and Humans Apart) manifests as the well-known images of contorted words that pop up whenever one is trying to complete an important transaction online. This standard is used (as apparent in its definition) to distinguish between humans and undesirable malicious programs. The authors of [19] state that the most commonly used CAPTCHAs are text-based and rely on the distortion of the letters and visual effects added to the background of the image. They designed a system to attack specifically the Yahoo! CAPTCHA which is made to be resistant to segmentation methods. The algorithm described was completed in three stages: remove the noise pixels and fix the broken characters, identify and then erase some characters to divide the text, and segment the remaining large chunks which contain two or more characters. Threshold values are manually determined to "sharpen" the image by removing the scattered pixels. Sharpening the image allows for the 'edge' of the characters to be defined and detected. The segmentation and extraction attack that was designed achieved success at a rate of 78%, while the recognition rate with the OCR was 54.7% [19].

2.3.2 Tesseract. Tesseract is a free open source OCR tool written in C++, making it platform independent [18]. It is flexible in that it can be used in other applications in the form of a Dynamic Link Library (DLL). Tesseract was developed at HP in between 1984 and 1994 and was improved with greater accuracy in 1995. In late 2005, HP released Tesseract for open source. Tesseract is more focused towards providing less rejection than accuracy [4]. This means that

Tesseract is more likely produce false positive results in order ensure more true positive results and decrease the number of false negative results.

Tesseract's architecture operates in five stages. The first step is to take the input image and convert it into a binary image. This step is referred to as adaptive thresholding [18]. Next comes the Connected Component Analysis which extracts character outlines. This gives the flexibility of performing OCR on images with white text and black background. After this, Tesseract finds the lines and words by the use of blobs, definite spaces and fuzzy spaces [4]. Finally, the last two steps are a two pass recognition stage [18]. In the first pass, an attempt is made to recognize each word from the text. Each word passed satisfactory is passed to an adaptive classifier as training data. The adaptive classifier tries to recognize text in a more accurate manner. The final phase is used to resolve various issues and extract text from images. According to a study done by [18], Tesseract is most accurate on grayscale images as opposed to colored images.

Tesseract has been used in many research projects. One of those projects involves using the engine on image spam mail filters to help in determining whether an image includes spam words [20]. The focus of this research was to recognize when four specific words were contained in spam mail images. The researchers in [20] use Tesseract's language settings to limit the amount of CPU processing time it takes for Tesseract to run as well as to increase accuracy.

These researchers created an equation detector which they built into Tesseract. The algorithm uses the density of special symbols and does not utilize traditional classifiers which require manually created training data. This algorithm was tested using the Google Books database and performed at a precision rate of 74.3% [21], where precision was defined as:

$$PC = \frac{decrease\ in\ M_{add}}{decrease\ in\ M_{add} + \ increase\ in\ M_{drop}}$$

M_{add} is the number of characters that are added into the OCR texts in comparison to the ground truth texts, and M_{drop} is the number of characters that are dropped from the OCR texts compared to the ground truth text [21].

Tesseract is a free open source OCR engine that has been used for a variety of applications. OCR has been used to break CAPTCHA security as well as to filter out spam emails. Tesseract works performs more favorably on grayscale images and can be integrated into any application. These factors make Tesseract a suitable engine to conduct recognition on grayscale images of Connect-the-Dots puzzles as depicted in this research.

CHAPTER 3

Compressed Image File Formats

Compressed image file formats are used in order to store an image without taking up as much space and memory as the size of the image requires. Compression can be classified as "lossless" or "lossy". Lossless Compression is the case when the image that is reconstructed after the compression has occurred is identical to the original image [22]. In contrast, lossy compression produces an image that is not an exact replica of the original [22]. If the human perceptual system is the judge of the fidelity of the reconstructed image, then some amount of data loss is acceptable [22].

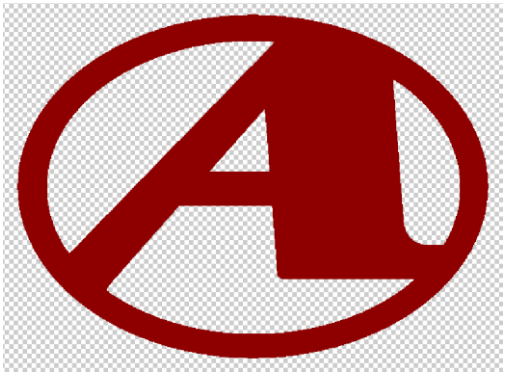
PNG (Portable Network Graphics) is a lossless image storage format [23]. PNG files are designed to encode low-resolution images that load quickly. PNG compressed files produce images with blurred or feathered edges as displayed in Figure 4(a) below [24].

GIF (Graphics Interchange Format) is the oldest and most widely supported Web-based graphic file format [25]. This compressed file type has lossless compression for images that only contain 256 colors or less [23]. If the image has more than 256 colors then several algorithms are used to approximate the colors in the image with the limited palette of 256 colors available [23]. GIF images are perfect for creating low resolution files with solid areas of color [24]. GIF images produce rougher pixelated, crisp-edged graphics as shown in Figure 4(b), compared to PNG images [24].

JPEG (Joint Photographic Experts Group) is a lossy algorithm that was developed for the storage and transmission of photographic images with many colors [25]. JPEGs can compress images while maintaining high image quality. JPEG are useful for compressing larger image files.

All of the compressed files mentioned in this chapter will be used in the development and testing of the system outlined in this Thesis.

(a)



(b)

Figure 4. (a) Sample image in PNG format, (b) Sample image in GIF format.

CHAPTER 4

Methodology

Tool for Repurposing of Online Puzzles for Education (TROPE) consists of five major components that contribute to the overall methodology of this research; the Image Preprocessor, the Dot Locator, the Number Locator, the Number Recognition and the Post-Processing components. The Image Preprocessor component prepares the uploaded image to be analyzed by the system. This component converts the image to a binary image and calculates the size of the image. The Dot Locator component discovers the dots that are in the puzzle. This component determines the pixel coordinates of each dot's center as well as the radius of each dot within a puzzle. The third component is the Number Locator Component. This component determines the approximate position of the numbers on a per-region basis. The pixel coordinates for each region that solely contains the number are output by this component. The third component is the Number Recognition component. This component analyzes the specified region(s) passed to it in order to identify each number. A first pass of the Number Recognition component is executed over the entire puzzle in order to obtain the preliminary number recognition results. The information obtained from this first pass is then used in to the Number Locator component to aid in the location of each number. The regions from the number locator are then passed in to the Number Recognition component for a second and final recognition. Finally, the Post-Processing component is executed in order to verify results as well correct any misclassified results. The five components and their connections to each other are presented in Figure 5 below.

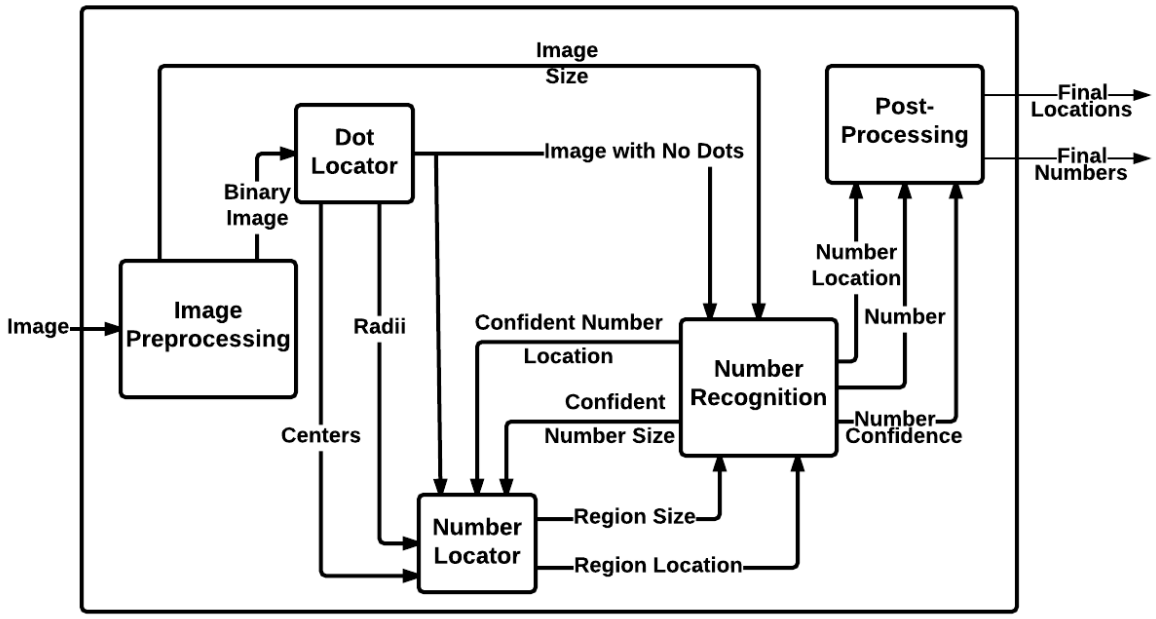


Figure 5. Block diagram of TROPE

4.1 Image Preprocessing Algorithm

The Image Preprocessing component takes, as input, the uploaded image file and outputs a binary image and the width and height (in pixels) of that image. First the component checks to see if the file uploaded is either a PNG, JPEG, or GIF image file. If the image is not in one of these formats then it is rejected. If the file is accepted then the uploaded image is taken and the component determines whether it is a grayscale image or an indexed image. The Image Preprocessing algorithm checks to see if the uploaded image has a colormap associated with it to determine whether it is a grayscale or indexed image. If the image is a grayscale image then it is converted to a binary image using the functions: graythresh() and im2bw(). If the image is an indexed image then it is converted to a grayscale image first using ind2gray() and then to a binary image. Once the image is in binary format, the width and height of the image are calculated using the size() function [5]. The binary image along with the image height and width are output from this component.

4.2 Dot Locator Algorithm

The Dot Locator component takes a Connect-the-Dots puzzle in the form of a binary image as input, and outputs the pixel coordinates of the centers of all the dots, as a matrix, and the radius size of the dots as a vector. Using a circle finding function in the MATLAB Image Processing Toolbox [5], TROPE conducts a two pass circle find to locate the dots. The first pass looks for circles that fit within a pre-defined relatively large radius range. Once the results for this pass are collected, the mode of the radii is taken to form a smaller radius range and used in the second pass. The second pass serves to increase accuracy by eliminating circles in the puzzle that may be misclassified as "dots" based on the size. If the circles found in the first pass are not similar in radius size to the mode of the results, then those circles are removed as possible dots after the second pass. Following the second pass, a false-positive elimination phase is implemented. True dots in Connect-the-Dots puzzles are filled black circles, containing only a minute number of white pixels. This phase scans the results of the second pass to eliminate circles found which have a significant amount of white pixels within them. If the dot locator component finds less than 10 dots in the puzzle then the image is resized to twice the size of the original image. The Dot Locator is then performed on the resized image again. This process is repeated recursively until the Dot Locater is able to find more than 10 dots in the puzzle.

Figure 6 shows the results of the Dot Locator component outlined in red. Once the final results are found, the parameters for the centers and radii of the dots are kept, however, all of the dots are removed from the image for the remainder of processing done on the puzzle, in order to reduce image clutter when locating and recognizing numbers.

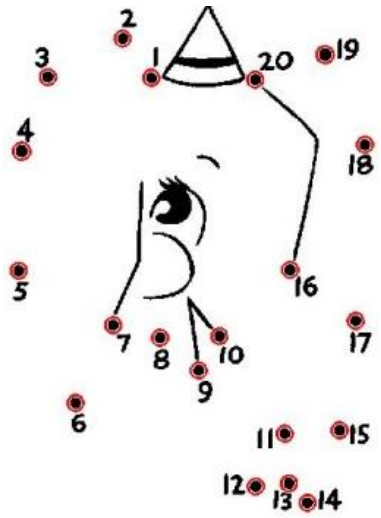


Figure 6. Visual results of the Dot Locator.

4.3 Number Locator Algorithm

The results of the Dot Locator algorithm, along with the results from the first pass of the Number Recognition component, are used in the Number Locator Algorithm to determine the general region in which the number lies. The Number Locator component can itself be broken into four component algorithms: (1) Determining the Initial Region, (2) Choosing a Component, (3) Finding the Top Left Corner, and (4) Validating the results. All of these components serve to analyze the region around each dot and determine which cluster of black pixels is the number associated with that dot. The Number Locator algorithm provides the second pass of the Number Recognition component with the top-left coordinates as well as the sizes (widths and heights) of the rectangles that will contain all of the numbers found.

4.3.1 Determining the initial region. Determining the initial region to begin searching for a number requires information about the dot corresponding to it. The dot's radius is used to create a square region around the dot whose sides are equal to 4 times the radius. Once this region is created, the bwconncomp() MATLAB function [5] from the Image Processing Toolbox, is used to find all connected components in the region. If no components are found in the initial region of

interest, then the region is expanded slightly (left, right, up, down) and searched again for connected components. This process continues, expanding the region in all directions incrementally, until a component is found within that region. Once any connected components are found, then the next step involves determining which set of components is most likely a fragment of the number (that is, if more than one set of components exists in the region).

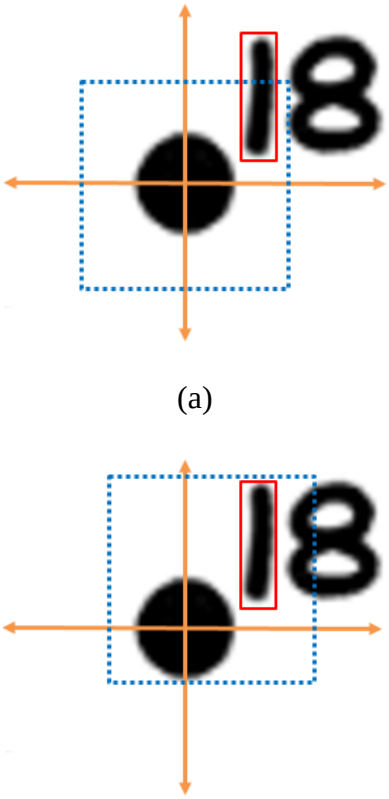
4.3.2 Choosing a component. The function (CorrectComp()) in the Number Locator process decides which of the sets of components that have been found in the region, is most likely to be a portion of the number. If there was only one element found then it is assumed that that element is the number. However, if there is more than one constituent represented within the region, then there are a few criteria that are used to select the correct component.

The first criterion that is explored involves the data gathered from the first pass of the Number Recognition component. If any number has been recognized with high confidence in the initial pass of the Number recognition, then information regarding the location of the numbers found is passed to the Number Locator component. These numbers' locations are traversed to determine if any of those numbers fits within the region of interest. If that is true, then we compare the connected components found to the number found to determine which component is in the same position as the number. Once that component is found, then it is accepted as the correct component.

The second criteria, providing that Tesseract's first pass did not recognize the number near this particular dot being analyzed, is to determine which component is closest to the center of the dot. In this case, the component that is nearest to the dot is returned as the correct component. The algorithm for choosing the correct component is shown in Appendix A.

4.3.3 Finding the top left corner. Once the targeted component is acknowledged, it is now the time to figure out what the dimensions of the section of interest will be. Since rectangles and rectangular regions in MATLAB are defined using the coordinate pair of the top-left corner as well as the width and the height of the rectangle, it is imperative that the top-left "corner" of the component is found. In MATLAB when connected components are returned, there is another function (regionprops()) which will allows for the calculation of a "bounding box" that contains the component found. This bounding box is an imaginary tightly fitting rectangle that surrounds the entire connected component and is utilized in this algorithm to find the top-left corner of the component.

To complete this task, an imaginary 2-dimensional coordinate plane whose origin is the center of the dot, is created. Using this imaginary coordinate plane TROPE determines within which quadrant the component of focus is found. Once that is established, the region is moved in a particular direction until the top-left corner of the number is completely contained in the region. For example, if the component is found in Quadrant II, then the algorithm will check to see if the pixel coordinates for the top of the region is equivalent to the coordinates of the top of the components bounding box, the same is done for the left side of the region and bounding box. If this equivalence is there then it is acknowledged that the entire component is not represented in the region. Thus the region is moved slightly up and left and checked again. Figure 7(a) and Figure 7(b) show the imaginary coordinate plane and the shifting of the region for the aforementioned example. Here the dot is shown for reference purposes only, but keep in mind that the dot is removed for this process. This process continues until the coordinates of the top left part of the bounding box are greater than those of the region, indicating that the entire component is enclosed in the box.



(a)

(b)

Figure 7. (a) Before region shift, (b) After region shift.

4.3.4 Post Processing (Validation of Results). The above mentioned techniques can result in low accuracy, specifically for numbers that were not recognized in the first pass of the Number Recognition. These cases result in choosing the closest component to the dot, which is not always the most accurate solution. For that reason, this final piece of the algorithm plays an extremely important role. The purpose of this final step in the Number Locator is to analyze the components that have been discovered in order to determine if these components fit the criteria (size) of a number, while also making sure that both digits in a two digit number are represented in the region.

Verifying that the size of the located component is comparable to that of the numbers in the puzzle has two elements to it. First, if the region has been shifted and the entire component does not fit within of the region and the top-left edge of the component has not been found, then

the size of the component will be checked. Secondly, if a component is found that does fit within the boundaries of the region, there must be a way of disregarding parts of the puzzle that may have been close enough to the dot to be mistaken for a number. For both of these cases there is a check point in the results validation process which examines the size of the component and compares it to the standard size of a number in this puzzle. If the component found is very different (over 50% difference) in size then it is established that this component found is not the number.

If it is resolved that the component found is not a number, the other components that were initially found in that region are explored. The next component, not previously inspected is sent through all the previously mentioned steps. This procedure is repeated with all the components found in the initial region until one of them satisfies the criteria (size), or it is the only component left near that dot.

Finally, there are cases where a double digit number is located on the left side of the dot. Since TROPE is looking at components closest to the dot, the above described algorithm will find the top left portion of the second digit of a double digit number, leaving out the first digit in a two digit number as exhibited in Figure 8. In order to make sure that part of a two digit number is not being left out, the region is extended in the left direction by the width of the already found component. At this point, the area is surveyed to conclude if an element has been found within this new region. If it has, then the "Finding the Top Left Corner" procedure is executed to find the top left corner of this new component found. Those coordinates are returned as the top left corner of the number found.

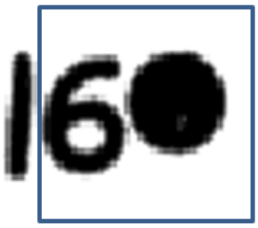


Figure 8. Double-digit number split by region of interest.

Once a number's upper left corner is found, the location and size of the rectangular region is estimated using this position, as well as the standard size of the first pass's most confident number. Standard size is defined as twice the width and height of the most confidently recognized number. This component returns the coordinates of the rectangular region in which the potential number is located. Figure 9 below shows each rectangular region in blue which encloses the number near each dot.

4.4 Number Recognition Algorithm

Utilizing Tesseract, the final component of TROPE attempts to identify the number in each of the specified regions. In this research, the Tesseract engine is also configured to look for only decimal digits, decreasing the variety of characters that it is looking to classify. During the first execution of this component, it is passed the image (with the dots removed) along with the width and height of the entire image (from the Preprocessing algorithm) to analyze. For the final operation it is passed the coordinates of the rectangular regions found by the number locator as well as the original image, after the dots have been removed. Removing the dots from the image help to increase the accuracy of the Tesseract engine by eliminating the incorrect identification of dots as "0"s. The result of the engine for each region is presented as a "word" (the number that is identified), along with a confidence rating for the word (number).

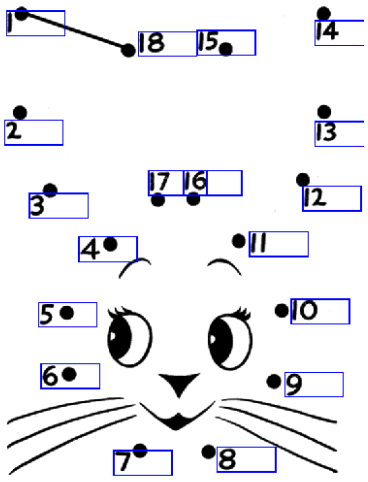


Figure 9. Visual result of the Number Locator.

4.5 Post-Processing Algorithm

Following the number recognition algorithm, a post-processing procedure takes place to "clean up" the results obtained. This algorithm seeks to eliminate results that cannot be possible, for example, numbers whose value exceed the count of dots in the puzzle. As input, the algorithm takes all the numbers recognized by the second pass of the Number Recognition component, as well as the locations and confidence ratings of those numbers. If a three digit number is found, and only a two digit amount of dots have been located by the first component in TROPE, then the three digit number will be split into a two digit and a one digit number. The results will be traversed to determine if any number of the same value as the new two digit number has already been detected. If not, then the previously three digit result is replaced with the new two digit result. If the number has been found, then the aforementioned process is repeated with the one digit number. If neither of these processes works, then the entire result and its location is eliminated. Furthermore, in this step, any duplicate numbers detected will be eradicated by choosing the result with the highest confidence and deleting the other one. The final collection of the recognized numbers, and their locations is output by this component.

4.6 Requirements for proper function

TROPE is designed to analyze Connect-the-Dots puzzle images in GIF, PNG, or JPEG format. The algorithm is most accurate for puzzles with dot sizes ranging from 5-15 pixels in radius length which usually contain, but are not limited to, 14-20 dots. The accuracy of the Dot Locator decreases tremendously with very small dots, which has a negative effect on the TROPE algorithm as a whole.

4.7 Experimental Procedure

A data set of 20 puzzles, with dot counts ranging from 12-20 dots, was gathered for use in this evaluation. It was decided to hand select puzzles of similar form (dot size, dot amount, and simplistic artwork) for this study to evaluate the system as it was designed to operate. The puzzles were obtained from a common online source of connect-the-dot puzzles [26] and were saved in a directory as .gif image files.

4.7.1 Dot locator experiment. A record of the amount of dots existing in each puzzle was manually gathered and saved. The puzzles were loaded into MATLAB and inputted into the Dot Locator function. The dots that were located were outlined in red (Figure 6. *Visual results of the Dot Locator*.) and displayed on the original image. The amount of dots located were counted and the results were compiled into a chart that displays the percentage of the true positive, false negative, and false positive results.

4.7.2 Number locator experiment. The location and size parameters of the dots obtained from the Dot Locator component were passed into the Number Locator function. The function produced an image of the puzzle with the resulting rectangular regions outlined in blue (Figure 9. *Visual result of the Number Locator*.). The amount of appropriately located numbers was counted and compared to the number of correctly identified dots.

4.7.3 Number recognition experiment. In the first part of this experiment the entire puzzle was inputted into the Recognition Component; once with dots, and then again with the dots removed. This portion of this experiment gathered the first pass results of the Tesseract OCR function. Both versions of the image (with and without dots) were tested to verify the best method for gathering first pass results. The Tesseract function outputted the recognized numbers along with a confidence rating for each recognized number. The confidence rating conveys how satisfied the engine is that the given text is properly identified.

In the second portion of this experiment the rectangular sections resulting from the Number Locator component test were passed to the Recognition Component. This component ran the Tesseract OCR function on the image over only the areas specified by those sections. The recognized number results from this test were collected.

4.7.4 Number of dots experiment. A final experiment was conducted in order to test the performance of TROPE on puzzles that contain a large amount of dots. TROPE was executed on JPEG, GIF, and PNG puzzles with dot sizes ranging from 12-91. The recognized numbers for each puzzle were outputted and collected.

CHAPTER 5

Results and Discussion

5.1 Dot Locator Accuracy

The Dot Locator was evaluated over the set of 20 puzzles, with dot counts ranging from 12-20 dots. The accuracy of this component was evaluated in terms of the true positive, false negative and false positive rates. The amount of correctly detected (true positive) dots and the results of the puzzle parts that were misclassified as dots (false positives) for each puzzle are displayed in Figure 10. The graph in Figure 11 depicts that as the amount of dots in the puzzle increases, there is no effect on the accuracy of the function. The component found true positive dots 100% of the time. On the other-hand, only an average of 3% of the total number of results found were false positives.

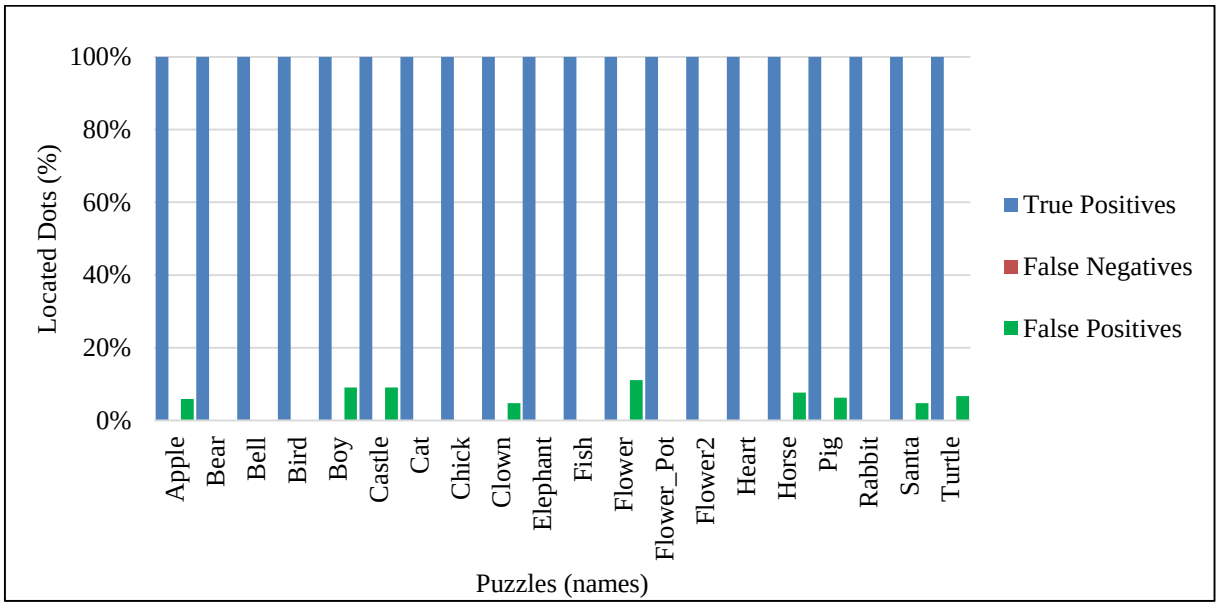


Figure 10. Dot Locator accuracy.

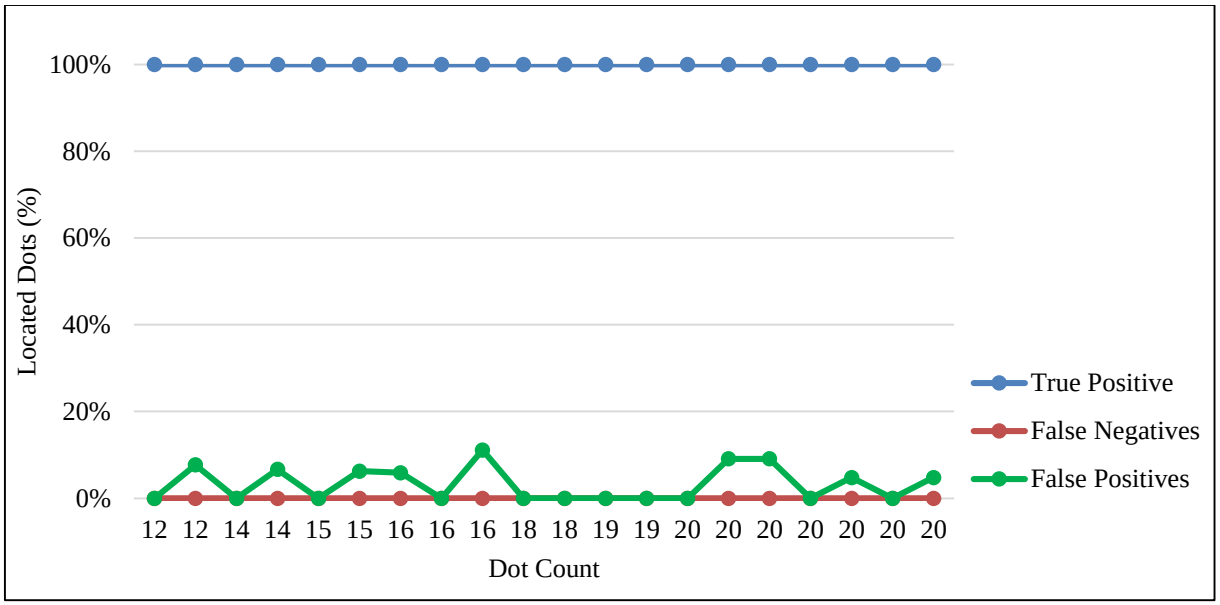


Figure 11. Dot Count vs. Located Dots.

5.2 Number Locator Accuracy

The Number Locator component was evaluated over the same set of puzzles taking into account the amount of dots that were accurately located in the first experiment. The percentage of accurately located numbers is displayed in and in Figure 12. Accurate location of the number is defined as the case when the rectangular region is completely enclosing the number. Numbers that are not fully within a region are not considered properly located by this component. Evaluating this function on only the true positive results of the Dot Locator, omits the effect of the false positive error of the previous component on the results of the current one.

The Number Locator accurately located an average of approximately 86% of the numbers that were associated with the dots that were correctly detected by the Dot Locator. Figure 13 demonstrates that as the amount of "busy-ness" in the puzzle increases, the percentage of accurately located numbers tends to decrease. "Busy-ness" is defined as the percentage of black pixels present in the image. This value corresponds to the amount of all drawing that is present in the puzzle.

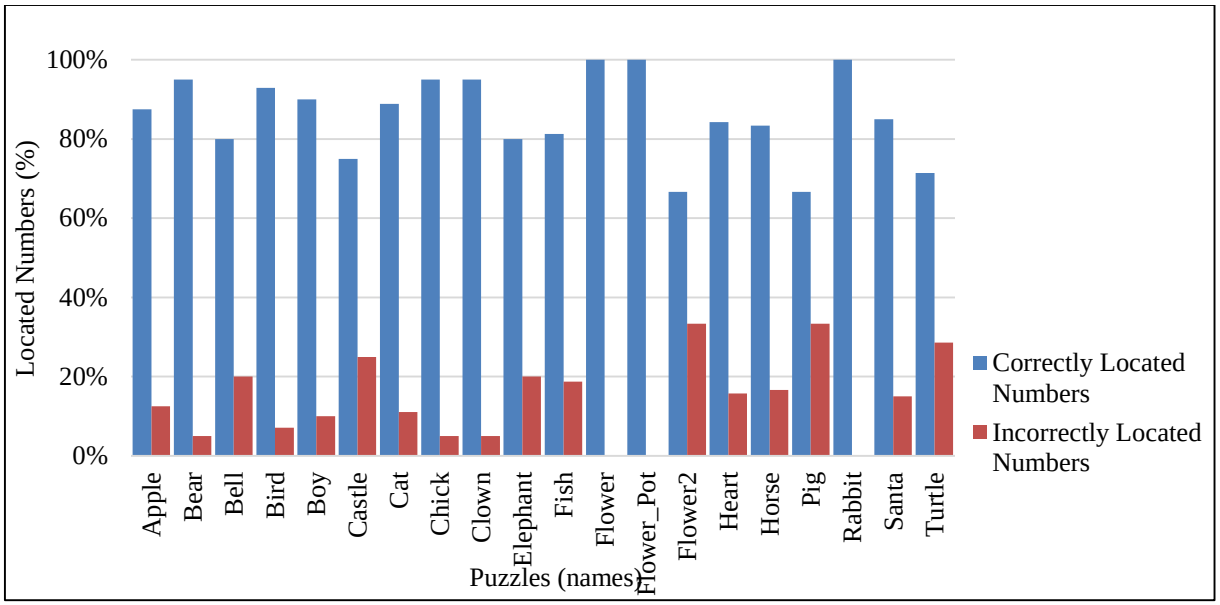


Figure 12. Number Locator accuracy.

Figure 13. Image "Busy-ness" vs. Located Numbers.

5.3 Number Recognition Accuracy

The first part of this experiment was conducted in order to gather the results of the first pass of the Number Recognition component, where the entire image as a whole (without the dots) was examined by Tesseract. It was determined to use the image with the dots removed by comparing the results of the first pass with the dots to the first pass without the dots (Figure 14). The first pass with dots recognized numbers at an average of 14% which was much lower than the first pass without dots. The first pass (without dots) was chosen to be used in this system and was evaluated based on the amount of correctly recognized numbers out of all the numbers present in the puzzle. The results of the first pass are displayed in Figure 15. The first pass of this component accurately located numbers at an average rate of 52%.

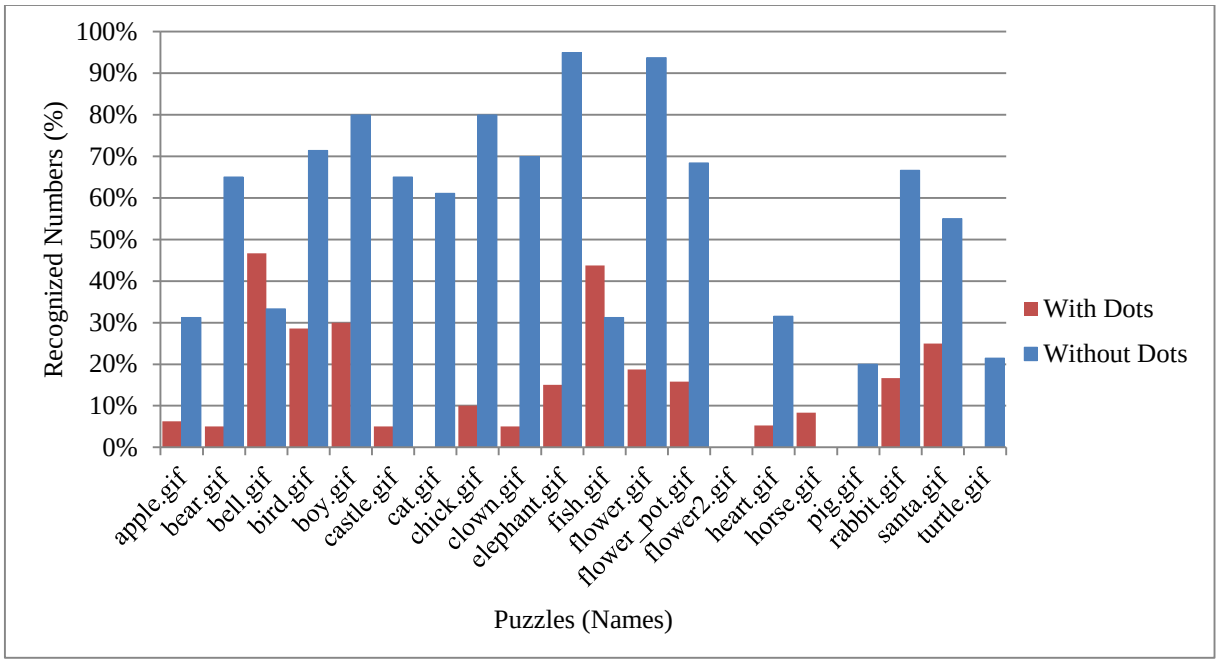


Figure 14. First Pass Recognition With Dots and First Pass Recognition Without Dots.

The second pass of the Number Recognition component was evaluated based on the amount of correctly located numbers output by the Number Locator component. An example of the output of the Number Recognition component can be seen in Appendix B, where the red number represents TROPE's estimation of the number that it is recognizing. The percentage of correctly recognized numbers out of the set of properly located numbers can be found in Figure 16. The percentage of correctly recognized numbers out of the total set of numbers, whether properly located or not, is also presented in and Figure 16. The rate at which the Number Recognition properly identified the properly located numbers was an average of approximately 76%. However, in evaluation of all of the numbers, the average rate at which all of the numbers in the puzzle were properly identified was 66%. Figure 17 shows that as "busy-ness" increases, the accuracy of the recognition algorithm tends to decrease.

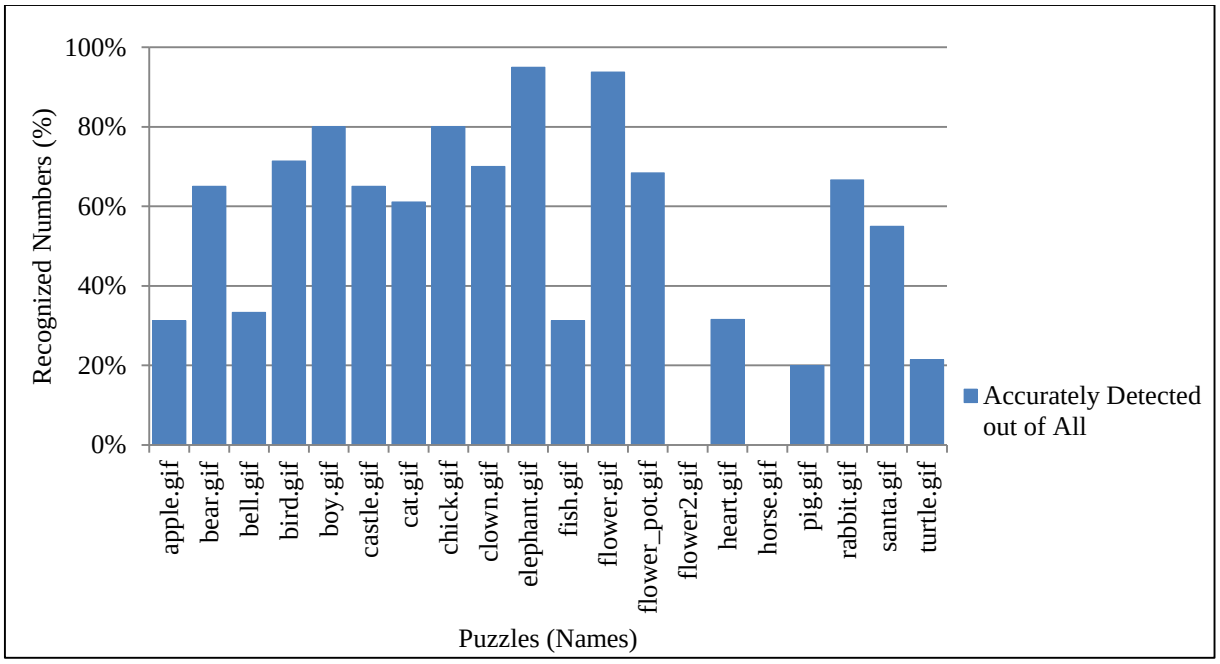


Figure 15. First pass Number Recognition accuracy.

The results of the first and second pass were gathered and compared. The true positive results of the first and second pass recognition can be seen together in Figure 18. Figure 19 shows the amount of objects in each puzzle that were incorrectly identified as numbers, in both the first and second passes. 40% of the first pass recognition results were misrecognized numbers. However, the second pass incorrectly identified numbers at an average rate of 21%.

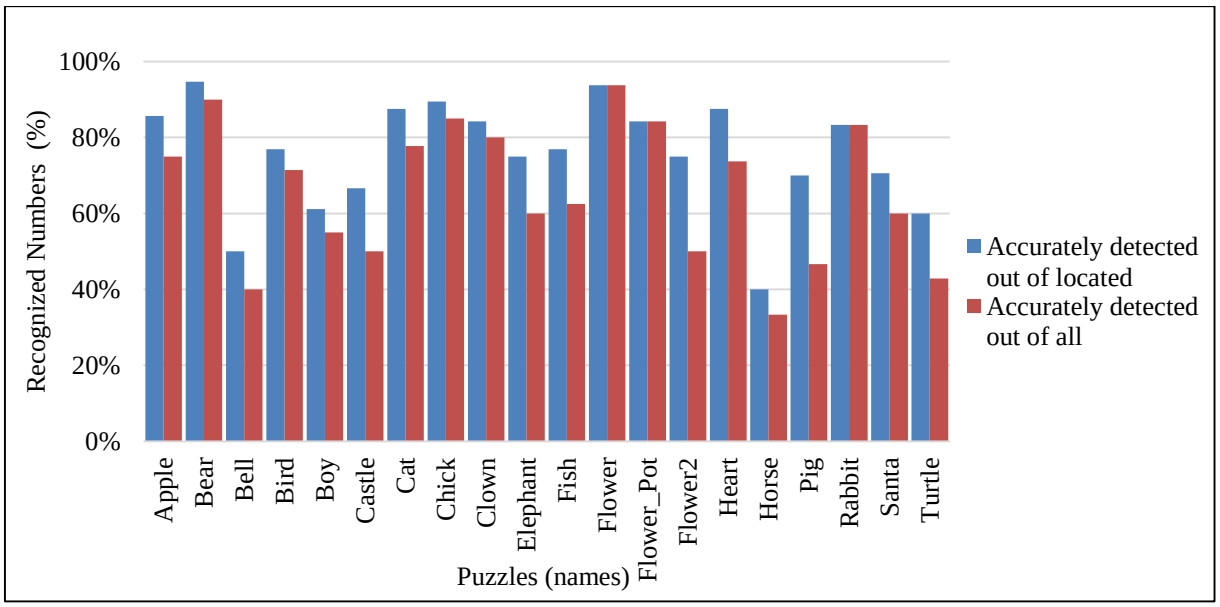


Figure 16. Second Pass Number Recognition accuracy.

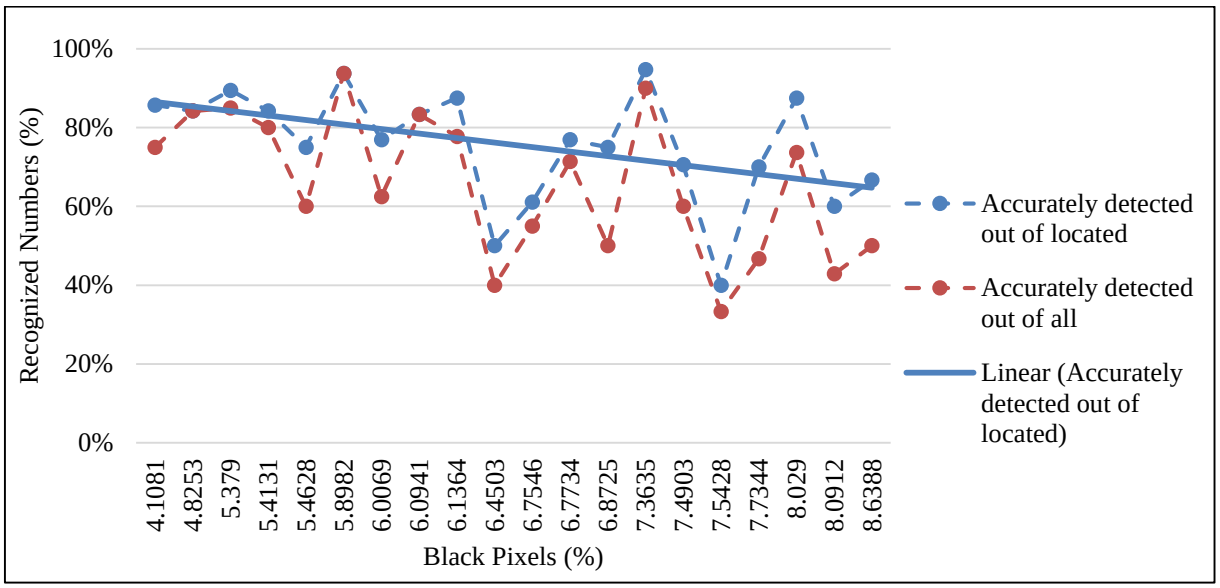


Figure 17. Puzzle "busy-ness" vs. Recognized numbers.

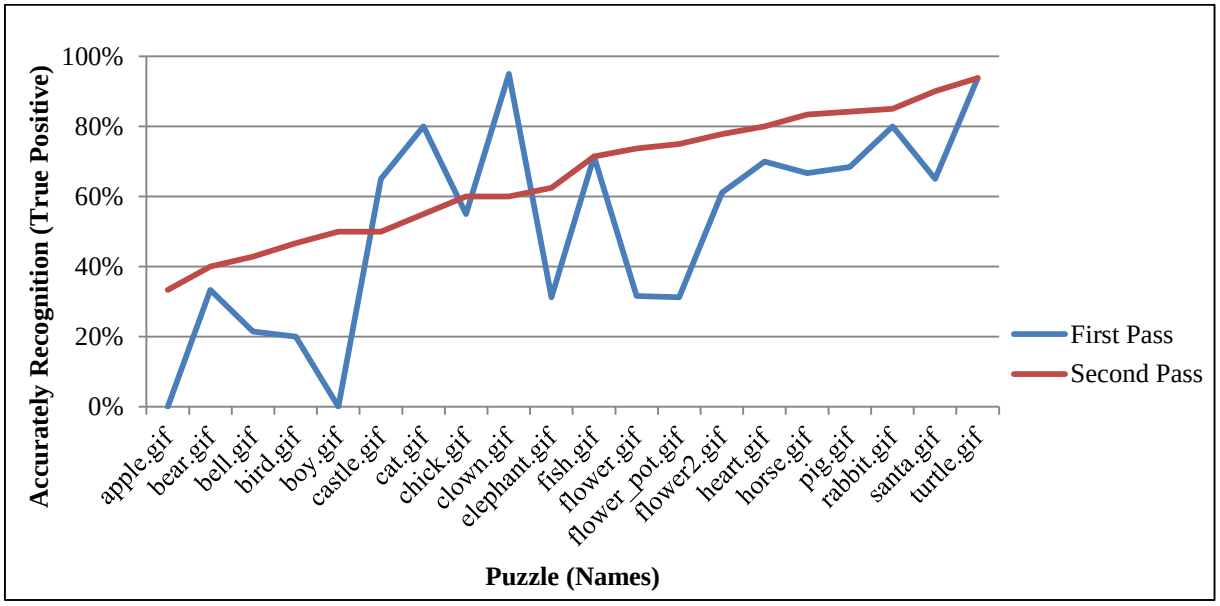


Figure 18. True positive results of the first and second pass of the Number Recognition component.

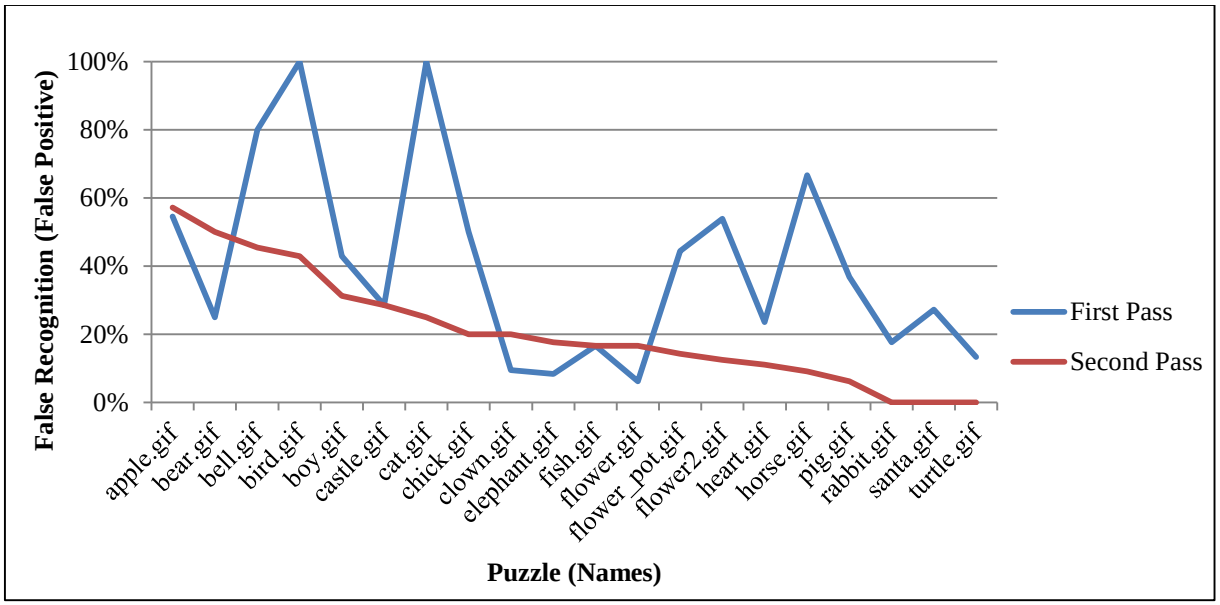


Figure 19. False positive results of the first and second pass of the Number Recognition component.

5.4 Dot Count

In this final experiment, TROPE was evaluated over puzzles with dot counts ranging from 12-91. The amount of correctly recognized numbers are plotted against the dot count of each puzzle in Figure 20. The results show that while it is not true for all cases, generally the recognition accuracy decreases as the dot count increases. Figure 21 is a plot of the "busy-ness" verses the correctly recognized numbers. It is juxtaposed with Figure 22 which displays the relationship between the dot count and the "busy-ness". The two figures demonstrate that the "busy-ness" does not necessarily correspond to the large dot counts and that "busy-ness" does not have a direct parallel to accuracy.

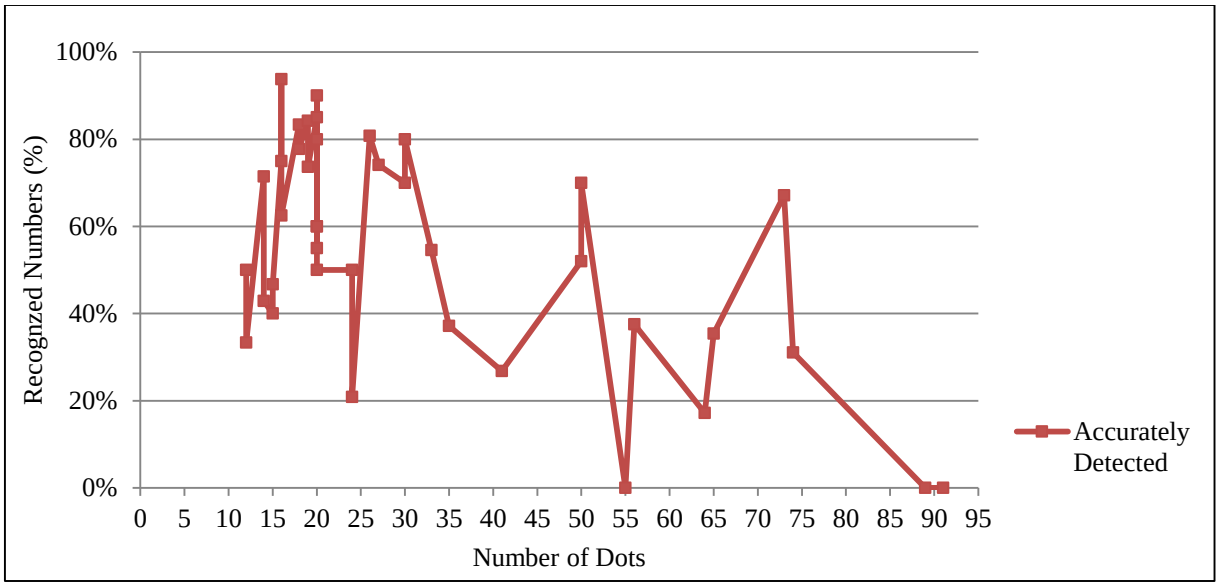


Figure 20. Dot count vs. True positive results of TROPE.

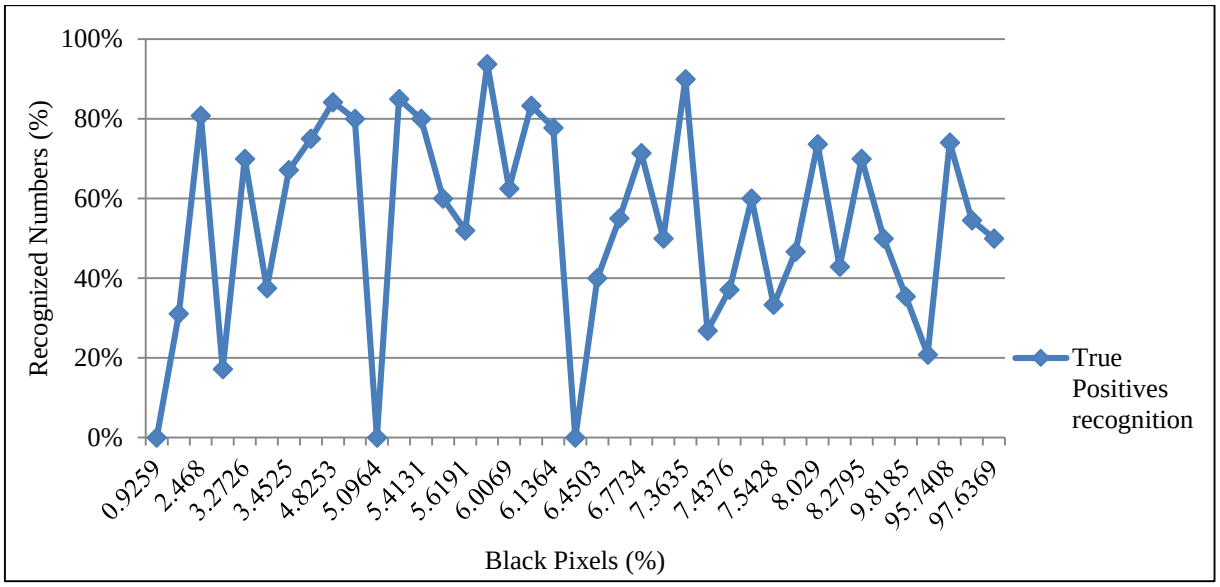


Figure 21. "Busy-ness" vs. True positive results of TROPE.

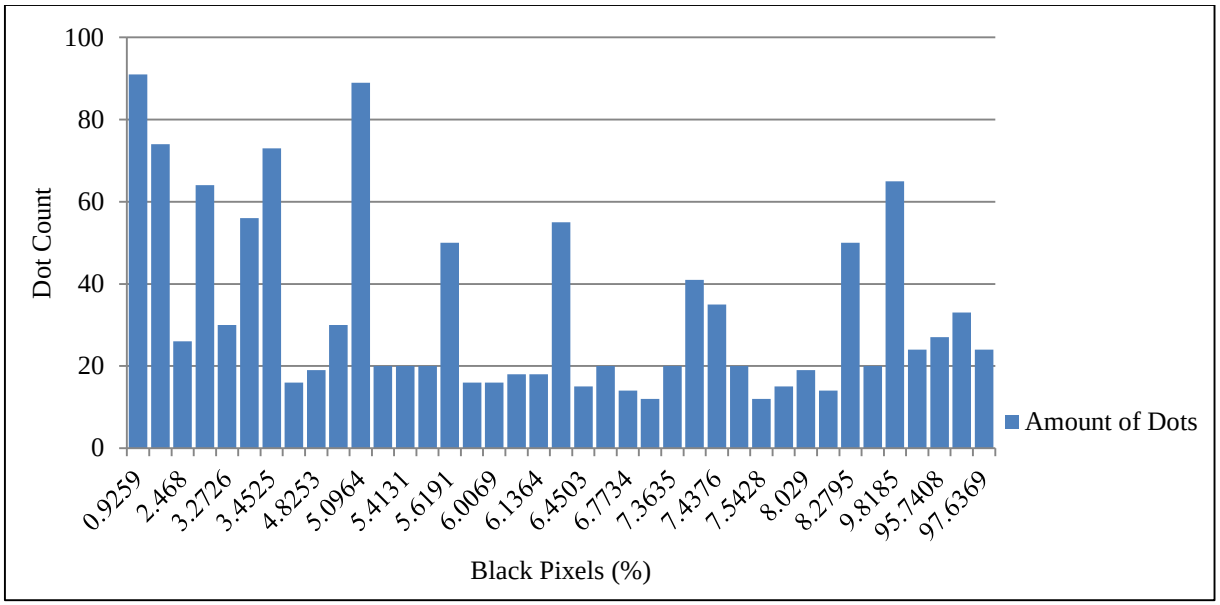


Figure 22. "Busy-ness" vs. Dot count.

CHAPTER 6

Conclusion and Recommendations

6.1 Conclusion

In conclusion, the objective of this research was to create a system that locates and recognizes the numbers in a connect-the-dots puzzle for the purpose of replacing them with characters of the user's choice. The system locates the dots in the puzzle using the Circle Find function in MATLAB [5], locates each number with respect to each dot using a combination of functions from the MATLAB image processing toolbox [5] and finally recognizes the located numbers using the OCR tool, Tesseract. This research studied Connect-the-Dots puzzles with dot sizes ranging from radius size of 5 to 15 pixels in length.

This system used 20 gif images of Connect-the-Dots puzzles for testing purposes. Each puzzle contained from 12 to 20 dots in them with drawings of various complexity. The accuracy of the overall system is 66%, which indicates the percentage of the total numbers in the puzzle that were recognized using the proposed 5 step process. The Dot Locator component by itself had accuracy of 100% while the Number Locator alone had accuracy of 86%, and the Number Recognition component alone had accuracy of 76%. The accuracy of each component was evaluated individually, based on the true positive information provided to it by the previous component.

The Number Locator proved to be affected by the "busy-ness" of the puzzle. This effect is due to the objects in the puzzle that are misclassified as numbers. The Number Recognition component in this case was also negatively affected by the increase in "busy-ness".

The evaluation of the first and second passes of the Number Recognition results displayed that the implementation of TROPE yielded an enriched outcome. The removal of dots produced

about a 271% increase on the accuracy of the first pass. This dramatic difference in accuracy is the reason why the image with no dots was used in the first pass recognition for this system. The second pass recognition produced true positive results at an average rate of 66%, which was an improvement from the first pass rate of 52%. A major contribution of TROPE is demonstrated by the decrease in false positive results from the first pass to the second. This reduction is significant because it is desired to only replace the numbers in the puzzle. The misrepresentation of a number will cause the placement of letters used to replace the numbers in areas that do not contain dots.

The final experiment gave insight on the consequences of the dot count on the accuracy of TROPE. It is demonstrated that in most cases, the recognition accuracy is negatively affected by the increase in dot count. The plots of "busy-ness" verses accuracy and "busy-ness" verses dot count in this experiment demonstrate that there is no direct connection between the "busy-ness" in a puzzle and the accuracy of the system. However, the comparison of the "busy-ness" verses dot count and the "busy-ness" verses accuracy plot suggests that the dot count and "busy-ness" together have an association with the accuracy. For example, when a puzzle has a high dot count and a low "busy-ness" percentage then the accuracy is in turn very low. Conversely, when the dot count is low and the busy-ness factor is high, then there is medium accuracy. When both the dot count and busy-ness factor are low, then the accuracy is high. Finally when, both the dot count and busyness factor are high, then the accuracy tends to be low. A high dot count with a low busy-ness factor suggests that the numbers and dots within the puzzle are very small in relation to the puzzle size. Small dots are more difficult to detect due to the other larger circular objects that may be present within the puzzle.

The TROPE algorithm increased the accuracy of the Tesseract OCR engine by 371%. This measurement refers to executing Tesseract over the entire connect the dot image without any

modification compared to sending the image into the entire TROPE system. Tesseract OCR only accurately recognizes numbers in these puzzles on an average of 14%. However, the addition of the components in the TROPE algorithm causes the recognition accuracy to increase to an average of 66%.

There are many factors that affect the accuracy rate of each component as well as the overall system. The amount of drawing in the puzzle, and how close the non-number components of the drawing come to each dot in the puzzle can both have an effect on the accuracy. The orientation of the numbers can also have an effect on the accuracy of the Recognition Component. The inconsistency of the puzzle metrics (e.g., number sizes in one given puzzle or dot sizes in a given puzzle) can also have an effect on the accuracy.

6.2 Recommendations

The experimental recommendations for this research are to change the algorithm to be able to detect dots that are smaller than 5 pixels and larger than 15 pixels in radius length. This will allow for a broader range of puzzles to be processed, meaning that any puzzle will be able to be processed by the system. Another recommendation is to not rely on the initial results of the OCR tool that is being used. The first pass recognition results are inaccurate which does not make it a reliable source for obtaining initial information regarding the puzzle. Combining the second pass recognition engine with the number locator component would allow the numbers to be located and recognized simultaneously. In this way, all components found near each dot will be evaluated by the recognition component and the highest confidence result will be chosen as the correct component and recognized as the number. This simultaneous localization and recognition will act as a check and balance, allowing for the localization results to be confirmed by the recognition component, while limiting the recognition component to only check the components near the dot.

Finally, the last recommendation would be to design a user friendly GUI for this application, which will allow any user to seamlessly upload an image file of a puzzle. The user would then be able to enter in the word or characters that they would like to have placed in the puzzle, and the system would produce a new image file of the puzzle with the numbers replaced by the user's input.

References

[1] [Online]. Available: http://www.rif.org/.

[2] K. Koile and D. Singer, "Improving Learning in CS1 via Tablet-PC-based In-Class Assessment," in *Second International Computing Education Research Workshop*, Canterbury, 2006.

[3] N. C. Sears and D. M. Johnson, "The Effects of Visual Imagery on Spelling Performance and Retention among Elementary Students," *The Journal of Educational Research*, pp. 230-233, 1986.

[4] R. Smith, "An Overview of the Tesseract OCR Engine," in *12th International Conference on Document Analysis and Recognition*, 2013.

[5] I. The MathWorks, "Image Processing Toolbox," 1994-2014. [Online]. Available: http://www.mathworks.com/products/image/. [Accessed September 2014].

[6] R. Gonzalez and R. Woods, Digital Image Processing: Second Edition, Upper Saddle River: Prentice-Hall, Inc., 2002.

[7] I. T. Young, J. J. Gerbrands and L. J. v. Vliet, Fundamentals of Image Processing, Delft PH Publications, 1998.

[8] J. Miano, Compressed Image File Formats, Reading: Addison Wesley Longman, Inc., 1999.

[9] S. E. Umbaugh, Digital Image Processing and Analysis: Human and Computer Vision Applications with CVIPtools 2nd Edition, Boca Raton, FL: Taylor and Francis Group,LLC, 2011.

[10] B. M. Namee, "Dr. Brian Mac Namee," 9 March 2015. [Online]. Available: http://www.comp.dit.ie/bmacnamee.

[11] K. Jung, K. Kim and A. Jain, "Text Information Extraction in Images and Video: a Survey," *The Journal of the Pattern Recognition Society*, pp. 977-997, 2004.

[12] W. Huang, Z. Lin, J. Yang and J. Wang, "Text Localization in Natural Images Using Stroke Feature Transform and Text Covariance Descriptors," in *Computer Vision (ICCV), 2013 IEEE International Conference on*, Sydney,NSW, 2013.

[13] Y.-F. Pan, X. Hou and C.-L. Liu, "Text Localization in Natural Scene Images based on Conditional Random Field," in *10th International Conference on Document Analysis and Recognition*, Beijing, 2009.

[14] A. G. Ghuneim, "Defining Connectivity," [Online]. Available: http://www.imageprocessingplace.com/downloads_V3/root_downloads/tutorials/contour_tracing_Abeer_George_Ghuneim/index.html.

[15] T. Brosnan and D.-W. Sun, "Inspection and grading of agricultural and food products by computer vision systems-a review," *Computers and Electronics in Agriculture*, pp. 193-213, 2002.

[16] R. Mithe, S. Indalkar and N. Divekar, "Optical Character Recognition," *International Journal of Recent Technology and Engineering (IJRTE)*, pp. 72-75, 2013.

[17] L. Eikvil, "OCR Optical Character Recognition," Norwegian Computing Center, Oslo, Norway, 1993.

[18] C. Patel, A. Patel and D. Patel, "Optical Character Recognition by Open Source OCR Tool Tesseract: A Case Study," *International Journal of Computer Applications Volume 55*, 2012.

[19] H. Gao, W. Wang and Y. Fan, "Divide and Conquer: An Efficient Attack on Yahoo! CAPTCHA," *Trust, Security and Privacy in Computing and Communications, 2012 IEEE 11th International Conference on*, pp. 9-16, 2012.

[20] D. Yamakawa, "Applying Tesseract-OCR to detection of image spam mails," in *Network Operations and Management Symposium (APNOMS), 2012 14th Asia-Pacific*, Seoul, 2012.

[21] Z. Liu and R. Smith, "A Simple Equation Region Detector for Printed Document Images in Tesseract," in *Document Analysis and Recognition(ICDAR), 2013 12th International Conference on*, Washington,DC, 2013.

[22] T. Acharya and P.-S. Tsai, JPEG2000 Standard for Image Compression : Concepts, Algorithms and VLSI Architectures, Hoboken, N.J.: Wiley, 2005.

[23] R. Matthews, "Digital Image File Types Explained," [Online]. Available: http://users.wfu.edu/matthews/misc/graphics/formats/formats.html.

[24] J. George, "GIF, JPG, and PNG- What's the Difference?," Collingwood, 2011.

[25] R. H. Wiggins, C. Davidson, R. Harnsberger, J. Lauman and P. Goede, "Image File Formats: Past, Present, and Future," *RadioGraphics*, vol. 21, no. 3, pp. 789-798, 2001.

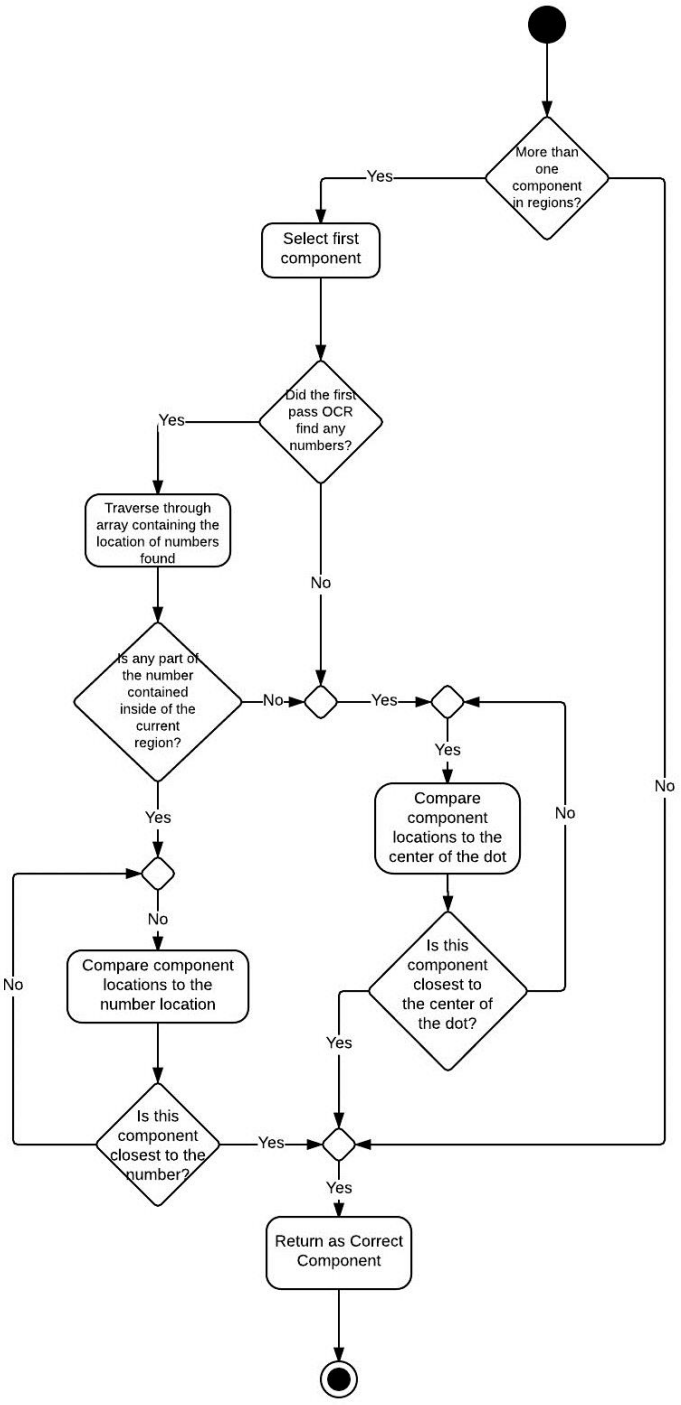
[26] "Easy Dot-to-Dot," BlueBonkers, [Online]. Available: http://www.activity-sheets.com/connect_dots/easydots/. [Accessed 1 January 2014].

[27] B. Nishanthi and S. H. Shahul, "Detection of Text with Connected Component Clustering," *International Journal of Innovative Research in Computer and Communication Engineering*, pp. 2434-2440, 2014.

[28] C. Yi and Y. Tian, "Text String Detection From Natural Scenes by Structure-Based Partition and Grouping," *Image Processing, IEEE Transactions on (Volume: 20, Issue: 9)*, pp. 2594-2605, 2011.

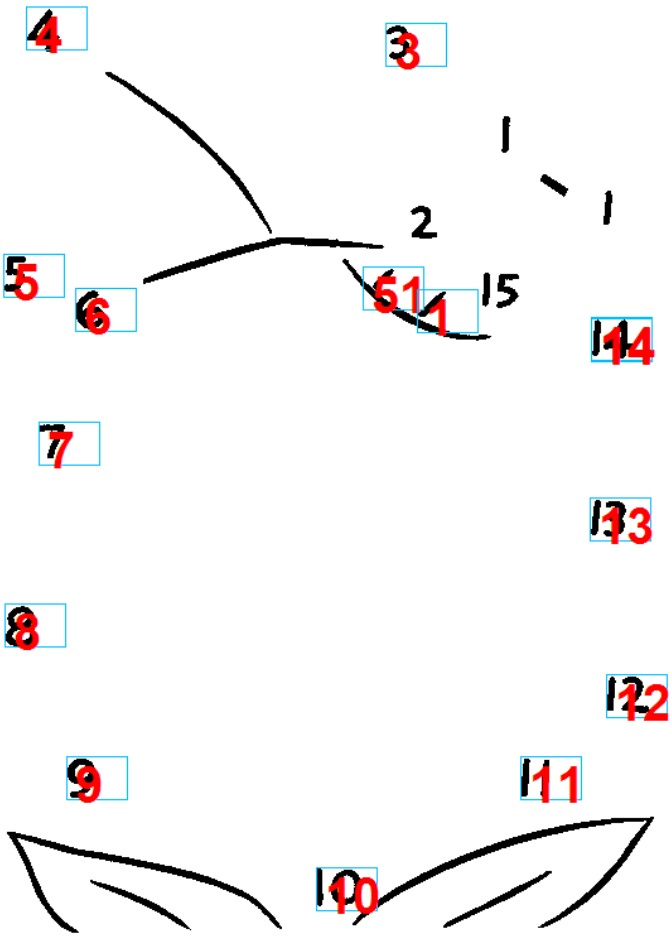
Appendix A

Activity diagram for Choosing a Component



Appendix B

Number Recognition results



Appendix C

Number Recognition Component Code

```
/*Program Executes 2 Passes of Number Recognition using Tesseract OCR engine.
 *Program calls functions created in MATLAB: ImagePreprocessing DotLocator,
 *  and NumberLocator.
 *Program includes a post processing function to validate results of the
 *  second pass recognition
 */
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Runtime.InteropServices;
using MLApp;
using tessnet2;
using System.IO;
using System.Windows;
using System.Windows.Media.Imaging;
using System.Drawing;
using System.Drawing.Imaging;
using System.Drawing.Drawing2D;
using System.Drawing.Text;
using System.Web;
using System.Web.UI;
using System.Web.UI.HtmlControls;
namespace OCRMAT
{
   public class Program
   {
      private MLApp.MLAppClass matlab;
      private System.Object Centers, Centers2;
      public System.Array Centersx, Centersy, Radiis, topx2, topy2, BinaryImageNoDots, BinaryImageWithDots;
      public System.Int32 AmountOfDotsFound, imageheight, imagewidth, xl, yl;
      public System.String filename, ext;
      public Boolean Filechosen, DotsTooSmall;
      public Graphics g;
      Bitmap ImageWithDots, ImageNoDots;

      public Program()
      {
         //Initialize Variables
         DotsTooSmall = false;
         this.matlab = new MLApp.MLAppClass();
         Centersx = new double[40];
         Centersy = new double[40];
         Centers = new double[40][];

         /*Execute Matlab Code: ImagePreprocessing, DotLocator.
          * Pass variable information from MATLAB to .net for execution in this program
          */
         try
         {
            /*MATLAB CODE*/
            matlab.Execute("clear all; clc;");
```

```
            matlab.Execute("[BW, I3, FileChosen, filename,ext,combinedStr,fullimagexl,fullimageyl] =
ImagePreprocessing();");
            matlab.Execute("[centers,radii,BWNoDots,I32,BWWDots,DotsTOOSmall,fullimagexl,fullimageyl] =
DotLocator(BW,I3,FileChosen,filename);");
            //Display Image without dots and with dots
            matlab.Execute("figure(1);imshow(BWNoDots);figure(2);imshow(BWWDots);");

            /*Get variables from MATLAB*/
            BinaryImageNoDots = new int[imagewidth, imageheight];
            BinaryImageWithDots = new int[imagewidth, imageheight];
            String cmb = (String)matlab.GetVariable("combinedStr", "base");
            imageheight = Convert.ToInt32(matlab.GetVariable("fullimageyl", "base"));
            imagewidth = Convert.ToInt32(matlab.GetVariable("fullimagexl", "base"));
            BinaryImageWithDots = (Array)matlab.GetVariable("BWWDots", "base");
            filename = (String)matlab.GetVariable("filename", "base");
            ext = (String)matlab.GetVariable("ext", "base");
            BinaryImageNoDots = (Array)matlab.GetVariable("BWNoDots", "base");
            Centers = (Array)matlab.GetVariable("centers", "base");
            Radiis = (Array)matlab.GetVariable("radii", "base");
            Filechosen = (Boolean)matlab.GetVariable("FileChosen", "base");

            //Original ImageWithDots with dots to allow for proper second pass recognition of each region
            ImageWithDots = new Bitmap(imagewidth, imageheight);

            //Image without dots to allow for first pass recognition and for number location
            ImageNoDots = new Bitmap(imagewidth, imageheight);

            //Fill in ImageWithDots
            for (int i = 0; i < BinaryImageWithDots.GetLength(0); i++)
            {
               for (int j = 0; j < BinaryImageWithDots.GetLength(1); j++)
               {
                  if ((Convert.ToInt16(BinaryImageWithDots.GetValue(i, j))) == 1)
                  {
                     ImageWithDots.SetPixel(j, i, Color.White);
                  }
                  else
                  {
                     ImageWithDots.SetPixel(j, i, Color.Black);
                  }
               }
            }

            //Fill in ImageNoDots
            for (int i = 0; i < BinaryImageNoDots.GetLength(0); i++)
            {
               for (int j = 0; j < BinaryImageNoDots.GetLength(1); j++)
               {
                  if ((Convert.ToInt16(BinaryImageNoDots.GetValue(i, j))) == 1)
                  {
                     ImageNoDots.SetPixel(j, i, Color.White);
                  }
                  else
                  {
                     ImageNoDots.SetPixel(j, i, Color.Black);
                  }
               }
            }
         }
```

```
        catch
        {
            DotsTooSmall = true;
            Console.WriteLine("Unable to Process Image. Check Image Format or The Dots may be too small.");
        }
    }

    public class MATLABInfo
    {
        public Array Centersx;
        public Array Radiis;
        public Int32 xl;
        public Int32 yl;
        public Array topx;
        public Array topy;
        public String cmb;
        public String filename;
        public Bitmap image;
        public Bitmap image2;
        public Int32 length;
    }

    public MATLABInfo MATLABNumberLocator(Bitmap image, String[] FoundWords, int[,] FoundWordsParam,
double avgheight, double avgwidth, Object Centers, Array radiis)
    {
        matlab = new MLApp.MLAppClass();
        Centersx = new double[40];
        Radiis = new double[40];
        topx2 = new double[40];
        topy2 = new double[40];
        bool Tfoundword = true;

        //Determine if Tesseract found any words in the first pass recognition
        if (FoundWords == null)
        {
            Tfoundword = false;
        }
        else
        {
            Tfoundword = true;
        }

        /*Send Information to MATLAB*/
        matlab.PutWorkspaceData("TesseractFoundWord", "base", Tfoundword);
        matlab.PutWorkspaceData("WordParam", "base", FoundWordsParam);
        matlab.PutWorkspaceData("avgheight", "base", avgheight);
        matlab.PutWorkspaceData("avgwidth", "base", avgwidth);
        matlab.PutWorkspaceData("center", "base", Centers);
        matlab.PutWorkspaceData("radii", "base", radiis);

        /*Execute Number Locator in MATLAB*/
        Console.WriteLine("Processing... Please Wait...");
        matlab.Execute("gh = size(radii);for i=1:gh;[BWI2,I32,topx(i),topy(i)] =
NumberLocator(center(i,:),radii(i),BWNoDots,I32,TesseractFoundWord,WordParam,avgheight,avgwidth);end;");
        matlab.Execute("figure(3);I322 = imadjust(I32,[0;1],[0;1]);imshow(I322);");
        //Save Number Locator Results to file in directory
        matlab.Execute("savefig(strcat('C:/',filename,'.fig'));");

        /*Get variables from MATLAB*/
```

```
            Centers2 = (Array)matlab.GetVariable("center", "base");
            Centersx = (Array)Centers2;
            Radiis = (Array)matlab.GetVariable("radii", "base");
            xl = Convert.ToInt32(matlab.GetVariable("fullimagexl", "base"));
            yl = Convert.ToInt32(matlab.GetVariable("fullimageyl", "base"));
            topx2 = (Array)matlab.GetVariable("topx", "base");
            topy2 = (Array)matlab.GetVariable("topy", "base");
            filename = (String)matlab.GetVariable("filename", "base");
            String cmb = (String)matlab.GetVariable("combinedStr", "base");
            AmountOfDotsFound = Radiis.Length;

            //Display center coordinates and radius of all dots found
            for (int i = 0; i < Centersx.Length / 2; i++)
            {
               Console.WriteLine("Center = " + (Centersx.GetValue(i, 0).ToString()) + " Center2 = " +
(Centersx.GetValue(i, 1).ToString()));
               Console.WriteLine("Radius = " + (Radiis.GetValue(i, 0).ToString()));
            }

            var matlabobject = new MATLABInfo { Centersx = Centersx, Radiis = Radiis, xl = xl, yl = yl, topx = topx2,
topy = topy2, cmb = cmb, filename = filename, image = image, image2 = ImageNoDots, length =
AmountOfDotsFound };
            return matlabobject;
         }

         /* Insert recognition results onto Image and save image to specified directory*/
         public void SaveResults(Bitmap tempBitmap, String Text, int Top, int Left, int i, Rectangle rec)
         {
            //Create a graphics object from the original image
            g = Graphics.FromImage(tempBitmap);

            //Specify font style, color
            Font drawFont = new Font("Arial", 30, System.Drawing.FontStyle.Bold);
            Pen skyBluePen = new Pen(Brushes.DeepSkyBlue);

            //specifications for rectangular region of interest
            g.DrawRectangle(skyBluePen, rec.X, rec.Y, rec.Width, rec.Height);
            SolidBrush drawBrush = new SolidBrush(Color.Red);

            //Draw word and rectangle onto picture
            g.DrawString(Text, drawFont, drawBrush, new PointF(Left, Top));

            //Save image results to C:/ directory as "[name-of-file]result.[ext]"
            //Any directory can be used by replacing "C:/" with desired directory path
            tempBitmap.Save("C:" + filename + "result" + ext);
         }

         public class PostProcessingInfo
         {
            public String Number;
            public Double NumberConfidence;
            public List<Word> NewResult;
            public List<Int32> WordTop;
            public List<Int32> WordLeft;
         }

         /*Post processing of results
          * Remove duplicates, remove numbers that are too large, remove ~ (unknown) result,
          * leading 0's, 0's
```

```
        */
       public PostProcessingInfo postprocessing(Bitmap tempBitmap, List<Word> allresult, int radiuslength,
List<Int32> WordTop, List<Int32> WordLeft)
       {

          List<Word> newallresult = allresult;
          string number = null;
          double numberconfidence = 255;

          //Traverse through list of results (all the numbers found in the puzzle)
          try
          {
             for (int c = 0; c < newallresult.Count; c++)
             {
                Word = newallresult.ElementAt(c);
                int cnt = 0;
                bool youcanstopnow = false;
                bool foundone = false;
                string numcheck = null;
                newallresult = allresult;
                number = null;
                numberconfidence = 255;
                int digits = 0;

                if (radiuslength >= 0 && radiuslength < 10)
                {
                   digits = 1;
                }
                else if (radiuslength >= 10 && radiuslength < 100)
                {
                   digits = 2;
                }
                else if (radiuslength >= 100 && radiuslength < 1000)
                {
                   digits = 3;
                }

                //Delete ~ (unkown) results
                //Eliminate ~
                if (word.Text.ToString().CompareTo("~") == 0)
                {
                   newallresult.RemoveAt(c);
                   WordTop.RemoveAt(c);
                   WordLeft.RemoveAt(c);
                   c = c - 1;
                }
                //Eliminate 0
                else if (Convert.ToInt32(word.Text) < 1)
                {
                   newallresult.RemoveAt(c);
                   WordTop.RemoveAt(c);
                   WordLeft.RemoveAt(c);
                   c = c - 1;
                }
                //Eliminate leading 0s
                else if (word.Text.StartsWith("0"))
                {
                   newallresult.ElementAt(newallresult.IndexOf(word)).Text =
newallresult.ElementAt(newallresult.IndexOf(word)).Text.Remove(0, 1);
```

```
            c = c - 1;
        }

        //Eliminate Duplicate
        double lowestconf = word.Confidence;
        for (int x = 0; x < newallresult.Count - 1; x++)
        {
            if (word.Text.CompareTo(newallresult.ElementAt(x).Text) == 0)
            {
                if (newallresult.ElementAt(x).Confidence < lowestconf)
                {
                    newallresult.RemoveAt(c);
                    WordTop.RemoveAt(c);
                    WordLeft.RemoveAt(c);
                    if (c > 0)
                    {
                        c = c - 1;
                    }
                }
            }
        }

        /*Eliminate Large Numbers*/
        //Break into no more than the number of digits in the radius AmountOfDotsFound
        //Break into digits
        //Check to see if either of the digits have already been found
        //If a digit has not been found then assume that number is the digit
        //If all of the single digits have been found then split into double digits if necessary
        //check to see if either number under the number of radiuslength has been found
        //if it has not then assume it is that number
        if (word.Text.CompareTo("~") == 1 && Convert.ToInt32(word.Text) > radiuslength)
        {
            int groupintwos = 0;
            foreach (Character in word.CharList)
            {
                groupintwos += 1;
                numcheck = numcheck + character.Value.ToString();
                if (groupintwos % digits == 0)
                {
                    for (int x = 0; x < newallresult.Count - 1; x++)
                    {
                        if (numcheck.CompareTo(newallresult.ElementAt(x).Text) == 0)
                        {
                            foundone = true;
                        }
                    }
                    if (foundone == false)
                    {
                        newallresult.ElementAt(newallresult.IndexOf(word)).Text = numcheck;
                        youcanstopnow = true;
                    }
                }
                if (youcanstopnow == true)
                {
                    break;
                }
            }
        }
    }
```

```
        }
        catch
        {
            newallresult = allresult;
        }
        var postproccesingobject = new PostProcessingInfo { Number = number, NumberConfidence =
numberconfidence, NewResult = newallresult, WordLeft = WordLeft, WordTop = WordTop };
        return postproccesingobject;
    }
    [STAThread]

    /*Main Method*/
    public static void Main()
    {
        Console.WriteLine("Welcome to TROPE!");
        Console.WriteLine("Please Upload a Connect-the-Dot Image");

        Program m = new Program();

        /*If an image file was chosen then run TROPE*/
        if (m.Filechosen == true || m.DotsTooSmall == false)
        {
            var ocr = new Tesseract();

            //Refine Tesseract engine to only recognize numbers.
            ocr.SetVariable("tessedit_char_whitelist", "01234567890");

            //Language package
            ocr.Init(@"C:\Users\Shelby\Documents\Visual Studio 2013\Projects\OCRDEMO\C#\tessdata", "eng", true);



            //Initiate MATLAB object
            MATLABInfo matobj = new MATLABInfo();
            Bitmap tempBitmap = new Bitmap(m.ImageNoDots.Width, m.ImageNoDots.Height);
            //Graphics g = Graphics.FromImage(tempBitmap);
            // g.DrawImage(m.ImageNoDots, 0, 0);
            // g.SmoothingMode = SmoothingMode.AntiAlias;


            //Region of Interests from Number Locator for second pass recognition
            Rectangle rec = new Rectangle();

            //Rectangle that specifies exact size of number found by Tesseract
            Rectangle rect = new Rectangle();
            int wl, hl;
            int sx, sy;
            int cnt = 0;

            /**FIRST PASS RECOGNITION**/
            var result = ocr.DoOCR(m.ImageWithDots, Rectangle.Empty);

            string[] FoundWords = new string[result.Count()];
            int[,] FoundWordsParam = new int[result.Count(), 4];
            List<double> FoundWordsConfidence = new List<double>();
            List<Word> AllResults = new List<Word>();
            List<Int32> WordTop = new List<Int32>();
            List<Int32> WordLeft = new List<Int32>();
            double avgheight = 0;
```

```
double avgwidth = 0;
double sumheight = 0;
double sumwidth = 0;

/*Save only the MOST confident First Pass Results*/
foreach (Word in result)
{
   if (word.Confidence < 127)
   {

      FoundWordsConfidence.Add(word.Confidence);
      FoundWords[cnt] = word.Text;
      FoundWordsParam[cnt, 0] = word.Top;
      FoundWordsParam[cnt, 1] = word.Bottom;
      FoundWordsParam[cnt, 2] = word.Left;
      FoundWordsParam[cnt, 3] = word.Right;
      cnt = cnt + 1;
   }
}

/*Calculate Standard height and width of a number in this puzzle using the
 * dimensions of the MOST confident number found by the first pass */
try
{
   avgheight = 0;
   avgwidth = 0;
   sumheight = 0;
   sumwidth = 0;
   int t = 0;
   double lowestconfidence = 0;
   bool equalsone = true;

   //If the highest confidence number is 1 then choose the next highest confidence number
   while (equalsone)
   {
      if (Convert.ToInt32(FoundWords[t]) == 1)
      {
         t = t + 1;
      }
      else
      {

         lowestconfidence = FoundWordsConfidence[t];
         avgheight = (FoundWordsParam[t, 1] - FoundWordsParam[t, 0]);
         avgwidth = (FoundWordsParam[t, 3] - FoundWordsParam[t, 2]);
         equalsone = false;
      }
   }

   //Calculate width and height of the most confident number
   for (int count = 0; count < FoundWordsConfidence.Count; count++)
   {
      if (FoundWordsConfidence[count] < lowestconfidence && Convert.ToInt32(FoundWords[count]) != 1)
      {
         lowestconfidence = FoundWordsConfidence[count];
         avgheight = (FoundWordsParam[count, 1] - FoundWordsParam[count, 0]);
         avgwidth = (FoundWordsParam[count, 3] - FoundWordsParam[count, 2]);
      }
   }
```

```
            }
            /*If a number has not been found that is confident enough then use the radius of the
               circle to create a standard size*/
            catch
            {

               for (int i = 0; i < m.Radiis.Length; i++)
               {
                  sumheight = sumheight + Convert.ToInt32(m.Radiis.GetValue(i, 0));
               }
               avgheight = (sumheight / m.Radiis.Length) * 2;
               avgwidth = avgheight;
               FoundWords = null;
            }

            /*Run Numer Locator Component*/
            matobj = m.MATLABNumberLocator(m.ImageNoDots, FoundWords, FoundWordsParam, avgheight,
avgwidth, m.Centers, m.Radiis);

            for (int i = 0; i < matobj.topx.GetLength(1); i++)
            {
               /*Create the rectangle to be fed into the OCR engine
                * from the region of interest for each number*/
               wl = (int)Math.Ceiling(avgwidth) * 2;
               hl = (int)Math.Ceiling(avgheight) + 2;
               sx = Convert.ToInt32(matobj.topx.GetValue(0, i));
               sy = Convert.ToInt32(matobj.topy.GetValue(0, i));
               rec = new Rectangle(sx, sy, wl, hl);
               if ((rec.X + rec.Width) > m.xl)
               {
                  wl = Math.Abs(m.xl - rec.X);
                  rec.Width = wl;
               }
               if ((rec.Y + rec.Height) > m.yl)
               {
                  hl = Math.Abs(m.yl - rec.Y);
                  rec.Height = hl;
               }
               if (sx < 1)
               {
                  sx = 1;
                  rec.X = sx;
               }
               if (sy < 1)
               {
                  sy = 1;
                  rec.Y = sy;
               }
               /**SECOND PASS NUMBER RECOGNITION**/
               var result2 = ocr.DoOCR(matobj.image, rec);
               int cnt2 = 0;

               /*Save results to AllResults list*/
               foreach (Word in result2)
               {
                  AllResults.Add(word);
                  cnt2 += 1;
                  var Left = rec.X + word.Left;
                  WordLeft.Add(Left);
```

```
                var Top = rec.Y + word.Top;
                WordTop.Add(Top);
                rect = new Rectangle(Left, Top, (word.Right - word.Left), (word.Bottom - word.Top));
            }
        }

        /*Run Postprocessing Component*/
        PostProcessingInfo postprocessobj = new PostProcessingInfo();
        int ct = 0;
        List<Word> AllResults2 = AllResults;
        postprocessobj = m.postprocessing(matobj.image, AllResults, matobj.length, WordTop, WordLeft);
        ct = ct + 1;

        //Save Final Results
        for (int i = 0; i < postprocessobj.NewResult.Count; i++)
        {
            wl = (int)Math.Ceiling(avgwidth) * 2;
            hl = (int)Math.Ceiling(avgheight) + 2;
            sx = Convert.ToInt32(postprocessobj.WordLeft.ElementAt(i));
            sy = Convert.ToInt32(postprocessobj.WordTop.ElementAt(i));
            rec.X = sx;
            rec.Y = sy;
            rec.Width = wl;
            rec.Height = hl;

            if ((rec.X + rec.Width) > m.xl)
            {
                wl = m.xl - rec.X;
            }
            if ((rec.Y + rec.Height) > m.yl)
            {
                hl = m.yl - rec.Y;
            }
            if (sx < 1)
            {
                sx = 1;
            }
            if (sy < 1)
            {
                sy = 1;
            }
            /*Save Results by drawing results onto image and saving it to a directory on computer*/
            m.SaveResults(matobj.image, postprocessobj.NewResult.ElementAt(i).Text,
postprocessobj.WordTop.ElementAt(i), postprocessobj.WordLeft.ElementAt(i), cnt, rec);
        }

        //Finish Program
        Console.WriteLine("FINISHED!");
        m.matlab.Quit();
        Console.ReadLine();
        m.g.Dispose();
        tempBitmap.Dispose();
        Environment.Exit(0);
    }
    else
    {
        Console.WriteLine("Thank You. Goodbye.");
        Console.ReadLine();
        Environment.Exit(0);
```

```
        }
      }
    }
}
```

Appendix D

Image Preprocessing Component Code

```
%Function allows user to select an image file from their computer directory
%The image is resized if the width or height is less than 500 pixels.
%The image is converted to a binary image
%Outputs:
% BW    Final Binary Image
% I3    Final Grayscale Image
% filename     Name of file and extension as listed in computer the
%              directory
% ext          Name of extension to the file
% combinedStr  Name of Path to file as listed in computer the directory
% fullimagexl  Width of image, in pixels
% fullimageyl  Height of image, in pixels
%This function takes no inputs
function [BW, I3, FileChosen,filename,ext,combinedStr,fullimagexl,...
    fullimageyl] = ImagePreprocessing()

%Prompt user to select the file
[filename,pathway] = uigetfile('*.*','Select a file','C:');

%Make sure a file is chosen
if isequal(filename,0)
    FileChosen = false;
    BW = 0;
    I3 = 0;
    combinedStr = [];
    filename = [];
    fullimagexl = 0;
    fullimageyl = 0;
else

    FileChosen = true;
    combinedStr = strcat(pathway,filename);
    [~,~,ext] = fileparts(combinedStr);
    [Image,map] = imread(combinedStr);

%Determine the file format and resize the image if necessary
%Convert the image to a binary image
    if (strcmp(ext,'.png'))
        if(isempty(map))
            I3=Image;
            I3 = imadjust(I3);
         else
            I3= ind2gray(Image,map);
        end
        th = graythresh(I3);
        BW = im2bw(I3,th);

    elseif(strcmp(ext,'.jpg'))
        if(isempty(map))
            I3 = Image;
```

```
        else
            I3= ind2gray(Image,map);
        end
        th = graythresh(I3);
        BW = im2bw(I3,th);

    elseif(strcmp(ext, '.gif'))
        if(isempty(map))
            I3 = imadjust(Image);
        else
            I3= ind2gray(Image,map);
        end
            th = graythresh(I3);
            BW = im2bw(I3, th);

    else
        disp('Improper File Format');
        FileChosen = false;
        BW = 0;
        I3 = 0;
        combinedStr = [];
        filename = [];
    end
    [fullimageyl,fullimagexl] = size(BW);
end
end
```

Appendix E

Dot Locator Component Code

```
%%%%%Dot Locator Component Code%%%%%
%Function takes a Connect-the-Dots puzzle in the form of an image file as
%input, and outputs the pixel coordinates of the center of all the dots, as
%a matrix, and the radius size of the dots as a vector.
%Using a imfindcircles()
%Inputs:
% BW            Final Binary Image
% I3            Final Grayscale Image
% FileChosen    Boolean value that tells if an image was selected from
%               the directory
% filename      Name of file and extension as listed in computer
%                the directory
%Outputs:
% centersnet    Final matrix containing the coordinates of the dots'
%               centers
% finalradii    Final matrix containing the size of the dots' radius
% BWNoDots      Final Binary Image after dots have been removed
% I3NoDots      Final Grayscale Image after dots have been removed
% BWWithDots    Final Binary Image before dots have been removed
% DotsTOOSmall  Boolean value that tells whether dots are too small for
%               recognition
% fullimagexl   Width of image, in pixels
% fullimageyl   Height of image, in pixels
function [finalcenters,finalradii,BWNoDots,I3NoDots,BWWithDots,...
    DotsTOOSmall,fullimagexl,fullimageyl] = DotLocator(BW,I3,FileChosen,...
    filename)

%Make sure a file was chosen from the directory
%User imfindcircles to look for circles in puzzle that have radius sizes
%between 5 and 15 pixels
if(FileChosen == true)
    finalcenters = [];
    DotsTOOSmall = false;
    rmv = [];
    radiusRange = [5,15];

    %First pass dot find
    [center,radi] = imfindcircles(BW,radiusRange,'ObjectPolarity','dark'...
        ,'Sensitivity',0.5,'Method','twostage');
    M = mode(radi);
    M2 = [ceil(M-1),floor(M+1)];

    %Try for second pass dot find with smaller radius range (mode of
    %radius' from first pass
    try
        [centers,finalradii]= imfindcircles(BW,M2,'ObjectPolarity',...
            'dark','Sensitivity',0.85,'Method','twostage');
    catch
        centers = center;
        finalradii = radi;
    end
```

```
%Remove results that contain a lot of white pixels
c = 1;
for i = 1:size(finalradii)
    x=[(centers(i,1)-(finalradii(i))):(centers(i,1)+(finalradii(i)))...
        (centers(i,1)):(centers(i,1))];
    y=[(centers(i,2)):(centers(i,2)) (centers(i,2)-(finalradii(i))):...
        (centers(i,2) +(finalradii(i)))];
    pixels = impixel(BW,x,y);
    cnt = numel(pixels);
    z = find(pixels==1);
    cntz = size(z);
    pixels2=impixel(BW,centers(i,1),centers(i,2));
    if(cntz(1,1)>=(cnt*(5/12)))
       rmv(c) = i;
       c = c+1;
    elseif(pixels2 == 1)
       rmv(c) = i;
       c = c +1;
    end
end
if(isempty(rmv)==false)
    centers(rmv,:) = [];
    finalradii(rmv) = [];
end
    figure(1);
    imshow(BW);
    centers;
    finalradii;
    h3 = viscircles(centers,finalradii);
    savefig(strcat('C:\Users\Shelby\Documents\Masters\AMPED\Results'...
    ,'\addedresults\dots',filename,'.fig'));

%"Remove" dot by inserting a white circle over it
try
    RGB = insertShape(I3, 'FilledCircle', [centers(:,1) centers(:,2)...
        finalradii(:)+1.5], 'LineWidth', 8,'Color',[255,255,255]);
    I3NoDots = imadjust(RGB,[0;1],[0;1]);

catch
    I3NoDots = I3;
    DotsTOOSmall = true;
end
BW22 = im2bw(I3NoDots);
BWNoDots = BW22;
ghh = size(finalradii);

%If Image is too small to locate the dots then resize the image and
% recursively run the Dot Locator
if(isempty(centers) || isempty(finalradii)|| DotsTOOSmall == true ||...
        ghh(1)<10)
    DotsTOOSmall = true;
    [fullyl,fullxl] = size(I3);
    scale = 2;
    [I3NoDots] = imresize(I3, scale);
    th = graythresh(I3NoDots);
```

```
        BW = im2bw(I3NoDots,th);
        BWWithDots = BW;

        [finalcenters,finalradii,BWNoDots,I3NoDots,BWWithDots,...
            DotsTOOSmall,fullimagexl,fullimageyl]=DotLocator(BW,...
            I3NoDots,FileChosen,filename);
    else
        finalcenters = NET.createArray('System.Double[]',2,...
            length(centers));
        finalcenters = centers;
    end
    %Output final binary image and its size
    BWWithDots = BW;
    [fullimageyl,fullimagexl] = size(BWNoDots);
end
```

Appendix F

Number Locator Component Code

```
%Function determines the region that the number is located in
%Inputs:
% centers               Matrix containing the coordinates of the dots'
centers
% radii                 Matrix containing the size of the dots' radius
% BW22                  Binary Image
% I3                    Grayscale Image
% TesseractFoundWord    Boolean value that tells that the first pass
%                       recognition returned a number with high confidence
% WordParam             Location and size of the number returned by the
%                       first pass
% standheight           The standard height of a number in this puzzle
% standwidth            The standard width of a number in this puzzle
%
%Outputs:
% BWIR          Binary Image
% I322          Grayscale Image
% topx          X-coordinates of top left corners of the regions that
%               contain the numbers
% topy          Y-coordinates of top left corners of the regions that
%               contain the numbers
function [BWI2,I322,topx,topy] = NumberLocator(centers,radii,BW22,I3,...
    TesseractFoundWord,WordParam,standheight,standwidth)

%Initialize variables
topx = [];
topy = [];
QuadXValue = centers(1);
QuadYValue = centers(2);
QuadI = false;
QuadII = false;
QuadIII = false;
QuadIV = false;
Right = false;
Left = false;
Bottom = false;
Top = false;
CompCount = 1;

%Initialize initial region size
c = [(centers(1)+radii*2) (centers(1)-radii*2) (centers(1)-radii*2)...
    (centers(1)+radii*2)];
r = [(centers(2)+radii*2) (centers(2)+radii*2) (centers(2)-radii*2)...
    (centers(2)-radii*2)];
regioniscorrect = false;
fitsinregion = true;
moveright = false;
moveleft = false;
moveup = false;
movedown = false;
addtoit = 0;
```

```
regionisbigenough = false;
expanding = 0;
totaladded= 0;
TesseractWord = [];

%Start with standard region of interest size, and expand
%region of interest size until a component is found within the region
%AvgDistNumToDot
while(regionisbigenough==false)
    [BWroi,xi,yi] = roipoly(BW22,c,r);
    BWI2 = BW22;
    disp('BWI2');
    BWI2(BWroi) = 1-BW22(BWroi);
    BWI2(~BWroi) = cast(0,class(BWI2));
    CC = bwconncomp(BWI2);
    CC.NumObjects;
    if(CC.NumObjects == 0)
        regionisbigenough = false;
        c = [];
        r=[];
        addtoit = addtoit + 0.5;
        c = [(centers(1)+radii*(2+addtoit)) (centers(1)-radii*(2+addtoit))
(centers(1)-radii*(2+addtoit)) (centers(1)+radii*(2+addtoit))];
        r = [(centers(2)+radii*(2+addtoit)) (centers(2)+radii*(2+addtoit))
(centers(2)-radii*(2+addtoit)) (centers(2)-radii*(2+addtoit))];
    else
        regionisbigenough = true;
        STATS = regionprops(CC,'PixelList','Centroid','BoundingBox');
        centroids = cat(1, STATS.Centroid);
    end
end
originalc = c;
originalr = r;
correctcomp = [];
thisBB = zeros(length(STATS),4);

%Get pixels, bounding box and centroid for all connected components
%in the region
for k = 1 : length(STATS)
    pixels{k} = STATS(k).PixelList;
    thisBB(k,:) = STATS(k).BoundingBox;
    cents(k,:) = STATS(k).Centroid;
end;

if(length(STATS)>1)
    MoreThanOneComponent = true;
else
    MoreThanOneComponent = false;
end
[RightComponent,TesseractFoundACloseWord,TesseractWord] =...
    FindCorrectComponent(TesseractFoundWord,WordParam,thisBB,...

MoreThanOneComponent,STATS,standheight,standwidth,QuadYValue,QuadXValue);
OriginalRightComponent(CompCount,:)=RightComponent;

%Look for the correct component in the region
```

```
while(regioniscorrect == false)
    try
        if(CC.NumObjects ~= 0)
            centroids = cat(1, STATS(RightComponent).Centroid);

        end
    catch
%Determine which component is most likely the number
        [RightComponent,TesseractFoundACloseWord,TesseractWord] = ...
            FindCorrectComponent(TesseractFoundWord,WordParam,thisBB,...
            MoreThanOneComponent,STATS,standheight,standwidth,QuadYValue,...
            QuadXValue);
        if(CC.NumObjects ~= 0)
            centroids = cat(1, STATS(RightComponent).Centroid);
        end
    end

%Determine which Quadrant the component is in
    [morethanoneside,QuadI,QuadII,QuadIII,QuadIV,Right,Left,Bottom,Top]...
        = CheckQuad(centers,pixels{RightComponent});

%Depending on what Quadrant it is in move to find the edge of the
    %component
    if(QuadI == true)
        [topx, topy, fitsinregion,moveup,movedown, moveleft,moveright]...
            = QuadISide(thisBB(RightComponent,:));
    elseif(QuadII == true)
        [topx, topy, fitsinregion,moveup,movedown, moveleft,moveright]...
            = QuadIISide(thisBB(RightComponent,:));
    elseif(QuadIII == true)
        [topx, topy, fitsinregion,moveup,movedown, moveleft,moveright]...
            = QuadIIISide(thisBB(RightComponent,:));
    elseif(QuadIV == true)
        [topx, topy, fitsinregion,moveup,movedown, moveleft,moveright]...
            = QuadIVSide(thisBB(RightComponent,:));
    elseif(Right == true)
        [topx, topy,fitsinregion,moveup,movedown, moveleft,moveright]...
            = RightSide(thisBB(RightComponent,:));
    elseif(Left == true)
        [topx, topy, fitsinregion,moveup,movedown, moveleft,moveright]...
            = LeftSide(thisBB(RightComponent,:));
    elseif(Bottom == true)
        [topx, topy, fitsinregion,moveup,movedown, moveleft,moveright]...
            = BelowSide(thisBB(RightComponent,:));
    elseif(Top == true)
        [topx, topy, fitsinregion,moveup,movedown, moveleft,moveright]...
            = AboveSide(thisBB(RightComponent,:));
    else
    end

%If the whole component is not found in the region then shift the region
    if(fitsinregion == false)
        expanding = expanding + 1;

%If the region has been shifted too many times then the component is too
%big for the region, so choose another component
```

```
        if(totaladded >= standwidth || totaladded >= standheight)
            thiswidth = thisBB(1,3);
            heightpercenterror=(abs(thisBB(1,4)-
standheight)/standheight)*100;
            widthpercenterror = (abs(thisBB(1,3)-
standwidth)/standwidth)*100;
            totalpercenterror = sqrt(heightpercenterror^2+...
                widthpercenterror^2);
            [BWroi,xi,yi] = roipoly(BW22,originalc,originalr);
            BWI2 = BW22;
            BWTRY = BWI2;
            BWI2(BWroi) = 1-BW22(BWroi);
            BWTRY(BWroi) = 1-BW22(BWroi);
            BWTRY(~BWroi) = cast(0,class(BWTRY));
            CC = bwconncomp(BWTRY);
            STATS = regionprops(CC,'PixelList','Centroid','BoundingBox');
            modifiedSTATS = STATS;
            modifiedSTATS(OriginalRightComponent,:) = [];
            thisBB = zeros(length(STATS),4);
            pixels = cell(length(STATS));
            cents = zeros(length(STATS),2);
            for k = 1 : length(STATS)
                pixels{k} = STATS(k).PixelList;
                thisBB(k,:) = STATS(k).BoundingBox;
                cents(k,:) = STATS(k).Centroid;
            end;
            modifiedthisBB = thisBB;
            modifiedthisBB(OriginalRightComponent,:) = [];
            if(length(modifiedSTATS)>1)
                MoreThanOneComponent = true;
            else
                MoreThanOneComponent = false;
            end
%Look for the component that is most likely the number
            [RightComponent,TesseractFoundACloseWord,TesseractWord] =...
                FindCorrectComponent(TesseractFoundWord,WordParam,...
                modifiedthisBB,MoreThanOneComponent,modifiedSTATS,...
                standheight,standwidth,QuadYValue,QuadXValue);
            try
                [rows,cols] = find(ismember(thisBB,...
                    modifiedthisBB(RightComponent,:),'rows'));
                RightComponent = rows;
                CompCount= CompCount + 1;
                OriginalRightComponent(CompCount,:) = RightComponent;
                c = originalc;
                r = originalr;
                totaladded = 0;
                expanding = 0;
                regioniscorrect = false;
            catch
                regioniscorrect = true;
            end
        else
            regioniscorrect = false;
            addtoit = 1;
            totaladded = totaladded + addtoit;
            if(moveright == true)
```

```
                c(1,1) = c(1,1) + addtoit;
                c(1,2) = c(1,2) + addtoit;
                c(1,3) = c(1,3) + addtoit;
                c(1,4) = c(1,4) + addtoit;
            end
            if(moveleft == true)
                c(1,1) = c(1,1) - addtoit;
                c(1,2) = c(1,2) - addtoit;
                c(1,3) = c(1,3) - addtoit;
                c(1,4) = c(1,4) - addtoit;
            end
            if(moveup == true)
                r(1,1) = r(1,1) - addtoit;
                r(1,2) = r(1,2) - addtoit;
                r(1,3) = r(1,3) - addtoit;
                r(1,4) = r(1,4) - addtoit;
            end
            if(movedown == true)
                r(1,1) = r(1,1) + addtoit;
                r(1,2) = r(1,2) + addtoit;
                r(1,3) = r(1,3) + addtoit;
                r(1,4) = r(1,4) + addtoit;
            end
            [BWroi,xi,yi] = roipoly(BW22,c,r);
            BWI2 = BW22;
            BWForComp = BW22;
            BWTRY = BWI2;
            BWI2(BWroi) = 1-BW22(BWroi);
            BWTRY(BWroi) = 1-BW22(BWroi);
            BWTRY(~BWroi) = cast(0,class(BWTRY));
            CC=[];
            CC = bwconncomp(BWTRY);
            STATS = [];
            pixels = [];
            thisBB = [];
            STATS = regionprops(CC,'PixelList','Centroid','BoundingBox');
            thisBB = zeros(length(STATS),4);
            pixels = cell(length(STATS));
            for l = 1:length(STATS)
                thisBB(l,:) = STATS(l).BoundingBox;
                pixels{l} = STATS(l).PixelList;
            end
            if(length(STATS)==1)
                RightComponent = 1;
            end
        end
    else
%Component fits in the region so check that it is close to the standard
%size
        if((thisBB(RightComponent,1) - min(xi)) >= 1 &&...
                (thisBB(RightComponent,2) - min(yi)) >=1 ...
                &&(max(xi)-(thisBB(RightComponent,1)+...
                thisBB(RightComponent,3)))>=1&&(max(yi)-...
                (thisBB(RightComponent,2)+thisBB(RightComponent,4)))>=1)
            thiswidth = thisBB(RightComponent,3);
            heightpercenterror=(abs(thisBB(RightComponent,4) -...
                standheight)/standheight);
```

```
            widthpercenterror = (abs(thisBB(RightComponent,3) -...
                standwidth)/standwidth);
            totalpercenterror = sqrt(heightpercenterror^2+...
                widthpercenterror^2)*100;
            if(heightpercenterror > 50 && widthpercenterror > 50)
                [BWroi,xi,yi] = roipoly(BW22,originalc,originalr);
                BWI2 = BW22;
                BWTRY = BWI2;
                BWI2(BWroi) = 1-BW22(BWroi);
                BWTRY(BWroi) = 1-BW22(BWroi);
                BWTRY(~BWroi) = cast(0,class(BWTRY));
                CC = bwconncomp(BWTRY);
                STATS = regionprops(CC,'PixelList','Centroid',...
                    'BoundingBox');
                modifiedSTATS = STATS;
                modifiedSTATS(OriginalRightComponent,:) = [];
                thisBB = zeros(length(STATS),4);
                pixels = cell(length(STATS));
                cents = zeros(length(STATS),2);
                for k = 1 : length(STATS)
                    pixels{k} = STATS(k).PixelList;
                    thisBB(k,:) = STATS(k).BoundingBox;
                    cents(k,:) = STATS(k).Centroid;
                end;
                modifiedthisBB = thisBB;
                modifiedthisBB(OriginalRightComponent,:) = [];
                if(length(modifiedSTATS)>1)
                    MoreThanOneComponent = true;
                else
                    MoreThanOneComponent = false;
                end
                [RightComponent,TesseractFoundACloseWord,TesseractWord]=...
                    FindCorrectComponent(TesseractFoundWord,WordParam,...
                    modifiedthisBB,MoreThanOneComponent,modifiedSTATS,...
                    standheight,standwidth,QuadYValue,QuadXValue);
                [rows,cols] = find(ismember(...
                    modifiedthisBB(RightComponent,:),'rows'))
                RightComponent = rows;
                CompCount = CompCount + 1;
                OriginalRightComponent(CompCount,:) = RightComponent;
                c = originalc;
                r = originalr;
                expanding = 0;
                regioniscorrect = false;
            else
%If component is close to standard size then check which quadrant it is in
%If the component is located to the left of the dot, then check to see if
%the number is a double digit number
                regioniscorrect=true;
                [morethanoneside,QuadI,QuadII,QuadIII,QuadIV,Right,Left,...
                    Bottom,Top]=CheckQuad(centers,pixels{RightComponent});
                [topx,topy]=CheckForDoubleDigits(TesseractFoundACloseWord...
                    ,WordParam,TesseractWord,QuadII,QuadIII,Left,thisBB,...
                    topx,topy);
            end
        else
%If the component is fully in the region then check which quadrant it is in
```

```
%If the component is located to the left of the dot, then check to see if
%the number is a double digit number
            regioniscorrect=true;
            [morethanoneside,QuadI,QuadII,QuadIII,QuadIV,Right,Left,...
                Bottom,Top] = CheckQuad(centers,pixels{RightComponent});
            [topx,topy]=CheckForDoubleDigits(TesseractFoundACloseWord,...
                WordParam,TesseractWord,QuadII,QuadIII,Left,thisBB,topx,...
                topy);
        end
    end
end

%If no component was found, then use the dots center as the topleft and
%topright region
if(isempty(topx)|| isempty(topy))
    topx = centers(1) - standwidth;
    topy = centers(2) - standheight;
else
end
%Insert each region as a rectangle in image
I322 = insertShape(I3, 'Rectangle', [topx topy standwidth*2.5...
    standheight+5], 'Color', 'Blue','LineWidth',2);

%The function creates a region slightly to the left of the input
%region to determine if there is another digit to the left of the found
%component
    function [topx,topy] = CheckForDoubleDigits(TessFoundCloseWord,...
            TessWords,TessWordCnt,QuadII,QuadIII,Left,thisBB,topx,topy)
        x=[];
        y=[];
        if(QuadII == true || QuadIII == true || Left == true)
            addto = 0;
            doublefitsinregion = false;
            if(TessFoundCloseWord == true)
                x = [TessWords(TessWordCnt,3) TessWords(TessWordCnt,4)...
                    TessWords(TessWordCnt,4) TessWords(TessWordCnt,3)];
                y = [TessWords(TessWordCnt,1) TessWords(TessWordCnt,1)...
                    TessWords(TessWordCnt,2) TessWords(TessWordCnt,2)];
            else
                x = [(thisBB(1,1)-(thisBB(1,3))) (thisBB(1,1)+...
                    thisBB(1,3)) (thisBB(1,1)+thisBB(1,3)) thisBB(1,1)];
                y = [thisBB(1,2) thisBB(1,2) (thisBB(1,2)+thisBB(1,4))...
                    (thisBB(1,2)+thisBB(1,4))];
            end
            while(doublefitsinregion ==false)
                x(1,1) = x(1,1) - addto;
                x(1,4) = x(1,4) - addto;
                y(1,1) = y(1,1) - addto;
                y(1,1) = y(1,1) - addto;
                [BWdd,xi,yi] = roipoly(BW22,x,y);
                BWI2 = BW22;
                BWTRY = BWI2;
                BWI2(BWdd) = 1-BW22(BWdd);
                BWTRY(BWdd) = 1-BW22(BWdd);
                BWTRY(~BWdd) = cast(0,class(BWTRY));
                CC=[];
```

```
                CC = bwconncomp(BWTRY);
                STATS = [];
                pixels = [];
                thisBB = [];
                STATS = regionprops(CC,'PixelList','Centroid',...
                    'BoundingBox');
                thisBB = zeros(length(STATS),4)
                pixels = cell(length(STATS));
                for m = 1:length(STATS)
                    thisBB(m,:) = STATS(m).BoundingBox;
                    pixels{m} = STATS(m).PixelList;
                end
                if(length(STATS) == 2)
                    mostleftcomponent = thisBB(1,1);
                    mostleftcompcount = 1;
                    for n = 1: length(STATS)
                        if(thisBB(n,1)< mostleftcomponent)
                            mostleftcompcount = n;
                        end
                    end
                    if((thisBB(mostleftcompcount,1) - min(xi)) >= 1 &&...
                            (thisBB(mostleftcompcount,2) - min(yi)) >= 1)
                        topx = thisBB(mostleftcompcount,1) - 2;
                        topy = thisBB(mostleftcompcount,2) - 2;
                        doublefitsinregion = true;
                    else
                        addto = addto + 1;
                        doublefitsinregion = false;
                    end
                else
                    topx = topx;
                    topy = topy;
                    doublefitsinregion = true;
                end
            end
        else
            topx = topx;
            topy = topy;
        end
    end

%The function determines which component is most likely the number
    function [CorrectComponent,TesseractFoundWordIsClose,thisword] =...
            FindCorrectComponent(TesseractFoundWord,WordParameter,...
            thisBB,MoreThanOneComponent,STATS,avgheight,avgwidth,...
            QuadYVal,QuadXVal)
        TesseractFoundWordIsClose = false;
        thisword = [];
        possiblewords = [];
%If Tesseract has found any words that it is confident in
        if(TesseractFoundWord == true)
            smalldiffx = 1000;
            smalldiffy = 1000;
            smalldiff = 1000;
            possibles = 1;
%Find if any paramater (x1,x2,y1,y2) of the word is contained within the
```

```
%region
            for play=1:length(WordParameter)
                if((WordParameter(play,3)>=min(xi)&&...
                        WordParameter(play,3)<=max(xi))||...
                        (WordParameter(play,4)>=min(xi)&&...
                        WordParameter(play,4)<=max(xi))...
                        ||(WordParameter(play,1)>=min(yi)&&...
                        WordParameter(play,1)<=max(yi))...
                        || (WordParameter(play,2)>=min(yi)&&...
                        WordParameter(play,2)<=max(yi)))
                            possiblewords(possibles) = play;
                            possibles = possibles +1;
                else
                end
            end
%If a parameter for a word is contained in the region then find the one
%whose center is closest to the center of the dot
            if(isempty(possiblewords))
            else
                QuadYVal = double(QuadYVal);
                QuadXVal = double(QuadXVal);
                mindistan = ((avgheight)^2+((avgwidth)^2))^(1/2);
                for cnting = 1:length(possiblewords)
                    midy = (WordParameter(possiblewords(cnting),1) +...
                        WordParameter(possiblewords(cnting),2))/2;
                    midx = (WordParameter(possiblewords(cnting),3) +...
                    WordParameter(possiblewords(cnting),4))/2;
                    midy = double(midy);
                    midx = double(midx);
                    QuadYVal = double(QuadYVal);
                    QuadXVal = double(QuadXVal);
                    distanc =sqrt(((midy-QuadYVal)^2)+((midx-QuadXVal)^2));
                    if(distanc <= mindistan)
                        mindistan = distanc;
                        thisword = possiblewords(cnting);
                        TesseractFoundWordIsClose = true;

                    else
                    end
                end
            end
        end
        if(MoreThanOneComponent == true)
%If there is a word near the dot (and there are two components)
                %Check to see which component is closest to the word that
                %was found
            if(isempty(thisword)==false)
                if(TesseractFoundWordIsClose == true)
%Check to see which component is closest to the word that was found
                    for cn=1:length(STATS)
                        diffx = abs(thisBB(cn,1)-WordParameter(thisword,3));
                        diffy = abs((thisBB(cn,2)+thisBB(cn,4))-...
                            WordParameter(thisword,2));
                        diffx = double(diffx);
                        diffy = double(diffy);
                        diff = sqrt(diffx^2 + diffy^2);
```

```
                        if(diff<=smalldiff)
                            smalldiffx = diffx;
                            smalldiffy = diffy;
                            smalldiff = diff;
                            realcomponentcount = cn;
                        end
                    end
%Once the closest component is found then figure out which quad the
%component is found in
                    CorrectComponent = realcomponentcount;
%If tesseract did not find a word that was close then determine which
%component is closest to the center of the dot
                else
                    distance = sqrt((cents(1,1)-centers(1))^2 +...
                        (cents(1,2)-centers(2))^2);
                    smallestdistance = distance;
                    correctcomp = 1;
                    for cntin=1:length(STATS)
                        centers(1)
                        centers(2)
                        distance = sqrt((cents(1,1)-centers(1))^2 +...
                            (cents(1,2)-centers(2))^2);
                        if(distance < smallestdistance)
                            CorrectComponent = cntin;
                            smallestdistance = distance;
                        end
                    end
                end
            else
                 distance = sqrt((cents(1,1)-centers(1))^2 +...
                     (cents(1,2)-centers(2))^2);
                 smallestdistance = distance;
                 CorrectComponent = 1;
                 for cntin=1:length(STATS)
                    distance = sqrt((cents(1,1)-centers(1))^2 +...
                        (cents(1,2)-centers(2))^2);
                    if(distance < smallestdistance)
                        CorrectComponent = cntin;
                        smallestdistance = distance;
                    end
                 end
            end
        else
            CorrectComponent = 1;
        end
    end

%The function determines which quadrant the component is located in
    function [morethanoneside,QuadI,QuadII,QuadIII,QuadIV,Right,...
            Left,Bottom,Top] = CheckQuad(centers2,pixels)
        QuadXV = centers2(1);
        QuadYV = centers2(2);
        QuadI = false;
        QuadII = false;
        QuadIII = false;
        QuadIV = false;
```

```
        Right = false;
        Left = false;
        Bottom = false;
        Top = false;
        morethanoneside = false;
        counter = 0;
        greaterthanX =0;
        lessthanX=0;
        lessthanY = 0;
        greaterthanY = 0;
        counter = 0;
        numberxpixels = pixels(:,1);
        numberypixels = pixels(:,2);
        PixelSize  = size(numberxpixels);
        for count=1:PixelSize(1)
            if(numberxpixels(count)> QuadXV)
               greaterthanX = greaterthanX +1;
            elseif(numberxpixels(count) < QuadXV)
               lessthanX = lessthanX + 1;
            end
            if(numberypixels(count) > QuadYV)
               greaterthanY = greaterthanY + 1;
            elseif(numberypixels(count) < QuadYV)
               lessthanY = lessthanY + 1;
            end
        end
        %%Quadrant I
        %pixel>X, pixel<Y
        if(greaterthanX > 0 && lessthanY >0)
            if(lessthanX ==0 && greaterthanY == 0)
                disp('Quad I');
                QuadI = true;
                counter = counter +1;
            elseif(lessthanX > 0 && greaterthanY == 0)
                disp('Top');
                Top = true;
                counter = counter +1;
            elseif(greaterthanY > 0 && lessthanX == 0)
                disp('Right Side');
                Right = true;
                counter = counter +1;
            end
        end
        %%Quadrant II
        %pixel<X, pixel<Y
        if(lessthanX > 0 && lessthanY > 0)
            if(greaterthanX == 0 && greaterthanY ==0)
                disp('Quad II');
                QuadII = true;
                counter = counter +1;
            elseif(greaterthanX > 0 && greaterthanY == 0)
                    disp('Top');
                    Top = true;
                    counter = counter +1;
            elseif(greaterthanY > 0 && greaterthanX == 0)
                    disp('Left Side');
                    Left = true;
```

```
                    counter = counter +1;
            end
        end
        %%Quadrant III
        %pixel<X, pixel>Y
        if(lessthanX > 0 && greaterthanY >0)
            if(greaterthanX == 0 && lessthanY == 0)
                disp('Quad III');
                QuadIII = true;
                counter = counter +1;
            elseif(greaterthanX > 0 && lessthanY == 0)
                disp('Bottom');
                Bottom = true;
                counter = counter +1;
            elseif(lessthanY >0 && greaterthanX == 0)
                disp('Left Side');
                Left = true;
                counter = counter +1;
            end
        end
        %%Quadrant IV
        %pixel>x, pixel >y
        if(greaterthanX >0 &&greaterthanY>0)
            if(lessthanX == 0 && lessthanY == 0)
                disp('Quad IV');
                QuadIV = true;
                counter = counter +1;
            elseif(lessthanX > 0 && lessthanY ==0)
                disp('Bottom');
                Bottom = true;
                counter = counter +1;
            elseif(lessthanY > 0 && lessthanX == 0)
                disp('Right Side');
                Right = true;
                counter = counter +1;
            end
        end
        if(counter > 1)
            morethanoneside= true;
        else
            morethanoneside =false;
        end
    end

%The function determines whether the upper left corner fits within the
%region. If it does not then shift the region left by 1 pixel
    function [topx, topy, fitsinregion, moveup,movedown, moveleft,...
            moveright] = LeftSide(thisBB)
        moveup = false;
        movedown = false;
        moveleft = false;
        moveright = false;
        numofcomp = size(thisBB);
        topx = thisBB(1,1) - 2;
        topy = thisBB(1,2) - 2;
        closestcompnum = 1;
```

```
        %Determine if the entire digit (connected component) is inside of
        %roi
         if((thisBB(closestcompnum,1) - min(xi)) >= 1 &&...
                 (thisBB(closestcompnum,2)-min(yi) >=1))
             topx = thisBB(closestcompnum,1) - 2;
             topy = thisBB(closestcompnum,2) - 2;
             fitsinregion = true;
         else
             moveup = false;
             movedown = false;
             moveleft =  false;
             moveright = false;
             fitsinregion = false;
             if((thisBB(closestcompnum,1) - min(xi)) < 1)
                 moveleft = true;
             end
             if( (thisBB(closestcompnum,2)-min(yi) < 1))
                 moveup= true;
             end
         end
    end

%The function determines whether the upper left corner fits within the
%region. If it does not then shift the region right by 1 pixel
    function [topx, topy, fitsinregion, moveup,movedown, moveleft,...
            moveright] = RightSide(thisBB)
        moveup = false;
        movedown = false;
        moveleft = false;
        moveright = false;
        numofcomp = size(thisBB);
        topx = thisBB(1,1) - 2;
        topy = thisBB(1,2) - 2;
        closestcompnum = 1;
        if((thisBB(closestcompnum,2) - min(yi)) >= 1 && (max(xi)-...
                (thisBB(closestcompnum,1)+thisBB(closestcompnum,3))) >=1)
            topx = thisBB(closestcompnum,1) - 2;
            topy = thisBB(closestcompnum,2) - 2;
            fitsinregion = true;
        else
            moveup = false;
            movedown = false;
            moveleft = false;
            moveright = false;
            fitsinregion = false;
            if((thisBB(closestcompnum,2) - min(yi)) <1)
                moveup = true;
            end
            if((max(xi)-(thisBB(closestcompnum,1)+...
                    thisBB(closestcompnum,3)))<1)
                moveright = true;
            end
        end
    end

%The function determines whether the upper left corner and right side fits
```

```
%within the region. If it does not then shift the region right and up by
%1 pixel
    function [topx, topy, fitsinregion, moveup,movedown, moveleft,...
            moveright] = QuadISide(thisBB)
        moveup = false;
        movedown = false;
        moveleft = false;
        moveright = false;
        numofcomp = size(thisBB);
        topx = thisBB(1,1) - 2;
        topy = thisBB(1,2) - 2;
        closestcompnum = 1;
        if((thisBB(closestcompnum,2)-min(yi) >= 1) &&(max(xi) -...
                (thisBB(closestcompnum,1)+thisBB(closestcompnum,3)))>=1)
            topx = thisBB(closestcompnum,1) - 2;
            topy = thisBB(closestcompnum,2) - 2;
            fitsinregion = true;
        else
            moveup = false;
            movedown = false;
            moveleft = false;
            moveright= false;
            fitsinregion = false;
            if((max(xi) - (thisBB(closestcompnum,1)+...
                    thisBB(closestcompnum,3)))<1)
                moveright= true;
            end
            if((thisBB(closestcompnum,2)-min(yi)) < 1)
                moveup = true;
            end
        end
    end

%The function determines whether the upper left corner fits within the
%region. If it does not then shift the region left and up by 1 pixel
    function [topx, topy, fitsinregion, moveup,movedown, moveleft,...
            moveright] = QuadIISide(thisBB)
        moveup = false;
        movedown = false;
        moveleft = false;
        moveright = false;
        numofcomp = size(thisBB);
        topx = thisBB(1,1) - 2;
        topy = thisBB(1,2) - 2;
        closestcompnum = 1;
        %Determine if the entire digit (connected component) is inside of
        %roi
        if((thisBB(closestcompnum,1)-min(xi)) >= 1 &&...
                (thisBB(closestcompnum, 2)- min(yi)) >=1)
            topx = thisBB(closestcompnum,1) - 2;
            topy = thisBB(closestcompnum,2) - 2;
            fitsinregion = true;
        else
            moveup= false;
            moveleft = false;
            moveright = false;
```

```
            movedown = false;
            fitsinregion = false;
            if((thisBB(closestcompnum,1)-min(xi)) < 1)
                moveleft= true;
            end
            if((thisBB(closestcompnum, 2)- min(yi)) <1)
                moveup = true;
            end
        end
    end

%The function determines whether the upper left corner fits within the
%region. If it does not then shift the region left and down by 1 pixel
    function [topx, topy, fitsinregion, moveup,movedown, moveleft,...
            moveright] = QuadIIISide(thisBB)
        moveup = false;
        movedown = false;
        moveleft = false;
        moveright = false;
        numofcomp = size(thisBB);
        topx = thisBB(1,1) - 2;
        topy = thisBB(1,2) - 2;
        closestcompnum = 1;
        %Determine if the entire digit (connected component) is inside of
        %roi
        if(((thisBB(closestcompnum,1) - min(xi)) >= 1)&&...
                  (max(yi)-(thisBB(closestcompnum,2)+...
                  thisBB(closestcompnum,4)))>=1)
             topx = thisBB(closestcompnum,1) - 2;
             topy = thisBB(closestcompnum,2) - 2;
             fitsinregion = true;
        else
            moveup= false;
            moveleft = false;
            moveright = false;
            movedown = false;
            fitsinregion = false;
            if(((thisBB(closestcompnum,1) - min(xi)) < 1))
                moveleft = true;
            end
            if((max(yi)-
(thisBB(closestcompnum,2)+thisBB(closestcompnum,4)))<1)
                movedown= true;
            end
        end
    end

%The function determines whether the upper left corner fits within the
%region. If it does not then shift the region right and down by 1 pixel
    function [topx, topy, fitsinregion, moveup,movedown, moveleft,...
            moveright] = QuadIVSide(thisBB)
        moveup = false;
        movedown = false;
        moveleft = false;
        moveright = false;
        numofcomp = size(thisBB);
```

```
        topx = thisBB(1,1) - 2;
        topy = thisBB(1,2) - 2;
        closestcompnum = 1;
        %Determine if the entire digit (connected component) is inside of
        %roi
        if(((max(xi)-
(thisBB(closestcompnum,1)+thisBB(closestcompnum,3)))>=1)&&...
                 (max(yi)-
(thisBB(closestcompnum,2)+thisBB(closestcompnum,4)))>=1)
             topx = thisBB(closestcompnum,1) - 2;
             topy = thisBB(closestcompnum,2) - 2;
             fitsinregion = true;
        else
             moveup = false;
             movedown = false;
             moveleft = false;
             moveright = false;
             fitsinregion = false;
             if((max(xi)-
(thisBB(closestcompnum,1)+thisBB(closestcompnum,3)))<1)
                moveright= true;
             end
             if((max(yi)-
(thisBB(closestcompnum,2)+thisBB(closestcompnum,4)))<1)
                movedown = true;
             end
        end
    end

%The function determines whether the upper left corner and bottom of number
%fits within the region. If it does not then shift the region down by 1
%pixel
    function [topx, topy, fitsinregion, moveup,movedown, moveleft,...
            moveright] = BelowSide(thisBB)
        moveup = false;
        movedown = false;
        moveleft = false;
        moveright = false;
        numofcomp = size(thisBB)
        topx = thisBB(1,1) - 2;
        topy = thisBB(1,2) - 2;
        closestcompnum = 1;
        %Determine if the entire digit (connected component) is inside of
        %roi
        if(((thisBB(closestcompnum,1) - min(xi)) >= 1)&&...
                 (max(yi)-
(thisBB(closestcompnum,2)+thisBB(closestcompnum,4))>=1))
            topx = thisBB(closestcompnum,1) - 2;
            topy = thisBB(closestcompnum,2) - 2;
            fitsinregion = true;
        else
            moveup = false;
            movedown = false;
            moveleft = false;
            moveright = false;
            fitsinregion = false;
```

```
            if(((thisBB(closestcompnum,1) - min(xi)) < 1))
                moveleft = true;
            end
            if( (max(yi)-
(thisBB(closestcompnum,2)+thisBB(closestcompnum,4))<1))
                movedown = true;
            end
        end
    end

%The function determines whether the upper left corner fits within the
%region. If it does not then shift the region up by 1 pixel
    function [topx, topy, fitsinregion, moveup,movedown, moveleft,...
            moveright] = AboveSide(thisBB)
        moveup = false;
        movedown = false;
        moveleft = false;
        moveright = false;
        numofcomp = size(thisBB);
        topx = thisBB(1,1) - 2;
        topy = thisBB(1,2) - 2;
        closestcompnum = 1;
        %Determine if the entire digit (connected component) is inside of
        %roi
         if((thisBB(closestcompnum,1) - min(xi)) >= 1 &&
(thisBB(closestcompnum,2) - min(yi)) >= 1)
            topx = thisBB(closestcompnum,1) - 2;
            topy = thisBB(closestcompnum,2) - 2;
            fitsinregion = true;
         else
            moveup = false;
            movedown = false;
            moveleft = false;
            moveright = false;
            fitsinregion = false;
            if((thisBB(closestcompnum,1) - min(xi)) < 1)
               moveleft= true;
            end
            if((thisBB(closestcompnum,2) - min(yi)) < 1)
               moveup = true;
            end
         end
    end
end
```