2015

Why do the order of the strokes matter in handwriting?

Lillian Elizabeth Reich
Bard College

Why do the order of the strokes matter in handwriting?

Senior Project Submitted to
The Division of Science, Mathematics, and Computing
of Bard College

by
Lilly Reich

Annandale-on-Hudson, New York
May 2015

Abstract

The order of strokes in handwriting is significant. There have been studies that show that knowledge of this order aids in the interpretation of the graphical display of a symbolic expression. Standard image storage does not take this into account for various reasons. This project explores formats that preserve this crucial information. In particular, I have developed Matlab software to facilitate the use of InkML, which is a graphical format that stores the order of strokes in a drawing.

Contents

Acknowledgements

My deepest recognition goes to Bob Mcgrail for having confidence and faith in me for my time at Bard College. It was a good decision to not drop Data Structures three years ago. This project would not have been possible. I would not have envisioned myself to achieve such heights without your encouragement. Thank you.

I would also like to sincerely thank professor Rebecca Thomas who provided me the tools to find meaning in computation. I would also like to thank Keith O'Hara and Sven Anderson for their instruction.

Lastly, I would like to thank all of my professors and people I've met along the way at Bard for developing my mind.

Dedications

I would like to thank my family for their undying support and unconditional love. You know who you are.
I would also like to thank my stuffed Albert Einstein that sat next to my computer throughout my Senior Project whom I squeezed frequently and spoke to when I was stressed or had a good idea.

Chapter 1: Introduction

Many people do not have legible handwriting and do not realize that a handwriting recognition program might not compensate for the bad handwriting while the user is trying to recreate the order of the strokes of a certain expression. For instance, children who are learning to write their names in school often do not have the most legible handwriting. However, their teacher can usually still understand what they are trying to do. A computer, on the other hand, will not be able to understand a user's input of illegible strokes if it is trying to match it to the correct order of strokes. There is a field known as pen-based computing that allows a computer to recreate the order of strokes from the user's input. My senior project takes pen based computing theories into consideration, but there is not a physical pen involved. Instead, I used a format called InkML that already knows the pen input of the stroke order, and then reads the output into software called Matlab, which allows the user to re-produce the order of the strokes. The user may not be able to match the stroke order perfectly. Interestingly, diverse natures of digital ink and other computing applications raises many obstacles since one might not be able to reach a perfect output. In my project, I explore how stroke order is essential for the user to recreate a symbolic expression from a provided graphic.

Handwriting recognition is most desirable for many digital ink applications in order to convert handwritten input into a machine friendly format that can perform useful computations, such as mathematical recognition of any given symbol. According to researcher Steven Watt, who is the creater of a digital ink format known as InkML, and his advisee Rui Hui cited this as a challenging task, especially for mathematical input since the machine is taking the handwritten and drawing components from the user into

consideration[1]. In terms of mathematical symbols, subscripts and superscripts may be written in different sizes, which can cause the machine to misinterpret the spatial relationships between each mathematical symbol based on Watt's and Hui's inferences.

Many studies have shown that once the user practices the stroke order of a given character, the stroke order becomes second nature for the human. Indeed, every human has their own unique handwriting that might not match that of their peers, but the computer might not necessarily understand the difference between imperfect and perfect handwriting. There have been numerous formats publicized in the pen-based computing field, but they are restricted to singular platforms, which causes results to be less than accurate.

In my project, I use a system called Matlab that widely accepts other technical platforms into its programming environment. For the purpose of solving this problem, I am constructing a Matlab/InkML library that will help render the users' input to match InkML's symbolic expression correctly. Matlab, high powered and reliable software, allows various image processing techniques to be used within its programming environment. Since Matlab is considered to be a fast paced software that is less prone to error, mathematical pen based computing should be able to benefit from Matlab's offerings. Mathematical notations are generally written as two-dimensional figures, but it can be difficult to capture each stroke accurately from a computer's perspective. According to Watt and Hui, pen input of mathematics is three times faster and two times less error-prone than standard keyboard and mouse input. After reviewing this research

[1] Watt,Steven, Rui Hui, *Representation, Recognition, and Collaboration with Digital Ink* (University of Western Ontario, 2013) , 9.

paper several times, I came to the conclusion that Matlab may be able to attempt this complicated task of re-storing stroke formation for mathematical pen based computing.

As I began my project, I had to consider how data is typically represented for handwriting recognition. I knew that pixels generally have an x and y-axis, which could suggest that characters may follow the same coordinate plane in real time. I wanted to be able to re-draw strokes in real time and receive the most accurate results in real time since this is what I had in mind for the end goal. Watt notes, "in order to accommodate detailed analysis and high definition rendering, higher sampling rates are used, allowing more points to be collected within an interval of time"[2]. In other words, there are numerous pixels that are being processed as each stroke is being rendered on the canvas. This raises the question as to how can each stroke be accurately displayed as it is being rendered on the canvas, which led to the motivation for my senior project.

Since I would like to have the autonomy of recreating stroke order in Matlab, I need a basis that will help me get started. As I mentioned briefly, I will need a format known as InkML that already contains the stroke order information. The InkML file will help provide guidance for the coordinate information that will be displayed on the canvas in Matlab. In short, Watt uses an algorithm for InkML that selects a subset of points from the original trace of the user. They also noted that the algorithm found an approximation with a specified number of points, which yielded the minimum cumulative error[3] . In more precise terms, the digital ink data is being used to preserve the curves of each mathematical character.

---

[2] Watt, Hui, *Representation, Recognition and Collaboration with Digital Ink*, 13.
[3] Ibid., 18.

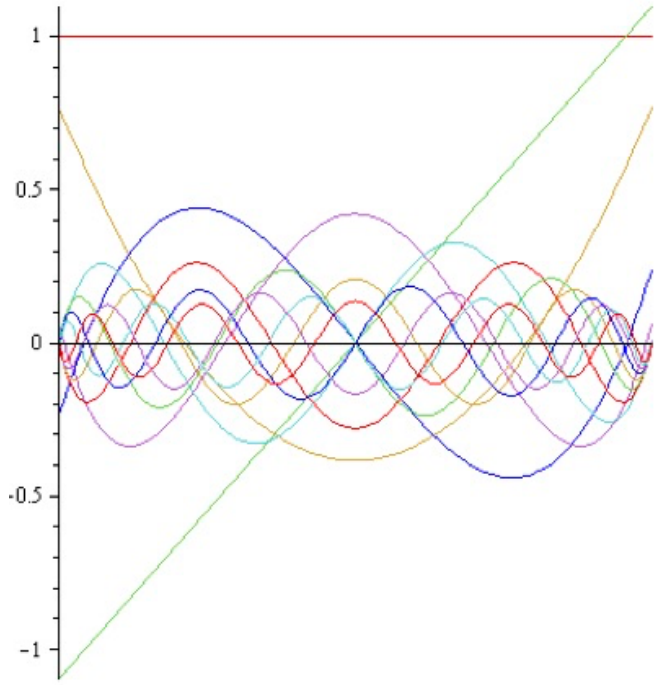
Speaking of mathematical representation, Watt described the difficulty of achieving preciseness of a character. In another paper, titled as *Polynomial Approximation in Handwriting Recognition*, he asserted that one of the main difficulties of other recognition methods is that the digital ink is thought of as a set of points. He described that "re-sampling" as a means of collecting the captured data from the pen is not sufficient enough to remove the device jitter in the final output of the character[4]. Below are displays of Watt's description:

1.1

Single- Stroke Characters

Juxtaposition Ambiguity

[4] Watt,Steven, *Polynomial Approximation in Handwriting Recognition* (University of Western Ontario), 2.



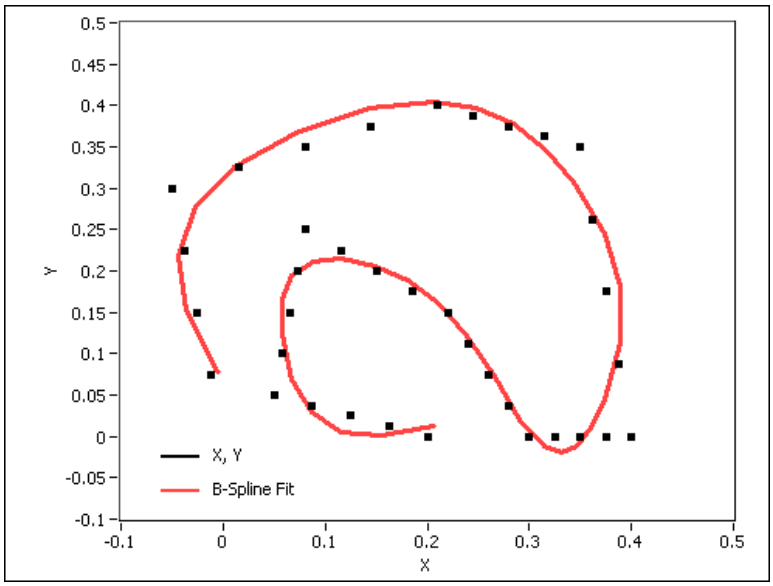
Legendre- Polynomial display for single stroke characters

He used a method called an "orthogonal series" to represent a curve in another paper titled as *Polynomial Approximation in Handwriting Recognition*. In short, he uses two methods called the Chebyshev Polynomial and Legendre Polynomial techniques. A keyword that he also uses is normalization, which means in mathematics to multiply a series by a factor to a desired value, which is usually 1. He uses a Chebyshev Polynomial function $\frac{1}{\sqrt{1-\lambda}}$ for capturing the entire curve of a character, normalizing it, then computing its series coefficients [5]. The Legendre-Polynomial approach allowed the coefficients to be calculated instantly on pen-up movements. Then, Watt used Euclidean distances of inner product spaces of the characters, which was achieved more easily by the orthogonal series calculations.

[5] Watt, Polynomial Approximation in Handwriting Recognition, 2.

Researchers Khoi Nguyen-Tan and Nguyen Nguyen-Hoang, in a paper titled as *Handwriting Recognition Using B-Spline Curve*, conducted another approach for character recognition. Instead, they argued that B- Spline curves are another steady approach to character recognition. In summary, they stated that each character presented as a curve and its' characteristics are described by the control points of a B-Spline curve [6]. A polynomial expression also takes advantage of a B-Spline curve representation since its points are interpolated within a graph. The researchers suggested that a B-Spline would present a continous curve representation. Watt asserted earlier that using orthogonal series would produce a stronger outcome for character recognition. Here is an example of a B-Spline curve approach of a stroke.

1.2



From Watt, Hui and Khoi Nguyen-Tan and Nguyen Nguyen-Hoangs' assertions, I also agreed that a symbol could have multiple strokes, which suggested that the user

[6] Tan – Nguyen, Khoi, Nguyen Nguyen-Hoang, *Handwriting Recognition Using B-Spline Curve* (DaNang University of Technology), 336.

should be able to connect the points into a single curve. However, it was not necessarily that simple. It was important to understand the scale and positioning of each mathematical character in order to move forward with better recognition. In less mathematical terms, Dr. Watt and Hui also used a term called a "determining point" in *Representation, Recognition and Collaboration with Digital Ink*, which meant determining points in a symbol. They mentioned that a certain point occured at different locations in symbols and their precise locations varied in different handwriting samples of the same symbol[7]. In response to this comment, I decided to imagine how each location point of a symbol could be perfected for all handwriting samples. They used another example in which they described that a lowercase "p" would have a baseline located at the lowest part of the bowl and the same would be true for a lowercase "k" that would be identified by the toes[8]. After viewing Dr. Watt, Hui and Drs.Nguyen-Tan and Nguyen-Hoangs' content, I wanted to consider other researchers' attempts at recognizing mathematical characters for pen based computing. After careful thought and consideration, I wanted to be able to see the location points of each symbol occur in real time in Matlab.

In a paper, *Handwriting Recognition Applications for Tablet PCs* written by researchers Dat Tran, Wanli Ma, and Dharmendra Sharma, discussed another approach to mathematical handwriting recognition by use of tablets for PCs. The researchers used fuzzy c-means vector quantization and Markov modeling, which helped detect the noise levels around each character. They took a different path from Watt's research, instead, by converting handwriting to printed text. I was thinking this could be somewhat more

---

[7] Ibid., 7.
[8] Ibid.

difficult since some people may not have the most legible handwriting. In their paper, they described using a tablet PC screen with a tablet pen and a built-in user independent handwriting recognition tool would help convert handwriting into printed text.

The purpose of their project was to better advance ink-enabled applications such as online grading and creating student assignments. In terms of handwriting recognition applications, they proposed applications for teaching crossword puzzles and Sudoku puzzles[9]. The tablet PC displays the crossword and Sudoku puzzles on the screen where the user uses a pen to handwrite letters or digits into a cell[10]. I began to consider how much more legible handwriting would be if people followed these researchers' model. I want to continue doing research on other algorithms that other researchers used for the same topic of mathematical recognition for pen based computing.

In another research paper, titled as *Diagonal Based Feature Extraction for Handwritten Alphabets Recognition System using Neural Network*, three researchers decided to use neural networks for recognizing alphabet letters for pen-based computing. Specifically, they used a method called diagonal based feature extraction, which means segmenting the character into multiple sub-features. Their task included using fifty data sets, each containing twenty- six alphabets written by various people[11]. To digress, I was aware that Matlab was able to perform neural network algorithms, but I did not think that would be necessary for my task. Furthermore, their proposed solution for this project

---

[9] Tran, Dat, Wanli Man, and Dharmendra Sharma, *Handwriting Recognition Applications for Tablet PCs* (University of Canberra), 788.
[10] Ibid.
[11] Pradeep, J., "Diagonal Based Feature Extraction for Handwritten Alphabets Recognition System using Neural Network*", International Journal of Computer Applications* 8, no. 9 (October 2010): 17-22.

was to convert handwritten documents into a text format and recognize handwritten names.

After viewing several papers that studied handwriting recognition, I decided to follow Watt's intuition as a basis for my senior project. His research closely resembled my idea of how I wanted to construct my project. I wanted to use his InkML format for my project for the sake of enhancing handwriting recognition for mathematical pen based computing. My ultimate goal was to construct a Matlab/InkML library that would once again fulfill the task of perfecting stroke order for mathematical pen based computing.

Chapter 2: Image Storage

Image storage in Matlab is essential to the task of recreating the order of the strokes for handwriting recognition since it is one of the most powerful softwares for image processing. One of the greatest resources of Matlab for the purposes of image storage is that it supports a variety of multimedia formats and computational algorithms. Matlab also allows its users to accurately solve problems, produce graphics easily, and produce code efficiently. Its benefits are also highlighted in extensive research, which focuses on Matlab's ability to call external libraries, read in a variety of domain-specific image formats, process both still images and video, and test algorithms immediately. Matlab also has a tool called the Image Processing Toolbox, which allows the user to alter the code to his/her liking. Matlab is also a general purpose programming language, which means that the programmer is able to write function files that allow other users to test and replicate when needed. Therefore, Matlab is user friendly for image processing work.

It is most important to understand the basics behind image storage. A digital image consists of pixels that can be thought of as small dots on the screen that have an x and y-axis. A digital image is technically titled as a "binary image," which is a term that represents all images that can be found on the internet. As a refresher, binary refers to 0s and 1s that are stored as bits in the hardware of the machine. The pixels are separated by RGB values that represent red, green, and blue colors. While there are several different types of image representations that can be used in image processing and in Matlab, I will be mostly concerned with a format called jpeg (2).

There are many variations of images in general, but the most significant image in Matlab is jpeg (2). Here is an example of a jpeg (2) image:

2.1

The reason for its popularity is due to image compression. Image compression means to reduce redundancy of image data in order to be able to store data in an efficient format[11]. The jpeg (2) format also allows a new method of compressing images based off of the wavelet transform. In short, wavelet technology is used to analyze frequency rates of an image[12]. It helps with image compression and filtering, especially for the sake of the jpeg (2) format.

Jpeg (2) format can also be described using the term "lossy compression". Lossy compression is a data encoding method that uses approximations for representing content in an image[13]. The lossy compression would be classified as the backend of the code of

---

[11] Sukanya, Y., and J. Preethi , "Analysis of Image Compression using Wavelet Transform with GUI in Matlab", *International Journal of Research in Engineering and Technology* 2, no. 10 (October 2013): 598.

[12] Galli, A.W., G.T. Heydt, and P.F. Ribeiro. *Exploring the Power of Wavelet Analysis, Computer Applications in Power* 9, no. 4 (October 1996): 38. Accessed April 27, 2015. doi:10.1109/67.539845.

[13] *Modifying zlib and libpng for lossy image compression,* Accessed April 27, 2015. http://membled.com/work/apps/lossy_png/.

that helps illustrate the interface, which is the image. For example, the images of the cat below display how the approximation data of an image becomes progressively coarser as more details of the data that created the original image are removed [14]. There are two images of the cat displayed below that distinguish between high compression and low compression.

Low Compression

High Compression

[14] Zabala, Alaitz, "Effects of JPEG2000 lossy compression on remote sensing image classification for mapping natural areas", *ACM* 9, no. 93 (July 14, 2005): 3.

There are additional advantages and disadvantages to the jpeg (2) format that must be considered as well. The advantages consist of better image quality, lossless compression mode, and smaller file sizes for comparable image quality[15]. The disadvantages suggested for the format are due to the fact that it takes up a demanding amount of memory, consumes higher computation time, and so a certain plug-in is needed to view [16]. There are other image formats that have similar qualities, but are not as powerful as the jpeg (2) format.

The traditional jpeg format consists of the same lossy compression and is used for the World Wide Web. The qualities are similar to the jpeg (2) format, but they are not as complex or as sophisticated. Below is an example of a jpeg format:

2.2

[15] *Digital Image File Types Explained*. www.users.wfu.edu. Accessed April 27, 2015. http://users.wfu.edu//matthews/misc/graphics/formats/formats.html.

[16] Ibid.

The image above is using an algorithm called the DCT transform, which stands for the Discrete Cosine Transform algorithm. The algorithm helps decompress the image by using a matrix of dimension M x N. The M represents the amount of compression needed and the N represents the pixel depth of the image[17]. M and the N are both positive integers, of course.

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}.$$

Here is an example of a jpeg matrix.

[17] *Read image from graphics file*. www.mathworks.com. Accessed April 27, 2015. http://www.mathworks.com/help/matlab/ref/imread.html.

Keep in mind that this matrix was not used for the image displayed above. The matrix with the higher values represents a higher compression rate for the image and the lower values represent a lower compression rate.

In addition to the compression rate, one must understand the advantages and disadvantages of the traditional jpeg format. The advantages of the jpeg format are that it has low complexity, is memory efficient, is applicable for most applications, has reasonable coding efficiency, and can be saved on a sliding resolution scale based on the quality desired. This means the user can change the color scale of the image [18]. The disadvantages suggest that the jpeg format is not great for text because the text may appear fuzzy and unreadable. Additionally, some of the data can get lost while the image is being compressed. From a researcher perspective, the jpeg (2) format seems most reliable for the purpose of my assignment in Matlab because there is higher reliability of not losing data while compressing the image and receiving the most accurate results.

While there are other formats that should also be considered for image processing, they may not be the best for my project and are also not supported for usage in Matlab. Another well-known format called the gif, which stands for Graphics Interchange Format, reduces the number of colors in an image to 256. This essentially means that the image has a white background. They also have the unique ability to display a sequence of images, which is called an animated gif. An animated gif displays a series of separate gif images that are linked together to automatically create animation. Below is an example of a gif image:

---

[18] *Digital Image File Types Explained*. www.users.wfu.edu . Accessed April 27, 2015. http://users.wfu.edu//matthews/misc/graphics/formats/formats.html.

2.3

The image above contains a white background and a moving figure, which is due to the image pixilation process that is taking place within the matrix. Interestingly, the gif format can handle no more than 256 colors, which suggests there is poor compression involved [19]. Furthermore, there is another image format that is compatible to the gif format, which is called the png.

Png stands for portable network graphics. The png has higher compression rate than gifs since they are able to allow 5 to 25 percent greater compression rate. Historically speaking, the png format was an alternative to the gif format when it was copyrighted and required permission to use[20]. While they are not supported for usage in Matlab, the png

---

[19] *Web Graphics*. www.mccauslandcenter.sc.edu. Accessed April 27, 2015. http://www.mccauslandcenter.sc.edu/mricro/obsolete/graphics/graphics.html.

[20] Coffin, Drew, ed. *Image Formats: What's the difference Between JPG, GIF,PNG?* www.practicalcommerce.com. Last modified April 15, 2010. Accessed April 27, 2015. http://www.practicalecommerce.com/articles/1821-Image-Formats-What-s-the-Difference-Between-JPG-GIF-PNG-.

does allow a wider range of colors than traditional gifs, which only allow up to 256 colors. However, the negative aspect of the png format is that not all older browsers are able to display the color scheme of the image. Below is an image in a png format:

2.4

The png image above has similar graphical qualities to the gif format that was mentioned earlier. However, the png is a more sophisticated graphical image than the gif since it can handle more compression and color schemes. Another advantage of the png format, compared to the gif format, is that it supports image interlacing, which means that the user has the ability to view a degraded copy of the entire image. Here is an example below:

As mentioned above, the image contains interlacing, which gif formats cannot handle due to the higher compression rate. If the user decides to use an image that contains many colors, then png is the best format. However, there is another format that is very similar to the png format that can be considered called the bmp file format.

The bmp file format stands for bit map digital images, which means that it is capable of storing 2D images of random width, height, and various color depths[21]. In Matlab, bmp files do not directly run because the files may either be too large or damaged. The user has to be able to save the image to a certain size, for example, such as 110kb instead of 540kb [22]. The advantage of the bmp file is that it is a lossless format, which once again means that the data compression allows the original data to be perfectly reconstructed from the compressed data[23]. The disadvantages of the bmp format suggested that the lossless format does not compress file size, which can result in very

---

[21] *Supported File Formats for Import and Export*. www.mathworks.com. Accessed April 27, 2015. http://www.mathworks.com/help/matlab/import_export/supported-file-formats.html.

[22] *Supported File Formats for Import and Export*. www.mathworks.com. Accessed April 27, 2015.

[23] Ibid.

large images. The png format is the most reliable route, if the user is deciding to compress an image successfully. Below is an image of a bmp file format:

2.5

Because there are a variety of colors in this image, Matlab may have difficulty performing image processing on the file if the image itself is too large and contains too many color schemes. This is not the best format to consider when performing image processing.

The next format that is also used as image storage in Matlab is the tiff format, which stands for Tagged Image Format. The main purpose of the tiff format is for document scanning. The greatest advantage of the tiff format is that it is compatible for

any image processing software, which most certainly includes Matlab. Matlab is able to read this format without any issue by using the t = Tiff('my_subimage_file.tif','r') command [24]. A tiff file can have one or more image file directories. The image file directory contains image data and the metadata tags associated with the image. Additionally, the image file directories usually contain sub image file directories, which contain image data. These subimages have reduced resolution of the image data, which suggests that there are disadvantages to this particular format for image processing. In relation to other image formats, tiff is not the most commonly used since they have large file sizes. Below is a visual example of the tiff file format:

2.6

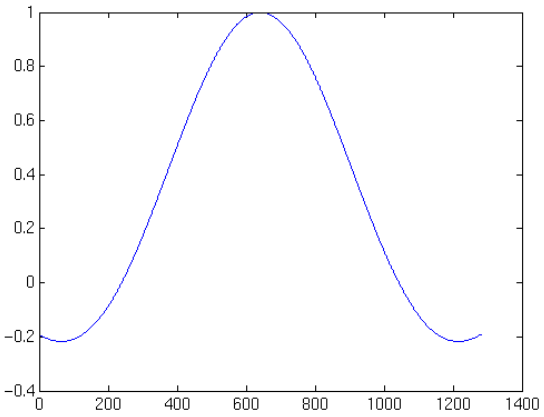
The image above describes a processing technique to play with the image's color scheme, which can be slightly noticed on the zebra if one looks closer at the image.

There is also another format for image processing called the PDF format; however, it does not perform substantial tasks like the other formats that were previously mentioned. The PDF format stands for portable document format, which is typically used to save an image. For instance, the user may save his/her image as a PDF that will link back to their original formatted image. Matlab has the capacity to store PDF images,

[24] Ibid.

but it cannot perform image processing tasks with the format. The biggest advantage of the PDF format for image storage is that it is compact, but not for programming usability. For the sake of image processing, it is best to avoid the PDF option because it is only used to save the image from the original format. Below is a classical PDF example:

2.7



The image above is a sin(x) example of a PDF file, which saved the original image format to the PDF format.

Chapter 3: Why do the order of the strokes matter?

For many individuals, one might question why the ordering of the strokes actually matter in handwriting. While a character can be written in a correct stroke order, it can be misread by the reader due to legibility. In theory, the stroke order gives flow to the character so that can be universally recognized by the reader. Although, individuals have the tendency to follow different stroke orders in the process of writing a character. Stroke order is important for recognition purposes because it can help determine the user's intent. This allows one to see if the user is able to replicate the same figure. For example, in figure 1, there are two pictures that display a symbol.

3.1

Figure 1:

(a) (b)

In picture (a), there is the digitalized version of the phi symbol and in picture (b) there is the discretized version of the phi symbol, which takes the order of the strokes into account. The viewer may immediately assume that the segmented phi appears completely different than picture(a). It can be difficult to detect the corrected phi symbol by means of the ordering of the strokes since typical image storage is not going to work well for the

handwriting dilemma. Luckily, there are computer software systems that help tackle the handwriting recognition issue.

In fact, studies have been completed on why pen-based computing for handwriting recognition is useful for the classroom. A paper titled *The Usability of Handwriting Recognition for Writing in the Primary Classroom*, written by Janet C. Read, Stuart MacFarlane, and Matthew Horton, discuss the three methods of writing that were obtained in a classroom. In short, the three methods they described consisted of the traditional pencil and paper, using a keyboard at the computer, and using a pen and graphics tablet. The authors summarize the three methods and discuss the most effective approach to tackling the problem.

The authors begin by distinguishing between hardcopy recognition and online character recognition of the students. The students in this experiment were ages 7 to 10. After the students wrote on the hardcopy format, their results were transferred by the use of scanning technology and resulted into a bitmap or vector image of the writing[25]. This example displays the traditional use of image storage that will not suffice for my project. It is generally the case that the handwriting on paper will include errors, such that the recognition algorithms that are being used within the scanning software will over look the 'noisy' text[26]. It is true that some characters are badly formed in human handwriting and are therefore not legible for the reader. This also befuddles the recognition software. The authors also discovered that audio formats would place an impact on the student's spelling, since that would likely cause them to make spelling mistakes.

---

[25] Read, C. Janet, Stuart MacFarlane, and Matthew Horton, *The Usability of Handwriting Recognition for Writing in the Primary Classroom* (London: Springer), 136.
[26] Ibid., 137.

In reflection of my handwriting recognition project, I began to consider that a student who is beginning to write his/her name for the first time or writing simple words might experience anxiety since handwriting does not come as second nature. For a novice, it is best to have a visual representation of the task so it can be better stored in the student's memory. The researchers in the article mentioned that audio dictation would cause errors in the students' handwriting, since it is not the best way to introduce something foreign. In addition, I argue that audio formats for beginners can cause other mishaps, such as anxiety.

The researchers decided to try another exercise that would provide the student a sentence on the screen and they would be asked to copy it down onto a sheet of paper. The exercise was purposefully built to help the child make better mental notes. The researchers devised their experiment by using three phrases at a time, believing that the first attempt would cause the character to be poorly recognized. For the second attempt, the student would be able to write the character significantly better. Ultimately, the student would be able to write the character perfectly without apprehension. I would also assume that this exercise would help students' confidence rates in future academic performance, but this was not mentioned in the paper. The researchers mentioned that there was a 'recognition rate' for the third trial, which meant they could find poorly formed letters from the students' handwriting since not all students are able to grasp certain characters immediately[27]. Indeed, there could have been many alternatives to representing the text to the children but this was one method.

---

[27] Ibid.

The second writing task was online and included recognition software that included an online dictionary. The dictionary had a feature that allowed word matching, which could for allow higher recognition rates for the students and reader[28]. In this case, the student would be able to notice the word that appeared most familiar to the word they are searching for in the instructions. The children were given a writing task with the dictionary on and were given twelve minutes to write. They were warned two minutes before they were out of time in case if they wanted to edit their work[29]. There was not a spell-check option enabled in the recognition software, which meant that if a word was misspelled but was accurately recognized, the child would not be able to notice the error [30]. From my perspective, this method allowed the student to discover their intuition more easily as opposed to being given pencil and paper. One could argue that the recognition software option is an easier method to understanding spelling, but that it may inhibit learning the importance of perseverance when acquiring new knowledge for a foreign task.

Many people do not realize that being mindful of the stroke order is essential while writing a character on paper. In a real world example, people who are beginning to learn how to write Chinese characters may consistently use incorrect stroke order but still have legible handwriting. While one may assume that stroke orders would be ingrained into everyone's mind after elementary school, the computer software may disagree due to the fact there has not been software that has been able to perfect the task.

---

[28] Ibid., 138.
[29] Ibid., 139.
[30] Ibid.

Because the computer may not be able to translate the incorrect stroke order into the accurate character, InkML attempts to accomplish this task. InkML takes Matlab to the next level in terms of handwriting correction. Matlab's use of the InkML document helps attempt to solve the issue behind stroke accuracy in computer software, which raises the question of why standard image storage is not going to work for recreating the order of the strokes of an expression in Matlab.

In a paper that was titled as *Character Recognition* by Ziga Zadnik, the student described that optical character recognition (OCR) software can help recognize a variety of fonts, but handwritten fonts are troublesome since it can be difficult for the computer to recognize the order of the strokes[31]. OCR software uses an image against stored bitmaps based on specific fonts in the hardware of the computer[32]. The bitmaps keep track of the hit or miss results of such patterns that help establish OCR's inaccurate results of the original image. Although this student is correct, I decided to construct a Matlab library from scratch that would attempt to solve a similar issue.

---

[31] Zagik, Ziga, *Character Recognition (University of Ljubljana)*, 2.
[32] Ibid.,3.

Chapter 4: What is the purpose of InkML?

The goal is for InkML to be able to record the user's data in order to improve handwriting recognition. It accomplishes this by fulfilling its purpose of transporting "ink" data input with a stylus, or electronic pen, and actively storing the strokes and the direction of each stroke and user information in order to better capture information as the user composed it [33]. In particular, my goal in my project is to establish that InkML needs Matlab in order to render the phi expression correctly. However, Matlab is not the only approach for image processing. The reasons a user may want to use Matlab is that it is flexible software to use and is the most popular software of choice for image processing work.

In specifics, Matlab's basic data is stored in an array, which is inherently built in Matlab's environment. One can also add two arrays together in a singular command as opposed to adding additional for-loops or while-loops, which makes life easier for the programmer. Besides the great features of Matlab, Matlab is not only a programming language but a programming environment as well, which hints that this could be read into other programming environments as well. This means that the programmer can perform operations from the command line of Matlab. Since Matlab works well with graphics, I came into consideration that it may be possible to build an InkML Matlab library. InkML's format works well with MATLAB due to the fact that Matlab is accepting of other formats being read into its software. MATLAB can also be easily incorporated with hardware, since InkML has the capacity to be linked to a stylus in order to create the

[33] Watt, Steven. *Ink Markup Language (InkML)*. www.w3.org. Accessed April 27, 2015.

handwriting strokes. MATLAB is once again useful for image processing because it uses a high-level scripting language that does not require libraries or memory management. MATLAB can easily read an image from a file and display it.

In terms of MATLAB, its programming environment is compatible for extensive prototyping, which suggests there are various ways to collaborate with other implementations such as InkML MATLAB has the capacity to work with image segmentation, signal processing, statistics, and more. Additionally, MATLAB has a native DOM object, which stands for The Document Object Model [34]. In order to understand this better, lets imagine that most of the data on the Internet is stored in some form of XML. Luckily, Matlab has built in functions that handle XML files. Here, the topic concerns a term known as a "Dom Object" in Matlab.

The DOM API allows MATLAB to assemble images, tables, paragraphs, and more, which follows a similar semantic format as XML. This allows results into a specific document. The Document Object Model is analogous to a report document for your machine's memory. The model consists of a hierarchical set of data structures, known as objects that represent the document and its contents. The document also contains children nodes that help represent its contents, which consist of paragraphs, images, tables, and lists. A child node could contain a list of the document's contents such as a table that lists its rows. An example of DOM Object code is listed below:

4.1

```
function theStruct = parseXML(filename) % PARSEXML Convert XML file to
a MATLAB structure. try
```

[34] *Reading XML document and return Document Object Model node.* www.mathworks.com.
Accessed April 27, 2015.

```
tree = xmlread(filename);
catch
error('Failed to read XML file %s.',filename);
end
% Recurse over child nodes. This could run into problems  % with very
deeply nested trees.
try
theStruct = parseChildNodes(tree);
 catch    error('Unable to parse XML file %s.',filename);
end   % ----- Local function PARSECHILDNODES ----- function children =
parseChildNodes(theNode) % Recurse over node children.
children = [];
if theNode.hasChildNodes
childNodes = theNode.getChildNodes;
numChildNodes = childNodes.getLength;
allocCell = cell(1, numChildNodes);
children = struct(             ...       'Name', allocCell,
'Attributes', allocCell,    ...       'Data', allocCell, 'Children',
allocCell);
for count = 1:numChildNodes
 theChild = childNodes.item(count-1);
children(count) = makeStructFromNode(theChild);
end
end  % ----- Local function MAKESTRUCTFROMNODE ----- function
nodeStruct = makeStructFromNode(theNode) % Create structure of node
info.
nodeStruct = struct(                         ...    'Name',
char(theNode.getNodeName),       ...    'Attributes',
parseAttributes(theNode),  ...    'Data', '',
...    'Children', parseChildNodes(theNode));
if any(strcmp(methods(theNode), 'getData'))
nodeStruct.Data = char(theNode.getData);
 else
nodeStruct.Data = '';
 end
% ----- Local function PARSEATTRIBUTES ----- function attributes =
parseAttributes(theNode) % Create attributes structure.
attributes = [];
if theNode.hasAttributes
```

```
theAttributes = theNode.getAttributes;
numAttributes = theAttributes.getLength;
 allocCell = cell(1, numAttributes);
attributes = struct('Name', allocCell, 'Value', ...
allocCell);
for count = 1:numAttributes
attrib = theAttributes.item(count-1);
attributes(count).Name = char(attrib.getName);
attributes(count).Value = char(attrib.getValue);
 end
end
```

Here, the code reads an XML document and returns a Document Object Model node. As one can see, this will produce a Dom object that represents the following elements:

```
<ink xmlns = "http://www.w3.org/2003/InkML">
<traceFormat>
<channel name="X" type="decimal"/>
<channel name="Y" type="decimal"/>
</traceFormat>
<annotation type="truth">$\phi(x)$</annotation>
<annotation type="writer">depart1</annotation>
<annotation type="gender">F</annotation>
<annotation type="age">24</annotation>
<annotation type="hand">R</annotation>
<annotation type="UI">2011_IVC_DEPART_F1_E01</annotation>
<annotationXML type="truth" encoding="Content-MathML">
    <math xmlns='http://www.w3.org/1998/Math/MathML'>
        <mrow>
            <mi xml:id="phi_1">phi</mi>
            <mrow>
                <mo xml:id="(_1">(</mo>
                <mrow>
                    <mi xml:id="x_1">x</mi>
                    <mo xml:id=")_1">)</mo>
                </mrow>
            </mrow>
        </mrow>
```

```
    </math>
</annotationXML>
<trace id="0">
11.7004 15.5288, 11.7004 15.5288, 11.7004 15.5208, 11.7004 15.5047,
11.7004 15.4887, 11.6844 15.4766, 11.6643 15.4686, 11.6322 15.4686,
11.5921 15.4766, 11.5319 15.4967, 11.4878 15.5328, 11.4476 15.585,
11.4316 15.6451, 11.4396 15.7174, 11.4597 15.7695, 11.5118 15.8097,
11.568 15.8097, 11.6603 15.7816, 11.7446 15.7455, 11.8007 15.6853,
11.8248 15.6251, 11.8007 15.5569, 11.7526 15.5127, 11.6804 15.4967,
11.5921 15.5007, 11.5239 15.5368
</trace>
<trace id="1">
11.7847 15.28, 11.7847 15.28, 11.7727 15.272, 11.7606 15.28, 11.7566
15.3081, 11.7365 15.3482, 11.7085 15.4405, 11.6643 15.597, 11.5921
15.7735, 11.5439 15.93, 11.5199 16.0384, 11.5159 16.0905, 11.5159
16.1106, 11.5199 16.1106, 11.5199 16.1026, 11.5319 16.0544
</trace>
<trace id="2">
12.1217 15.3924, 12.1217 15.3924, 12.1137 15.3603, 12.1057 15.3562,
12.0937 15.3603, 12.0616 15.3883, 12.0134 15.4445, 11.9733 15.5408,
11.9412 15.6411, 11.9251 15.7495, 11.9372 15.8377, 11.9532 15.8939,
11.9853 15.934, 12.0054 15.9461, 12.0295 15.9501, 12.0656 15.93
</trace>
<trace id="3">
12.1538 15.7254, 12.1538 15.7254, 12.1498 15.7174, 12.1498 15.7134,
12.1619 15.7093, 12.1699 15.7053, 12.1819 15.6933, 12.21 15.6853,
12.2301 15.6853, 12.2662 15.7013, 12.2983 15.7294, 12.3063 15.7695,
12.2903 15.8177, 12.2582 15.8458, 12.2221 15.8698, 12.206 15.8739,
12.206 15.8658, 12.2261 15.8578, 12.2662 15.8297, 12.3264 15.7735,
12.3665 15.7294, 12.3866 15.7013, 12.3946 15.6893, 12.3906 15.6933,
12.3826 15.6973, 12.3665 15.7214, 12.3384 15.7575, 12.3264 15.8016,
12.3264 15.8377, 12.3424 15.8698, 12.3665 15.8779, 12.4227 15.8779,
12.523 15.8418
</trace>
<trace id="4">
12.5832 15.3442, 12.5832 15.3442, 12.5792 15.3282, 12.5832 15.3282,
12.5992 15.3442, 12.6153 15.3683, 12.6554 15.4365, 12.6915 15.5408,
12.7116 15.6612, 12.7196 15.8056, 12.7076 15.926, 12.6795 15.9942,
12.6434 16.0544, 12.5872 16.0825
```

```
</trace>
<traceGroup xml:id="5">
    <annotation type="truth">From ITF</annotation>
    <traceGroup xml:id="6">
        <annotation type="truth">\phi</annotation>
        <traceView traceDataRef="0"/>
        <traceView traceDataRef="1"/>
        <annotationXML href="phi_1"/>
    </traceGroup>
    <traceGroup xml:id="7">
        <annotation type="truth">(</annotation>
        <traceView traceDataRef="2"/>
        <annotationXML href="(_1"/>
    </traceGroup>
    <traceGroup xml:id="8">
        <annotation type="truth">)</annotation>
        <traceView traceDataRef="4"/>
        <annotationXML href=")_1"/>
    </traceGroup>
    <traceGroup xml:id="9">
        <annotation type="truth">x</annotation>
        <traceView traceDataRef="3"/>
        <annotationXML href="x_1"/>
    </traceGroup>
</traceGroup>
</ink>
```

The XML schema validator was not working, but the code worked well anyway. In Matlab, the parser is tolerant of various XML formats, but the schema validator was poorly programmed from the other user's end.

XML Schema Validator Failure:

4.2

I performed extensive research on the schema validator issue, and I came across several convincing reasons for the schema validator failure. I decided to perform my

schema validation by using my terminal and an online validator called CoreFilling because that would be fastest and most obvious. I did hack multiple times within my InkML document and in my terminal, but still did not have any luck. I also replaced the first line in the InkML document with the line <?xml version = "1.0" encoding = "UTF-8">, which did not make a difference. The other important commands I used in my terminal were as follows:

xmllint – noout –schema fle1.m and fle1.inkml

The schema fle1.m is the schema file and and fle1.inkml is the XML file. The –noout command line means that the XML file will print out validation results. The obstacle that I discovered with xmllint is that it may not support the entire XML schema since the file may be too large, which was oddly the case for me. So, I decided to try the online validator option.

I decided to use CoreFilling because I only had to upload my respective files and get results. The CoreFilling validator uses the Apache Xerces parser, which could have been a likely culprit for the validation failure. While conducting my research on the Apache Xecerces parser, I discovered that there was a history of flaws behind this notorious parser. One of the biggest issues in this parser that grabbed my attention was that they do not label Xerces jars to the most updated version, which will cause confusion for many other software systems such as CoreFilling. For example, the Xerces 2.11.0 jar is titled under xercesImpl.jar and not xercesImpl.2.11.0.jar[35]. I was surprised that the developers have not taken the time to resolve the issue, since Xerces has caused much disruption for schema validators. As a result, my files were not to blame.

---

[35] *Dealing with "Xerces hell" in Java/Maven?.www.*StackOverflow.com.Accessed April 27, 2015.

InkML contains an element called trace, which stores the x and y coordinates of the strokes. Matlab is trained by the programmer to understand InkML's trace attribute since it will deliver a graphical user interface based on the stroke data. The stroke data is listed as 0-4 in trace attribute below:

4.3

Figure 2:

```
</annotationXML>
<trace id="0">
11.7004 15.5288, 11.7004 15.5288, 11.7004 15.5208, 11.7004 15.5047,
11.7004 15.4887, 11.6844 15.4766, 11.6643 15.4686, 11.6322 15.4686,
11.5921 15.4766, 11.5319 15.4967, 11.4878 15.5328, 11.4476 15.585,
11.4316 15.6451, 11.4396 15.7174, 11.4597 15.7695, 11.5118 15.8097,
11.568 15.8097, 11.6603 15.7816, 11.7446 15.7455, 11.8007 15.6853,
11.8248 15.6251, 11.8007 15.5569, 11.7526 15.5127, 11.6804 15.4967,
11.5921 15.5007, 11.5239 15.5368
</trace>
<trace id="1">
11.7847 15.28, 11.7847 15.28, 11.7727 15.272, 11.7606 15.28, 11.7566
15.3081, 11.7365 15.3482, 11.7085 15.4405, 11.6643 15.597, 11.5921
15.7735, 11.5439 15.93, 11.5199 16.0384, 11.5159 16.0905, 11.5159
16.1106, 11.5199 16.1106, 11.5199 16.1026, 11.5319 16.0544
</trace>
<trace id="2">
12.1217 15.3924, 12.1217 15.3924, 12.1137 15.3603, 12.1057 15.3562,
12.0937 15.3603, 12.0616 15.3883, 12.0134 15.4445, 11.9733 15.5408,
11.9412 15.6411, 11.9251 15.7495, 11.9372 15.8377, 11.9532 15.8939,
11.9853 15.934, 12.0054 15.9461, 12.0295 15.9501, 12.0656 15.93
</trace>
<trace id="3">
12.1538 15
```

As one can see in Figure 2, the trace data has varied lengths from 0-4 in the InkML document. This means that each stroke in the phi symbol has a different length. One

could also see in figure 2 that the x and y coordinates inside <trace> have around the same floating - point values. These values are key in deciphering the placement of the x and y coordinates that will help be displayed in a function known as "plot."

In order to fully understand the information inside trace id, it is necessary to understand where exactly these floating - point values originally derived from. Once again, the numbers are based off the InkML document, but were guided from an element called <traceFormat> from the original document. The <traceFormat> element describes the format used to encode points within trace elements. In particular, it defines the sequence of channel values that occurs within trace elements. The order of declaration of channels in the <traceFormat> element determines the order of appearance of their values within <trace> elements [36]. The trace attributes provide keen insight into where the strokes are actually stored. From <traceFormat>, there must have been a format that was used to gather the points. Therefore, the attribute called <trace> must have allowed the storage of those x and y coordinates.

A term known as <channel> becomes relevant after discovering the purpose of <traceFormat>. The regular channels are allowed to appear within the <trace> attribute and are then followed by sporadic channels [37]. Indeed, the <traceFormat> allows <channel> to follow an ordered sequence of values to be visually displayed as a stroke, which means that the order of the coordinates in each point of a trace is determined by the order of the <channel> elements in the <traceFormat> [38]. The InkML <trace> section is displayed below:

---

[36] Watt, Steven. *Ink Markup Language (InkML).* www.w3.org. Accessed April 27, 2015.
[37] Ibid.
[38] Ibid.

4.4

```
<ink xmlns = "http://www.w3.org/2003/InkML">
<traceFormat>
<channel name="X" type="decimal"/>
<channel name="Y" type="decimal"/>
</traceFormat>
<annotation type="truth">$\phi(x)$</annotation>
<annotation type="writer">depart1</annotation>
<annotation type="gender">F</annotation>
<annotation type="age">24</annotation>
<annotation type="hand">R</annotation>
<annotation type="UI">2011_IVC_DEPART_F1_E01</annotation>
<annotationXML type="truth" encoding="Content-MathML">
    <math xmlns='http://www.w3.org/1998/Math/MathML'>
        <mrow>
            <mi xml:id="phi_1">phi</mi>
            <mrow>
                <mo xml:id="(_1">(</mo>
                <mrow>
                    <mi xml:id="x_1">x</mi>
                    <mo xml:id=")_1">)</mo>
                </mrow>
            </mrow>
        </mrow>
    </math>
</annotationXML>
<trace id="0">
11.7004 15.5288, 11.7004 15.5288, 11.7004 15.5208, 11.7004 15.5047,
11.7004 15.4887, 11.6844 15.4766, 11.6643 15.4686, 11.6322 15.4686,
11.5921 15.4766, 11.5319 15.4967, 11.4878 15.5328, 11.4476 15.585,
11.4316 15.6451, 11.4396 15.7174, 11.4597 15.7695, 11.5118 15.8097,
11.568 15.8097, 11.6603 15.7816, 11.7446 15.7455, 11.8007 15.6853,
11.8248 15.6251, 11.8007 15.5569, 11.7526 15.5127, 11.6804 15.4967,
11.5921 15.5007, 11.5239 15.5368
</trace>
<trace id="1">
11.7847 15.28, 11.7847 15.28, 11.7727 15.272, 11.7606 15.28, 11.7566
15.3081, 11.7365 15.3482, 11.7085 15.4405, 11.6643 15.597, 11.5921
```

```
15.7735, 11.5439 15.93, 11.5199 16.0384, 11.5159 16.0905, 11.5159
16.1106, 11.5199 16.1106, 11.5199 16.1026, 11.5319 16.0544
</trace>
<trace id="2">
12.1217 15.3924, 12.1217 15.3924, 12.1137 15.3603, 12.1057 15.3562,
12.0937 15.3603, 12.0616 15.3883, 12.0134 15.4445, 11.9733 15.5408,
11.9412 15.6411, 11.9251 15.7495, 11.9372 15.8377, 11.9532 15.8939,
11.9853 15.934, 12.0054 15.9461, 12.0295 15.9501, 12.0656 15.93
</trace>
<trace id="3">
12.1538 15.7254, 12.1538 15.7254, 12.1498 15.7174, 12.1498 15.7134,
12.1619 15.7093, 12.1699 15.7053, 12.1819 15.6933, 12.21 15.6853,
12.2301 15.6853, 12.2662 15.7013, 12.2983 15.7294, 12.3063 15.7695,
12.2903 15.8177, 12.2582 15.8458, 12.2221 15.8698, 12.206 15.8739,
12.206 15.8658, 12.2261 15.8578, 12.2662 15.8297, 12.3264 15.7735,
12.3665 15.7294, 12.3866 15.7013, 12.3946 15.6893, 12.3906 15.6933,
12.3826 15.6973, 12.3665 15.7214, 12.3384 15.7575, 12.3264 15.8016,
12.3264 15.8377, 12.3424 15.8698, 12.3665 15.8779, 12.4227 15.8779,
12.523 15.8418
</trace>
<trace id="4">
12.5832 15.3442, 12.5832 15.3442, 12.5792 15.3282, 12.5832 15.3282,
12.5992 15.3442, 12.6153 15.3683, 12.6554 15.4365, 12.6915 15.5408,
12.7116 15.6612, 12.7196 15.8056, 12.7076 15.926, 12.6795 15.9942,
12.6434 16.0544, 12.5872 16.0825
</trace>
<traceGroup xml:id="5">
    <annotation type="truth">From ITF</annotation>
    <traceGroup xml:id="6">
        <annotation type="truth">\phi</annotation>
        <traceView traceDataRef="0"/>
        <traceView traceDataRef="1"/>
        <annotationXML href="phi_1"/>
    </traceGroup>
    <traceGroup xml:id="7">
        <annotation type="truth">(</annotation>
        <traceView traceDataRef="2"/>
        <annotationXML href="(_1"/>
    </traceGroup>
```

```
    <traceGroup xml:id="8">
        <annotation type="truth">)</annotation>
        <traceView traceDataRef="4"/>
        <annotationXML href=")_1"/>
    </traceGroup>
    <traceGroup xml:id="9">
        <annotation type="truth">x</annotation>
        <traceView traceDataRef="3"/>
        <annotationXML href="x_1"/>
    </traceGroup>
</traceGroup>
</ink>
```

Based on of the InkML document, one could follow that under traceFormat, there are two lines that say `<channel name="X" type="decimal"/>` and `<channel name="Y" type="decimal"/>`. Therefore, these X and Y coordinates are allowed to have a graphical representation.

Before describing the graphical representation of the X and Y coordinates, it is important to understand the idea behind a width channel, which is prevalent to gathering more information about the meaning behind InkML's trace values. According to InkML, there are three channels that provide stroke width[39]. The three channels will help allow optical devices, such as pen based computing, to record measured stroke width and allows applications that generate InkML to specify desired width rendering. Therefore, the programmer can gain a better understanding behind the importance of the array values within the <trace> attribute in the InkML docment.

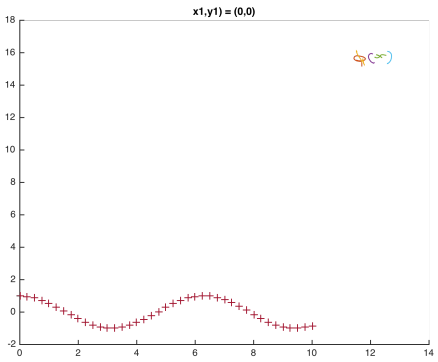
In analyzing lines 38-41, it becomes clear that there are random pairs of x and y coordinates that are used as a trial to see the possible line formation for Matlab. These x and y coordinates help display the dots on the canvases that can be seen in figure 4.

---

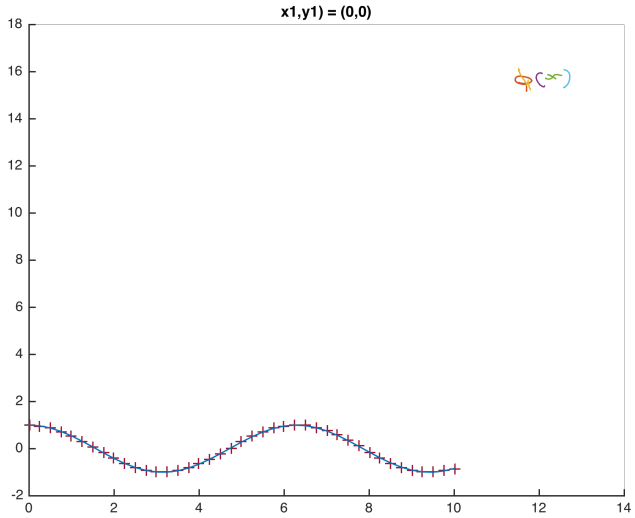
[39] Ibid.

```
x=0:0.25:10;
y=cos(x);
plot(x,y,'+'); % plots x,y as separate points
plot(x,y) %plots a line connecting all the points
```

Figure 4.5



After the dots have been drawn on the canvas, the lines can now connect. On line 41, the line called plot(x,y) helps draw the lines between x and y coordinates.



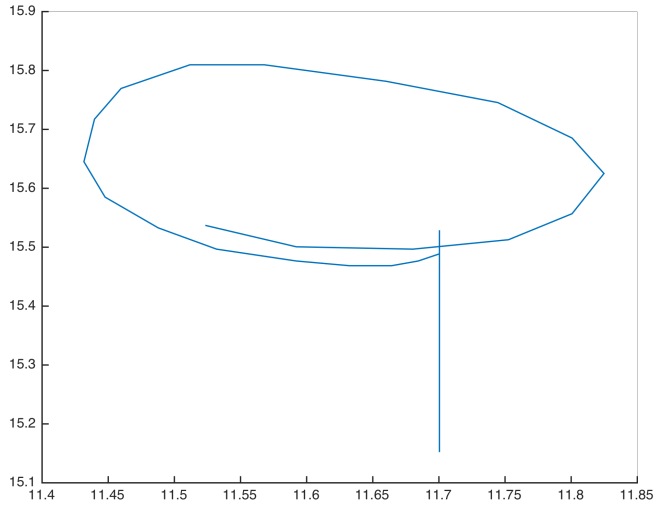
Now, we can assume that Matlab has the potential to record the strokes from the InkML document. Then, Matlab can read the fle1.inkml file into its software and breaks up the order of the strokes more carefully within each trace array.

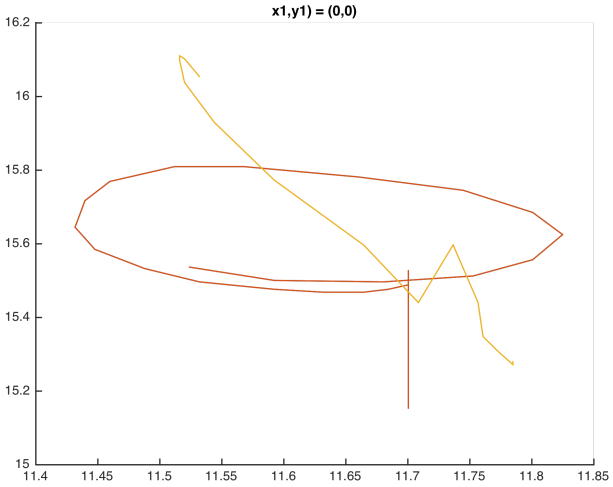
Here is the code that helped start the task:

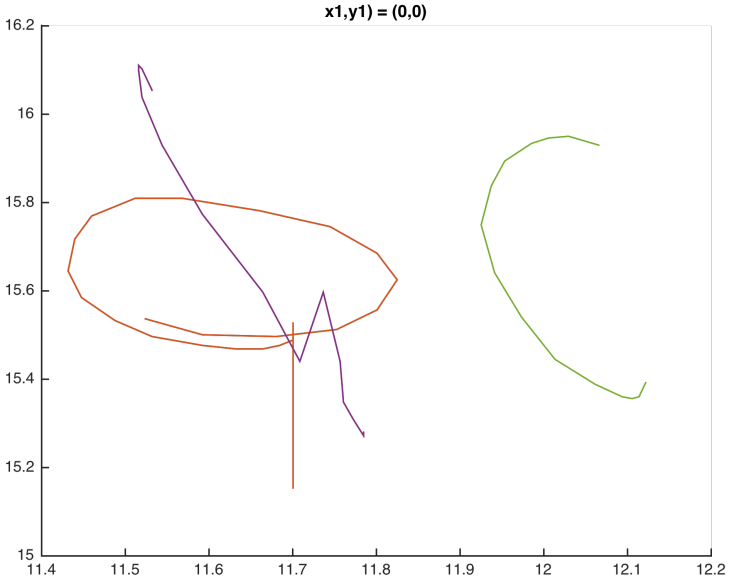
```
figure;
hold on;
x1 = [11.7004 11.7004 11.7004 11.7004 11.6844 11.6643 11.6322 11.5921
11.5319 11.4878 11.4476 11.4316 11.4396 11.4597 11.5118 11.568 11.6603
11.7446 11.8007 11.8248 11.8007 11.7526 11.6804 11.5921 11.5239];
y1 = [15.288 15.5288 15.15208 15.4887 15.4766 15.4686 15.4686 15.4766
15.4967 15.5328 15.585 15.6451 15.7174 15.7695 15.8097 15.8097 15.7816
15.7455 15.6853 15.6251 15.5569 15.5127 15.4967 15.5007 15.5368];
plot(x1,y1);
title(gca, ['x1,y1) = (',num2str(C(1,1)),',',num2str(C(1,2)),')']);
x2 = [11.7847 11.7847 11.7847 11.7727 11.7606 11.7566 11.7365 11.7085
11.6643 11.5921 11.5439 11.5199 11.5159 11.5159 11.5199 11.5319];
y2 = [15.28  15.28 15.272  15.3081 15.3482 15.4405 15.597 15.4405
15.597 15.7735 15.93 16.0384 16.0985 16.1106 16.1026 16.0544];
plot(x2,y2);
x3 = [12.1217 12.1217 12.1137 12.1057 12.0937 12.0616 12.0134 11.9733
11.9412 11.9251 11.9372 11.9532 11.9853 12.0054 12.0295 12.0656];
y3 = [15.3924 15.3924 15.3603 15.3562 15.3603 15.3883 15.445 15.5408
15.6411 15.7495 15.8377 15.8939 15.934 15.9461 15.9501 15.93];
plot(x3,y3);
x4 = [12.1538 12.1538 12.1498 12.1498 12.1619 12.1699 12.1819 12.21
12.2301 12.2662 12.2983 12.3063 12.2903 12.2582 12.221 12.206 12.206
12.2261 12.2662 12.3264 12.3665 12.3866 12.3946 12.3906 12.3826 12.3665
12.3384 12.3264 12.3424 12.3665 12.4227 12.523];
y4 = [15.7254 15.7254 15.7174 15.7134 15.7093 15.7053 15.6933 15.6853
15.6853 15.7013 15.7294 15.7695 15.8177 15.8458 15.8698 15.8739 15.8658
15.8578 15.8297 15.7735 15.7294 15.7013 15.6893 15.6933 15.6973 15.7214
15.7575 15.8016 15.8377 15.8698 15.8779 15.8418];
plot(x4,y4);
x5 = [12.5832 12.5832 12.5792 12.5832 12.5992 12.6153 12.6554 12.6915
12.7116 12.7196 12.7076 12.6795 12.6434 12.5872];
```

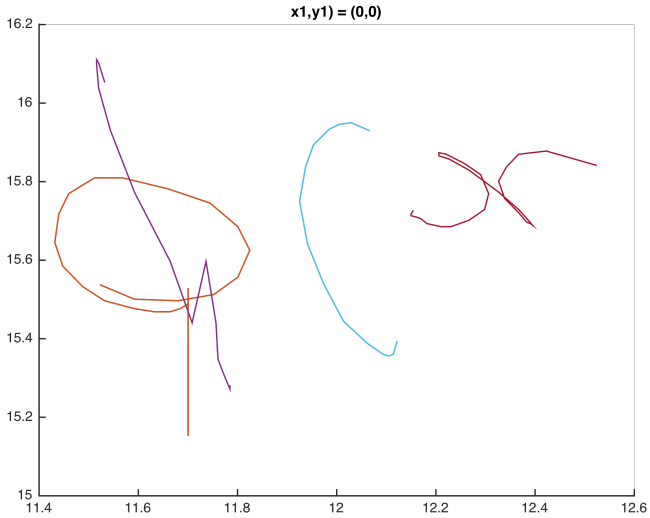
```
y5 = [15.3442 15.3442 15.3282 15.3282 15.3442 15.3683 15.4365 15.5408
15.6612 15.8056 15.926 15.9942 16.0544 16.0825];
plot(x5,y5);
```

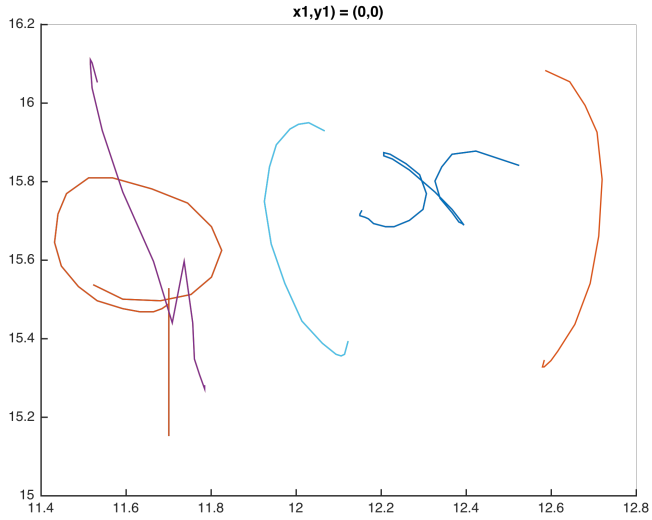
4.6





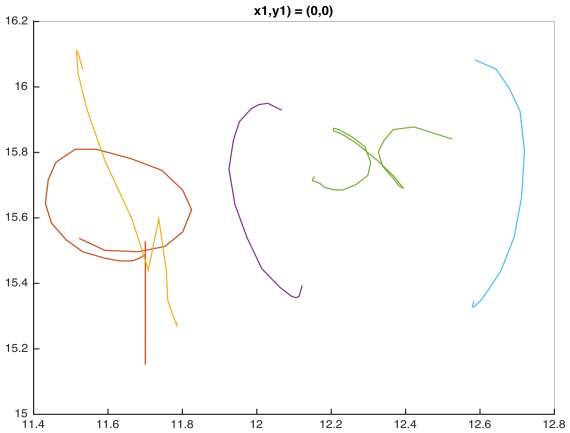

x1,y1) = (0,0)
16.2
16
15.8
15.6
15.4
15.2
15
11.4
11.5
11.6
11.7
11.8
11.9
12
12.1
12.2


x1,y1) = (0,0)
16.2
16
15.8
15.6
15.4
15.2
15
11.4
11.6
11.8
12
12.2
12.4
12.6



Since the <channel> attribute provided a hint into determining where the <trace> values were discovered, we could assume that the x and y coordinates from the InkML document were allowed to have a graphical representation. The function known as “plot” helps plot the points of each x and y coordinate of each trace array onto a function known as “figure”. Once *figure* is called in the Matlab program, a canvas will appear. This is displayed in Figure 3:

4.7



Interestingly, the phi symbol from figure 4.7, is completely segmented in this figure. The combination of all the strokes from the <trace> array from lines 155-174 are combined within this figure. The reasons are such that the cursor has not been placed on any specific <trace> array from lines 155-174.

The reasons that all of the strokes are displayed in figure 4.7 is due to the fact that the cursor has not been placed on any specific plot function call for the <trace> arrays from lines 155-174. The code that performs the designated task is displayed below:

4.8

```
h = plot(x1,y1,'x','ButtonDownFcn', @startDragFcn);
h= plot(x2,y2, ‘x’, ‘ButtonDownFcn’, @startDragFcn);
h =  plot(x3,y3,'x','ButtonDownFcn', @startDragFcn);
```

```
h = plot(x4,y4,'x','ButtonDownFcn', @startDragFcn);
h = plot(x5,y5,'x','ButtonDownFcn', @startDragFcn);
```

A term known as GUI (graphical user interface) becomes applicable for lines 155-174 and now is appropriate to introduce. The purpose of GUI programming in MATLAB is to allow the user to take advantage of the cursor and draw the related stroke in relation to the strokes 0-4 within the <trace> attribute in the InkML document. The user will be able to see if his/her stroke was in the realms of reason within the original trace of the segmented phi symbol. The code remainder of the GUI code is displayed below:

```
traceid0 =[];
traceid1 =[];
traceid2 =[];
traceid3=[];
traceid4 =[];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
f = figure;
aH = axes('Xlim', [0,20], 'Ylim', [0,20]);
x1NEW = []; % intiialize global variable
y1NEW = []; % initialize global variable
x2NEW = [];
y2NEW = [];
x3NEW = [];
y3NEW = [];
x4NEW = [];
y4NEW = [];
x5NEW = [];
y5NEW = [];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
h = plot(x1,y1,'x','ButtonDownFcn', @startDragFcn);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
axis([10 12 14 16])
set(f,'WindowButtonUpFcn', @stopDragFcn);

function startDragFcn(varargin)
set(f, 'WindowButtonMotionFcn', @draggingFcn)
end

function draggingFcn(varargin)
pt = get(aH, 'CurrentPoint');%take mouse point and store into pt
axis([10 12 14 16])% concatenates new pts of the array, empty array
starts to fill up
set(h, 'XData', pt(1,1), 'YData', pt(1,2));% updates mouse point
x1NEW = [x1NEW pt(1,1)];% x location of mouse
y1NEW = [y1NEW pt(1,2)]; %y location of mouse
x2NEW =[x2NEW pt(1,1)];
y2NEW =[y2NEW pt(1,2)];
x3NEW =[x3NEW pt(1,1)];
y3NEW =[y4NEW pt(1,2)];
x4NEW =[x4NEW pt(1,1)];
y4NEW =[y4NEW pt(1,2)];
x5NEW =[x5NEW pt(1,1)];
y5NEW =[y5NEW pt(1,2)];

end

function stopDragFcn(varargin)
set(f, 'WindowButtonMotionFcn', '');
end

plot(x1NEW,y1NEW)
plot(x2NEW, y2NEW)
plot(x3NEW,y3NEW)
plot(x4NEW, y4NEW)
plot(x5NEW,y5NEW)
```

```
axis([0 20 0 20])
```

This helps display the rendering of the strokes.

4.9

A Context-Free Grammar for InkML fle1.inkml

ROOT -> <ink xmlns = "http://www.w3.org/2003/InkML">TRACEFORMAT ANNOTATIONS ANNOTATIONXML TRACES TRACEGROUP</ink>

TRACEFORMAT -> <traceFormat>CHANNELS</traceFormat>

// Channel elements
CHANNELS -> CHANNEL CHANNELSE | CHANNEL
CHANNELSE-> CHANNEL CHANNELSE | CHANNEL | empty-string
CHANNEL -> <channel name="CHANNELNAMES" type="CHANNELTYPES"/>
CHANNELNAMES -> X | Y
CHANNELTYPES -> decimal

// Annotations
ANNOTATIONS -> ANNOTATION ANNOTATIONS | ANNOTATION | empty-string
ANNOTATION -> <annotation type="WORD">WORD</annotation>

// Annotation XML
ANNOTATIONXML -> <annotationXML ANNOTATIONHREF ANNOTATIONTYPE ANNOTATIONCOD>MATH</annotationXML>
ANNOTATIONHREF -> href="URL" | empty-string

ANNOTATIONTYPE -> type="WORD" | empty-string
ANNOTATIONCOD -> encoding="Content-MathML" | empty-string

// Traces
TRACES -> TRACE TRACES | empty-string
TRACE -> <trace id="INTEGER">TRACECONTENT</trace>
TRACECONTENT -> FLOAT FLOAT | FLOAT FLOAT, TRACECONTENTNE | empty-string
TRACECONTENTNE -> FLOAT FLOAT | FLOAT FLOAT, TRACECONTENTNE

// Tracegroup
TRACEGROUP -> <traceGroup xml:id="INTEGER">ANNOTATIONS TRACEGROUP</traceGroup> | <traceGroup xml:id="INTEGER">ANNOTATIONS TRACEVIEWS</traceGroup>
TRACEVIEWS -> <traceView traceDataRef="INTEGER"/> TRACEVIEWS | ANNOTATIONXML

MATH -> ANY VALID MATH EXPRESSION
INTEGER -> ANY INTEGER
FLOAT -> ANY FLOATING POINT NUMBER
WORD -> ANY WORD
URL -

Chapter 5: Matlab library code

In my project, I devised a Matlab library for InkML as the goal of my research. The Matlab library was purposefully built to help render the XML code from the InkML document into the graphical user interface, which allowed the user to create the strokes. Matlab would not face any difficulty solving this task because Matlab is flexible software that is able to store more memory than other image processing environments. I had to study the InkML document carefully and understand each line of XML code needed to help construct my library. By studying the InkML format closely, I was able to understand the data from the <trace> arrays that will help me construct the Matlab library for InkML.

In the beginning, I had to consider how to capture each stroke of a symbolic expression from the InkML arrays. As mentioned in chapter 4, the InkML arrays indicate the points in the coordinate plane. However, I wanted to view those points in real time. First, I had to read the InkML formatted file into Matlab. The code below reads the InkML document and gathers the tag names so the user is able to understand the trace information.

5.1

```
inkmlDoc = xmlread('f1e1.inkml');
allTraces = inkmlDoc.getElementsByTagName('trace');
traceCount = allTraces.getLength;
firstTrace = allTraces.item(1).getTextContent;
```

The code that is displayed above reads the InkML code that is listed below:

```
<ink xmlns = "http://www.w3.org/2003/InkML">
<traceFormat>
```

```
<channel name="X" type="decimal"/>
<channel name="Y" type="decimal"/>
</traceFormat>
<annotation type="truth">$\phi(x)$</annotation>
<annotation type="writer">depart1</annotation>
<annotation type="gender">F</annotation>
<annotation type="age">24</annotation>
<annotation type="hand">R</annotation>
<annotation type="UI">2011_IVC_DEPART_F1_E01</annotation>
<annotationXML type="truth" encoding="Content-MathML">
    <math xmlns='http://www.w3.org/1998/Math/MathML'>
        <mrow>
            <mi xml:id="phi_1">phi</mi>
            <mrow>
                <mo xml:id="(_1">(</mo>
                <mrow>
                    <mi xml:id="x_1">x</mi>
                    <mo xml:id=")_1">)</mo>
                </mrow>
            </mrow>
        </mrow>
    </math>
</annotationXML>
```

After gathering the tag names, I iterated through the traces and grabbed the text from traceText. By necessity, I split the strings by the ‘,’ character and turned the strings into actual integers. In order to view each point, I examined each x and y coordinate. Then, I parsed for each trace and produced a list of points. The code is listed below:

5.2

```
for k = 1:traceCount
thisTrace = allTraces.item(k-1);
%the child is the text of the trace
%char() converts into a matlab string from a java string
traceText = char(thisTrace.getFirstChild.getData);
splitText = textscan(traceText,',');
```

```
%now make a matrix from the trace data to place in traceCell
%each matrix represents a seperate trace
%each matrix has 2 columns, one for x, one for y
% and also n rows, one for each point in the trace
numPoints = length(splitText)-2;
traceMat = zeros(numPoints,2);
for j =1:numPoints
%the first and last elements of splitText are unimportant values
%offset indices for j
pointData = strssplit(char(splitText(j+1)));
xValue = str2double(pointData(2));
yValue = str2double(pointData(3));
traceMat(j,1) = xValue
traceMat(j,2) = yValue
end
traceCell{k} = traceMat
end
```

As one can see, I have a line that says `%xValue = str2double(pointData(2))` and `%yValue = str2double(pointData(3))` that contains different integers. The integers are allowed to be random because I needed to see if I was able to receive any data from these lines of code. Luckily, I did not have to make any alterations to those lines of code. The output I received is displayed below:

```
11.7004
15.5288
11.7004
15.5288
11.7004
15.5208
11.7004
15.5047
11.7004
15.4887
```

11.6844
15.4766
11.6643
15.4686
11.6322
15.4686
11.5921
15.4766
11.5319
15.4967
11.4878
15.5328
11.4476
15.585
11.4316
15.6451
11.4396
15.7174
11.4597
15.7695
11.5118
15.8097
11.568
15.8097
11.6603
15.7816
11.7446
15.7455
11.8007
15.6853
11.8248
15.6251
11.8007
15.5569
11.7526
15.5127
11.6804
15.4967
11.5921

```
15.5007
11.5239
15.5368
```

The output above means that I have gathered all of the points from the first trace and my tag names from the InkML document correctly. For the programmer, it is important to know that data is beginning to appear. Since I was able to view data, I could presume to my next steps. Next, I wanted to be able to visualize all of my points in real time and consider which algorithmic approach would best fit this task.

I used a straightforward rendering algorithm in order to display the fle1.inkml document: I displayed each x and y point from the trace arrays and plotted each point into a Matlab plot function call. I stored each trace array points from fle1.inkml into x1,y1,x2,y2,x3,y3,x4,y4,x5,y5 arrays into Matlab. Then, I was able to move onto the GUI programming that allowed me to render stroke order as represented in the file. Later, I generated new points into the trace arrays from the recreated stroke orders. In all cases this runs in time O(n). In fact it is also Ω(n) since I am plotting each point from the trace arrays onto a canvas.

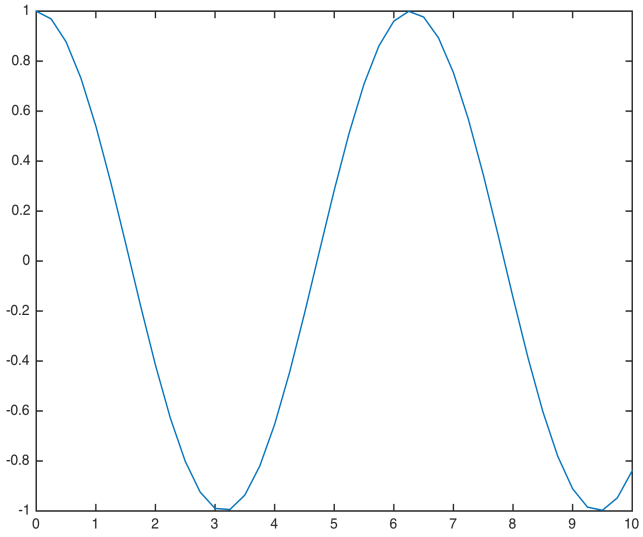
In the InkML file, I have five trace arrays that store all of the points of the x and y-coordinates. Before I begin plotting the points in real time, I needed to see if I was able to display any random x and y coordinates in Matlab. For instance, I made a simple cosine function that was able to plot the points by use of the plot(x,y) function.

```
x =0:0.25:10;
y = cos(x);
plot(x,y, '+');
```

```
plot(x,y)
```

I was able to receive this figure as my output:

5.3



As a result, it was possible for me to see points connected in Matlab, which suggested that I could even see strokes in real time.

After viewing it was possible to see points on the figure in Matlab, I needed to gather each x and y point from the arrays specified in the InkML document. The purpose was to view the x and y points in the prompt of my Matlab editor. The code is displayed below:

5.4

```
disp('List 1:')
str1 = '';
[rows,cols] = size(x1);
for k = 1:cols
    %%temp = str1;
    str1 = strcat(str1, '(', num2str(x1(k)), ' , ' ,
num2str(y1(k)),')');
end
disp(str1)

disp('List 2:')
str2 = '';
[rows,cols] = size(x2);
for k =1:cols
    str2 = strcat(str2, '(', num2str(x2(k)), ' , ', num2str(y2(k)),
')');
end
disp(str2)

disp('List 3:')
str3 = '';
[rows, cols] = size(x3);
for k =1:cols
    str3 = strcat(str3, '(',num2str(x3(k)), ',', num2str(y3(k)), ')');
end
disp(str3)

disp('List 4:')
str4 = '';
[rows,cols] = size(x4);
for k = 1:cols
    str4 =strcat(str4, '(', num2str(x4(k)), ',', num2str(y4(k)), ')');
end
disp(str4)

disp('List 5:')
str5 = ' ';
[rows, cols] = size(x5);
```

```
for k = 1:cols
    str5 = strcat(str5, '(', num2str(x5(k)), ',' , num2str(y5(k)),
')');
end
disp(str5)
%}
```

These display functions allow the user to view the x and y-coordinates as pairs in the prompt. This helps organize the point destination in the coordinate plane. I received output that appears in this fashion:

```
(11.7004,15.5288) (11.7004 ,15.5288) (11.7004 ,15.5208) (11.7004,
15.5047) (11.7004 ,15.4887), (11.6844, 15.4766) (11.6643,15.4686)
(11.6322 15.4686) (11.5921 15.4766) (11.5319 15.4967) (11.4878 15.5328)
(11.4476 15.585) (11.4316 15.6451) (11.4396,15.7174) (11.4597,15.7695)
(11.5118,15.8097) (11.568,15.8097) (11.6603,15.7816) (11.7446, 15.7455)
(11.8007,15.6853), (11.8248,15.6251) (11.8007,15.5569) (11.7526,
15.5127) (11.6804,15.4967) (11.5921, 15.5007) (11.5239,15.5368)


(11.7847, 15.28) (11.7847,15.28) (11.7727, 15.272) (11.7606, 15.28)
(11.7566, 15.3081) (11.7365,15.3482) (11.7085 ,15.4405) (11.6643
15.597) (11.5921 ,15.7735) (11.5439 ,15.93) (11.5199, 16.0384) (11.5159
,16.0905) (11.5159, 16.1106) (11.5199 ,16.1106) (11.5199 ,16.1026)
(11.5319 ,16.0544)


(12.1217 ,15.3924) (12.1217, 15.3924) (12.1137, 15.3603) (12.1057
,15.3562) (12.0937 ,15.3603) (12.0616 ,15.3883) (12.0134, 15.4445)
(11.9733, 15.5408) (11.9412 ,15.6411) (11.9251 ,15.7495) (11.9372,
15.8377) (11.9532 ,15.8939) (11.9853, 15.934) (12.0054, 15.9461)
(12.0295, 15.9501) (12.0656, 15.93)


(12.1538, 15.7254) (12.1538, 15.7254) (12.1498, 15.7174) (12.1498,
15.7134) (12.1619, 15.7093) (12.1699, 15.7053) (12.1819 ,15.6933)
```

```
(12.21, 15.6853) (12.2301, 15.6853) (12.2662, 15.7013) (12.2983,
15.7294) (12.3063, 15.7695) (12.2903, 15.8177) (12.2582, 15.8458)
(12.2221, 15.8698) (12.206, 15.8739) (12.206, 15.8658) (12.2261,
15.8578) (12.2662, 15.8297) (12.3264, 15.7735) (12.3665, 15.7294)
(12.3866, 15.7013) (12.3946, 15.6893) (12.3906, 15.6933) (12.3826
,15.6973) (12.3665, 15.7214) (12.3384 ,15.7575) (12.3264, 15.8016)
(12.3264, 15.8377) (12.3424, 15.8698) (12.3665, 15.8779) (12.4227,
15.8779) (12.523, 15.8418)


(12.5832, 15.3442) (12.5832, 15.3442) (12.5792, 15.3282) (12.5832,
15.3282) (12.5992, 15.3442) (12.6153, 15.3683) (12.6554 15.4365)
(12.6915 ,15.5408) (12.7116, 15.6612) (12.7196, 15.8056) (12.7076,
15.926) (12.6795, 15.9942) (12.6434, 16.0544) (12.5872, 16.0825)
```

After gathering the x and y points into pairs, I was able to move onto the next task to display a symbolic expression into Matlab in real time. I wanted to be able to view the x and y points on the canvas that helps display a symbolic expression. The arrays that helped produce the stroke order are listed below.

5.5

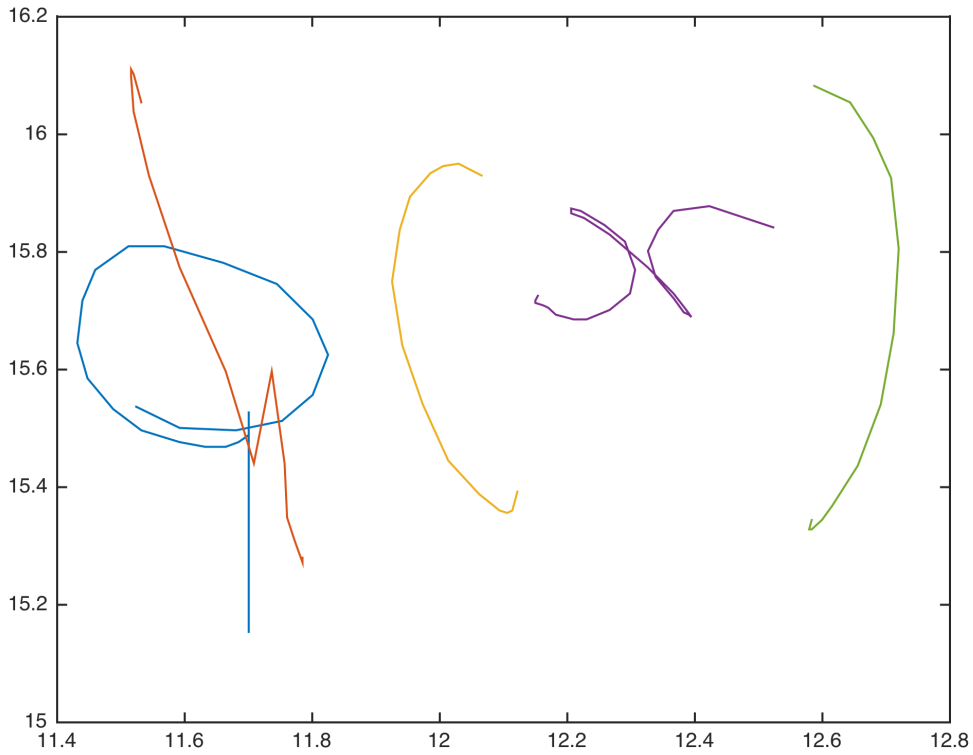
```
x1 = [11.7004 11.7004 11.7004 11.7004 11.6844 11.6643 11.6322 11.5921
11.5319 11.4878 11.4476 11.4316 11.4396 11.4597 11.5118 11.568 11.6603
11.7446 11.8007 11.8248 11.8007 11.7526 11.6804 11.5921 11.5239];
y1 = [15.288 15.5288 15.15208 15.4887 15.4766 15.4686 15.4686 15.4766
15.4967 15.5328 15.585 15.6451 15.7174 15.7695 15.8097 15.8097 15.7816
15.7455 15.6853 15.6251 15.5569 15.5127 15.4967 15.5007 15.5368];
plot(x1,y1);
hold on
% plot(x1,y1,'ButtonDownFcn', @startDragFcn)
% C= get(gcf, 'CurrentPoint'); %I don't know what these 2 lines do
% title(gca, ['x1,y1) = (',num2str(C(1,1)),',',num2str(C(1,2)),')']);
```

```
x2 = [11.7847 11.7847 11.7847 11.7727 11.7606 11.7566 11.7365 11.7085
11.6643 11.5921 11.5439 11.5199 11.5159 11.5159 11.5199 11.5319];
y2 = [15.28  15.28 15.272  15.3081 15.3482 15.4405 15.597 15.4405
15.597 15.7735 15.93 16.0384 16.0985 16.1106 16.1026 16.0544];
plot(x2,y2);
% plot(x2,y2,'ButtonDownFcn',@startDragFcn)
x3 = [12.1217 12.1217 12.1137 12.1057 12.0937 12.0616 12.0134 11.9733
11.9412 11.9251 11.9372 11.9532 11.9853 12.0054 12.0295 12.0656];
y3 = [15.3924 15.3924 15.3603 15.3562 15.3603 15.3883 15.445 15.5408
15.6411 15.7495 15.8377 15.8939 15.934 15.9461 15.9501 15.93];
plot(x3,y3);
% plot(x3,y3,'ButtonDownFcn',@startDragFcn)
x4 = [12.1538 12.1538 12.1498 12.1498 12.1619 12.1699 12.1819 12.21
12.2301 12.2662 12.2983 12.3063 12.2903 12.2582 12.221 12.206 12.206
12.2261 12.2662 12.3264 12.3665 12.3866 12.3946 12.3906 12.3826 12.3665
12.3384 12.3264 12.3424 12.3665 12.4227 12.523];
y4 = [15.7254 15.7254 15.7174 15.7134 15.7093 15.7053 15.6933 15.6853
15.6853 15.7013 15.7294 15.7695 15.8177 15.8458 15.8698 15.8739 15.8658
15.8578 15.8297 15.7735 15.7294 15.7013 15.6893 15.6933 15.6973 15.7214
15.7575 15.8016 15.8377 15.8698 15.8779 15.8418];
plot(x4,y4);
% plot(x4,y4,'ButtonDownFcn',@startDragFcn)
x5 = [12.5832 12.5832 12.5792 12.5832 12.5992 12.6153 12.6554 12.6915
12.7116 12.7196 12.7076 12.6795 12.6434 12.5872];%% go in a clock wise
fashion when drawing vertical line
y5 = [15.3442 15.3442 15.3282 15.3282 15.3442 15.3683 15.4365 15.5408
15.6612 15.8056 15.926 15.9942 16.0544 16.0825];
plot(x5,y5);
```

The arrays x1-y5 originated once again from the InkML document and are read into Matlab. Instead of having a trace id 1-5, I labeled Matlab's arrays as x1 through y5 to more easily identify the stroke order.

Furthermore, the arrays allowed the construction of a symbolic expression to be displayed onto a canvas in Matlab. The *figure* method allowed the arrays to come into fruition for Matlab, which makes life simpler for the programmer. Thus, the programmer

does not have to loop through each pixel to receive a physical display. As one can see, the Φ expression is separated by its stroke order. This is useful for the user so he or she can visualize the x and y-coordinates more easily. Below, a Φ symbol is displayed:



GUI Programming

Since the Φ symbol is now displayed onto a figure plot in Matlab, the next question is how to have the user draw strokes autonomously onto a blank canvas. The term that will be used frequently in this section is known as GUI (graphical user interface) programming, which was mentioned briefly in chapter 4. The GUI was the biggest factor for the handwriting recognition component of my project. I would also note that this section was the most technical and complicated for someone who has not had previous experience coding in Matlab.

The GUI component was the most fascinating aspect of the project because I had to understand how to use the certain functionalities of the GUI library that may not necessarily have been the most intuitive. For instance, I had to test the axis array (see below) with values that are closest to the x and y arrays in the InkML document. At first, I used the values [0 20 0 20], but they were far off from my coordinate points. To resolve this,I changed the values to [10 15 14 16] since they are closest to my x and y values.

After I corrected my arrays, I had to declare my new x and y values as global variables. These variables store the new stroke points and I created an extra folder in my Matlab editor, saving all of the new x and y coordinates as a reference. In the new x and y arrays, I listed <-past tense the point position on the coordinate plane, which detects the location of the mouse. In order to view each plot separately, the user has to click on the side bar of the plot(x1New,y1New) and so forth on functions to receive the blank canvas. Once the user is able to view the blank canvas, they have the autonomy to create any random stroke. However, the purpose is to try to be able to recreate the same strokes of each original plot function. After the user a stroke, they can see their results after

clicking on the “quit-debugging” option in the Matlab editor. The user can then click on the side bar of the plot(x1,y1) etc. functions and see each part of a symbolic expression next to their drawings on each canvas. Below is the code and the output:

5.6

```
 axis([10 12 14 16])%%original to the left just be more accurate on the
arrays x1 y1

set(f,'WindowButtonUpFcn', @stopDragFcn);


Tstart = tic;
function startDragFcn(varargin)

set(f, 'WindowButtonMotionFcn', @draggingFcn)
end

function draggingFcn(varargin)

pt = get(aH, 'CurrentPoint');%take mouse point and store into pt
%axis([0 20 0 20])
axis([10 12 14 16])% concatenates new pts of the array, empty array
starts to fill up
set(h, 'XData', pt(1,1), 'YData', pt(1,2));% updates mouse point
x1NEW = [x1NEW pt(1,1)];% x location of mouse
y1NEW = [y1NEW pt(1,2)]; %y location of mouse
x2NEW =[x2NEW pt(1,1)];
y2NEW =[y2NEW pt(1,2)];
x3NEW =[x3NEW pt(1,1)];
y3NEW =[y4NEW pt(1,2)];
x4NEW =[x4NEW pt(1,1)];
y4NEW =[y4NEW pt(1,2)];
x5NEW =[x5NEW pt(1,1)];
y5NEW =[y5NEW pt(1,2)];
```

```
end



function stopDragFcn(varargin)
set(f, 'WindowButtonMotionFcn', '');
end


plot(x1NEW,y1NEW)
plot(x2NEW, y2NEW)
plot(x3NEW,y3NEW)
plot(x4NEW, y4NEW)
plot(x5NEW,y5NEW)

axis([0 20 0 20])




figure;
hold on;
x1 = [11.7004 11.7004 11.7004 11.7004 11.6844 11.6643 11.6322 11.5921
11.5319 11.4878 11.4476 11.4316 11.4396 11.4597 11.5118 11.568 11.6603
11.7446 11.8007 11.8248 11.8007 11.7526 11.6804 11.5921 11.5239];
y1 = [15.288 15.5288 15.15208 15.4887 15.4766 15.4686 15.4686 15.4766
15.4967 15.5328 15.585 15.6451 15.7174 15.7695 15.8097 15.8097 15.7816
15.7455 15.6853 15.6251 15.5569 15.5127 15.4967 15.5007 15.5368];
plot(x1,y1);
plot(x1,y1,'ButtonDownFcn', @startDragFcn)
C= get(gcf, 'CurrentPoint');
title(gca, ['x1,y1) = (',num2str(C(1,1)),',',num2str(C(1,2)),')']);
x2 = [11.7847 11.7847 11.7847 11.7727 11.7606 11.7566 11.7365 11.7085
11.6643 11.5921 11.5439 11.5199 11.5159 11.5159 11.5199 11.5319];
y2 = [15.28  15.28 15.272  15.3081 15.3482 15.4405 15.597 15.4405
15.597 15.7735 15.93 16.0384 16.0985 16.1106 16.1026 16.0544];
plot(x2,y2);
```

```
plot(x2,y2,'ButtonDownFcn',@startDragFcn)
x3 = [12.1217 12.1217 12.1137 12.1057 12.0937 12.0616 12.0134 11.9733
11.9412 11.9251 11.9372 11.9532 11.9853 12.0054 12.0295 12.0656];
y3 = [15.3924 15.3924 15.3603 15.3562 15.3603 15.3883 15.445 15.5408
15.6411 15.7495 15.8377 15.8939 15.934 15.9461 15.9501 15.93];
plot(x3,y3);
plot(x3,y3,'ButtonDownFcn',@startDragFcn)
x4 = [12.1538 12.1538 12.1498 12.1498 12.1619 12.1699 12.1819 12.21
12.2301 12.2662 12.2983 12.3063 12.2903 12.2582 12.221 12.206 12.206
12.2261 12.2662 12.3264 12.3665 12.3866 12.3946 12.3906 12.3826 12.3665
12.3384 12.3264 12.3424 12.3665 12.4227 12.523];
y4 = [15.7254 15.7254 15.7174 15.7134 15.7093 15.7053 15.6933 15.6853
15.6853 15.7013 15.7294 15.7695 15.8177 15.8458 15.8698 15.8739 15.8658
15.8578 15.8297 15.7735 15.7294 15.7013 15.6893 15.6933 15.6973 15.7214
15.7575 15.8016 15.8377 15.8698 15.8779 15.8418];
plot(x4,y4);
plot(x4,y4,'ButtonDownFcn',@startDragFcn)
x5 = [12.5832 12.5832 12.5792 12.5832 12.5992 12.6153 12.6554 12.6915
12.7116 12.7196 12.7076 12.6795 12.6434 12.5872];
y5 = [15.3442 15.3442 15.3282 15.3282 15.3442 15.3683 15.4365 15.5408
15.6612 15.8056 15.926 15.9942 16.0544 16.0825];
plot(x5,y5);
plot(x5,y5,'ButtonDownFcn',@startDragFcn)
```

I also added new points into the fle1.inkl document since I was able to obtain new points as I drew each stroke. Here are the files:

The first file was to add a new point into the fle1.inkl document.

5.7

```
function append_trace_inkml(x, y, id, filename)

    %
    % function apend_trace_inkml(x, y, id, filename)
    %
    % Takes two vectors of numbers x and y representing a trace
```

```
% and appends the trace (specified by id) in the inkml file
% specified by filename.
%
% x: a vector
% y: a vector, same length as x
% id: numeric, an id of a trace in the inkml file
% infile: string, filename of the inkml file
%

% Open read file and new to write to.
fid = fopen(filename);
filenamew = ['new_', filename];
fidnew = fopen(filenamew,'w');

% get first line.
tline = fgets(fid);

% Iterate through all the lines in the file.
while ischar(tline)

    % If the length of the line is greater than 10 and if
    % the line specifies a trace id ...
    if length(tline) >= 10
        if strcmp(tline(1:10), '<trace id=')
            % disp(tline);
            % Split the line into a cell array and pull out the
            % trace id of the line.
            splitstring = strsplit(tline,'"');
            thisid = str2num(splitstring{2});
            % If this line's trace id matches id...
            if thisid == id

                % Get the next line.
                tline = fgets(fid);

                % Strip off the last character (new line).
                tline = tline(1:end-2);
```

```
                % Then, construct a trace line in inkml format.
                trace_string = construct_trace(x,y);

                % Append the trace string onto the line, along
                % with a new line.
                write_string = [tline, ', ', trace_string, '\n'];

                % Write the appended trace to the new file.
                fprintf(fidnew,write_string);

            else

                % Otherwise, write line to new file.
                fprintf(fidnew,tline);

            end

        else

            % Otherwise, write line to new file.
            fprintf(fidnew,tline);

        end
    else
        % Otherwise, write line to new file.
        fprintf(fidnew,tline);

    end
    % Get the next line for the outer while loop.
    tline = fgets(fid);
    %tline
end
% Close both files.
fclose(fid);
fclose(fidnew);
% If id wasn't matched up in the file, warn the user.
if isnan(x)
```

```
        warningstring = ['trace id = "' mat2str(id) '" not found.'];
        warning(warningstring);
    end
end
```

I was able to append a new point into the trace arrays in the fle1.inkl document, however, I received a minor bug as I ran the application. The bug was not a logic error, but a parsing error that suggested that Matlab was unable to parse to the end tag, which was identified as /p. Instead, I warned the user towards the bottom of the code. The error that appeared in my editor is displayed below :

```
Warning: Control Character '\p is not valid. See 'doc
sprintf' for control characters valid in the format string.
```

In other words, Matlab interprets the end tag differently than InkML and it is solely a format issue.

```
function s = construct_trace(x,y)

    %
    % x = construct_trace(x,y)
    %
    % Given numerical arrays x and y of the same length,
    % construct_trace constucts a string representation of x and y
    % in inkml trace format. construct_trace is the inverse of
    % extract_trace.
    %
    % x: a numerical vector of same length as y
    % y: a numerical vector of same length as x
    %

    L = length(x);
```

```
    s = '';
    for i = 1:L
        if i ~= L
            this_s = [num2str(x(i)), ' ', num2str(y(i)), ', '];
        else
            this_s = [num2str(x(i)), ' ', num2str(y(i))];
        end
        s = [s, this_s];
    end
end


function [x,y] = extract_trace(s)

    %
    % [x,y] = extract_trace(s)
    %
    % Given an string in inkml trace format, extract_trace
    % extracts the x's and y's of the trace and returns them
    % in numerical vectors x and y. extract_trace is the inverse of
    % construct_trace.
    %
    % s: a string in inkml trace format
    %



    % Split the string.
    xycell = strsplit(s,',');

    % From the split string, put all the values into
    % arrays x and y.
    L = length(xycell);
    x = zeros(1,L);
    y = zeros(1,L);
    for i = 1:L
        xy = strsplit(xycell{i});
        if length(xy) == 2
```

```
            x(i) = str2num(xy{1});
            y(i) = str2num(xy{2});
        else
            x(i) = str2num(xy{2});
            y(i) = str2num(xy{3});
        end
    end
end



function [x,y] = get_trace_inkml(id, filename)

    %
    % function [x, y] = get_trace_inkml(id, filename)
    %
    % returns arrays x and y, the traces from an inkml file
    % id: numeric, an id of a trace in the inkml file
    % filename: string, filename of the inkml file
    %

    % Inititialize x and y.
    x = NaN;
    y = NaN;

    % Open file and get first line.
    fid = fopen(filename);
    tline = fgetl(fid);

    % Iterate through all the lines in the file.
    while ischar(tline)
        % If the length of the line is greater than 10 and if
        % the line specifies a trace id ...
        if length(tline) >= 10
            if strcmp(tline(1:10), '<trace id=')
                % disp(tline);
                % Split the line into a cell array and pull out the
                % trace id of the line.
```

```
                splitstring = strsplit(tline,'"');
                thisid = str2num(splitstring{2});
                % If this line's trace id matches id...
                if thisid == id
                    % Get the next line (trace)...
                    tline = fgetl(fid);
                    % ... and extract the x and y values to arrays.
                    [x,y] = extract_trace(tline);
                end
            end
        end
        % Get the next line for the outer while loop.
        tline = fgetl(fid);
    end
    % Close the file.
    fclose(fid);
    % If id wasn't matched up in the file, warn the user.
    if isnan(x)
        warningstring = ['trace id = "' mat2str(id) '" not found.'];
        warning(warningstring);
    end
end



function replace_trace_inkml(x, y, id, filename)

    %
    % function replace_trace_inkml(x, y, id, filename)
    %
    % Takes two vectors of numbers x and y representing a trace
    % and replaces the trace (specified by id) in the inkml file
    % specified by filename.
    %
    % x: a vector
    % y: a vector, same length as x
    % id: numeric, an id of a trace in the inkml file
    % filename: string, filename of the inkml file
```

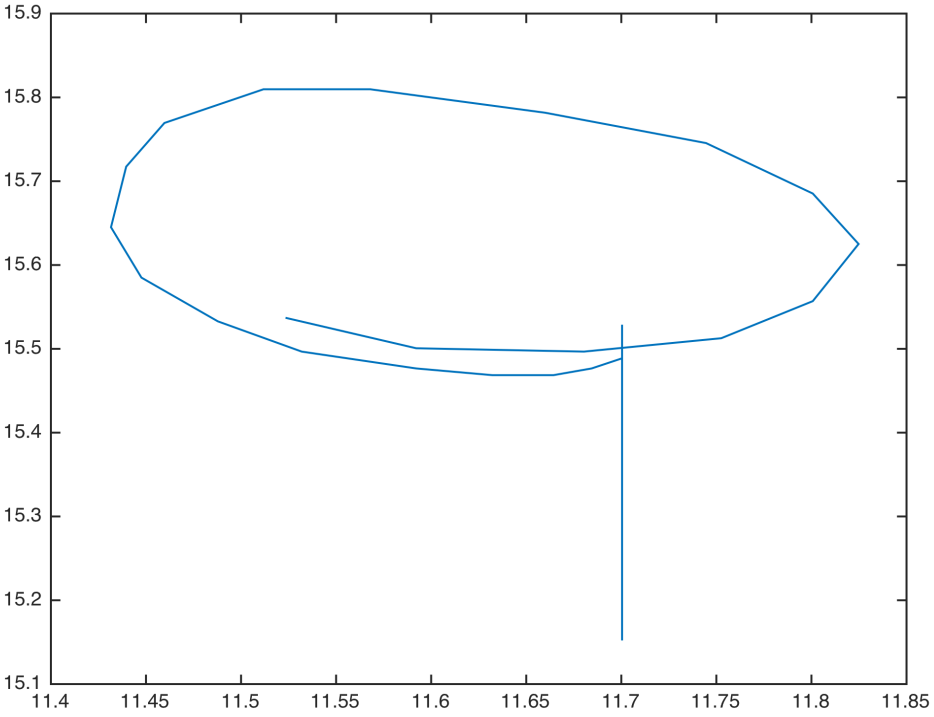
```
%

% Open read file and new to write to.
fid = fopen(filename);
filenamew = ['new_', filename];
fidnew = fopen(filenamew,'w');

% get first line.
tline = fgets(fid);

% Iterate through all the lines in the file.
while ischar(tline)

    % If the length of the line is greater than 10 and if
    % the line specifies a trace id ...
    if length(tline) >= 10
        if strcmp(tline(1:10), '<trace id=')
            % disp(tline);
            % Split the line into a cell array and pull out the
            % trace id of the line.
            splitstring = strsplit(tline,'"');
            thisid = str2num(splitstring{2});
            % If this line's trace id matches id...
            if thisid == id

                % Get the next line, but discard it.
                tline = fgets(fid);

                % Then, construct a trace line in inkml format.
                trace_string = construct_trace(x,y);

                % The trace string to be written needs a carriage
                % return.
                write_string = [trace_string, '\n'];

                % Write the new trace to the new file.
                fprintf(fidnew,write_string);
```

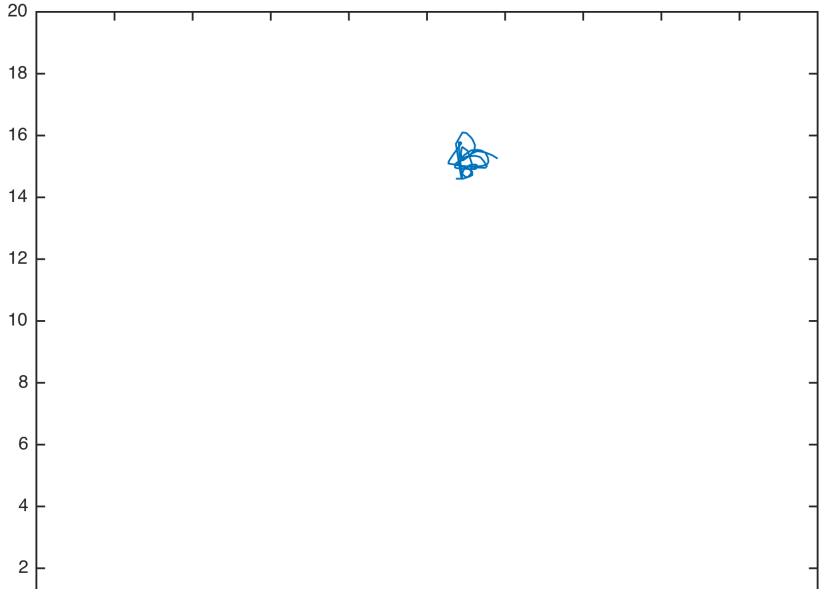
```
                else

                    % Otherwise, write line to new file.
                    fprintf(fidnew,tline);

                end

            else

                % Otherwise, write line to new file.
                fprintf(fidnew,tline);

            end
        else
            % Otherwise, write line to new file.
            fprintf(fidnew,tline);

        end
        % Get the next line for the outer while loop.
        tline = fgets(fid);
        tline
    end
    % Close both files.
    fclose(fid);
    fclose(fidnew);
    % If id wasn't matched up in the file, warn the user.
    if isnan(x)
        warningstring = ['trace id = "' mat2str(id) '" not found.'];
        warning(warningstring);
    end
end
```

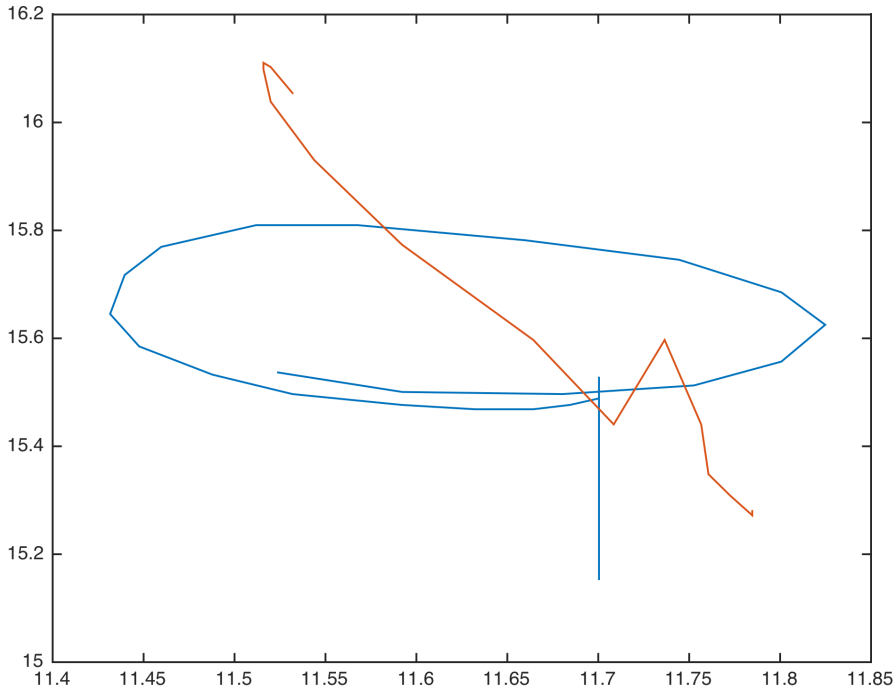
These files helped add the new traces to the fle1.inkml document. These new points derived from the user's strokes. The points were almost exact as the original traces in the fle1.inkml file.

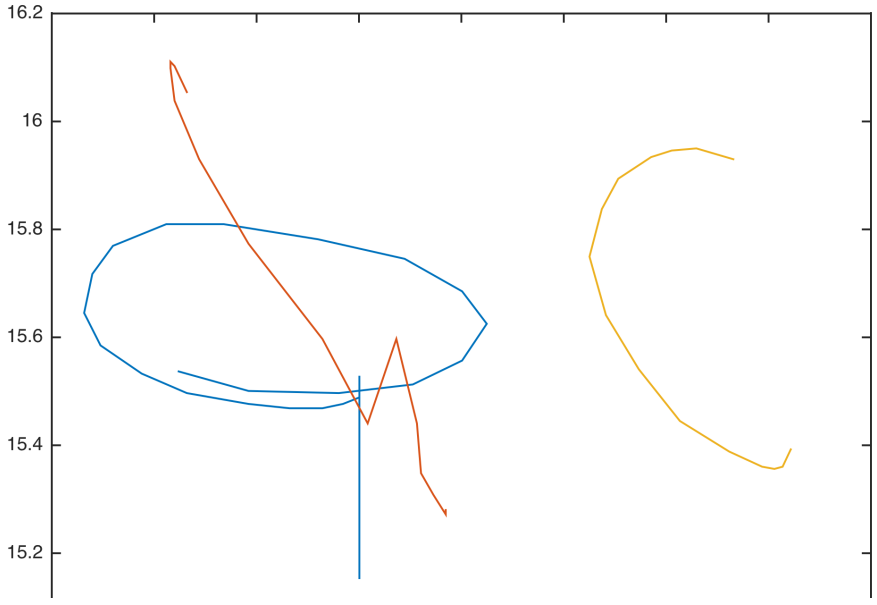
5.8

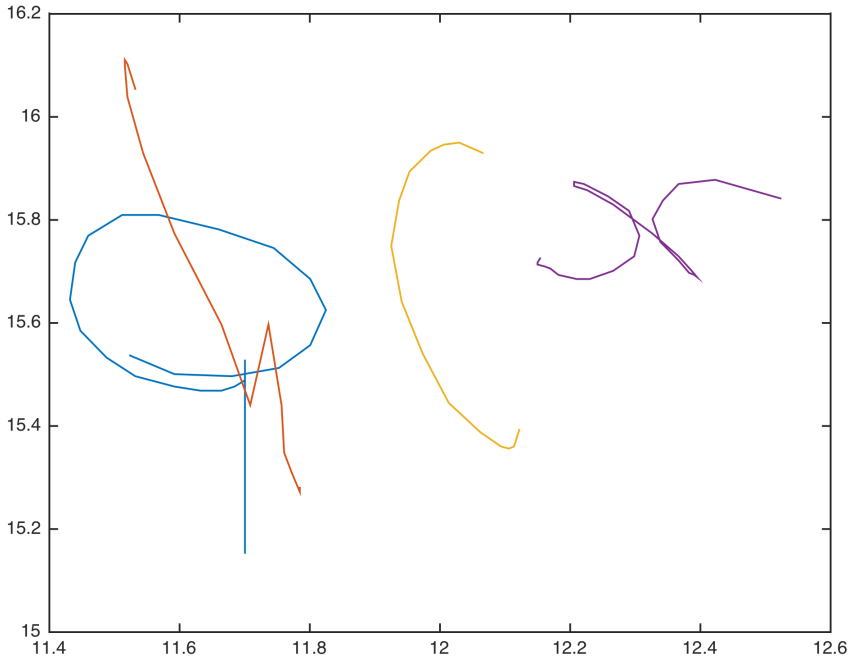
x1 x2 x3 x4 x5/y1 y2 y3 y4 y5 array figures:

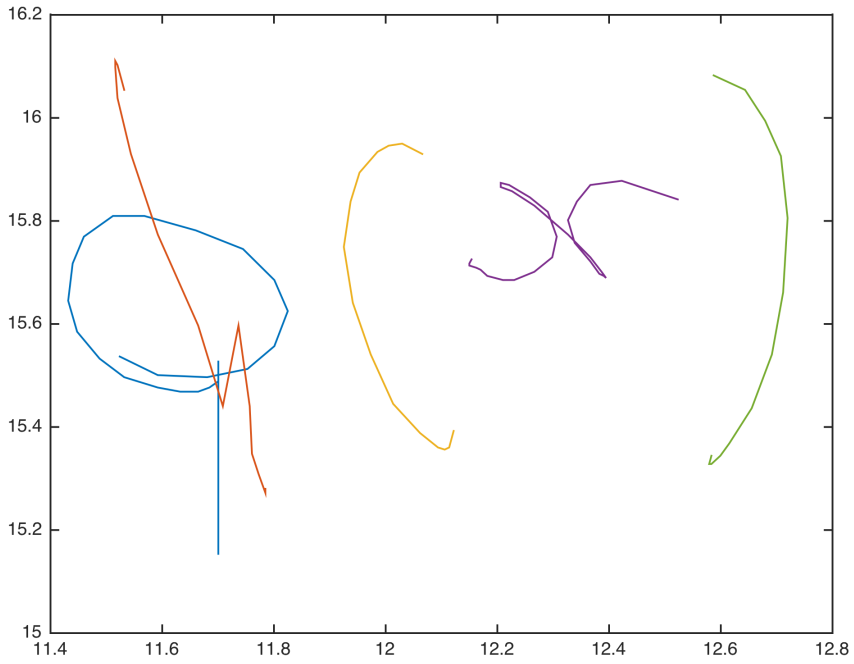





16.2
16
15.8
15.6
15.4
15.2
15
11.4
11.45
11.5
11.55
11.6
11.65
11.7
11.75
11.8
11.85


16.2
16
15.8
15.6
15.4
15.2


16.2
16
15.8
15.6
15.4
15.2
15
11.4
11.6
11.8
12
12.2
12.4
12.6


16.2
16
15.8
15.6
15.4
15.2
15
11.4
11.6
11.8
12
12.2
12.4
12.6
12.8

Finally, I was then able to allow the user to create random strokes. I could test to see if I was able to replicate each stroke from the InkML document. The next step was a slider button that allowed the user to draw each stroke of a Φ expression. I added two more global variables at the top of my program called x1Total and y1Total that stored all of the new points into an array. The code is displayed below:

```
function fle1_Lilly
global x1NEW %make variables global in nature because nested function
or not we can use it
global y1NEW
global x2NEW
global y2NEW
global x3NEW
global y3NEW
global x4NEW
global y4NEW
global x5NEW
global y5NEW
global x1Total
global y1Total

%% reads inkml file
inkmlDoc = xmlread('f1e1.inkml');

allTraces = inkmlDoc.getElementsByTagName('trace');
traceCount = allTraces.getLength;
firstTrace = allTraces.item(1).getTextContent;



traceid0 =[];
traceid1 =[];
traceid2 =[];
traceid3=[];
```

```
traceid4 =[];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
f = figure;
aH = axes('Xlim', [0,20], 'Ylim', [0,20]);
x1NEW = []; % intiialize global variable
y1NEW = []; % initialize global variable
x2NEW = [];
y2NEW = [];
x3NEW = [];
y3NEW = [];
x4NEW = [];
y4NEW = [];
x5NEW = [];
y5NEW = [];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Display the picture first
x1 = [11.7004 11.7004 11.7004 11.7004 11.6844 11.6643 11.6322 11.5921
11.5319 11.4878 11.4476 11.4316 11.4396 11.4597 11.5118 11.568 11.6603
11.7446 11.8007 11.8248 11.8007 11.7526 11.6804 11.5921 11.5239];
y1 = [15.288 15.5288 15.15208 15.4887 15.4766 15.4686 15.4686 15.4766
15.4967 15.5328 15.585 15.6451 15.7174 15.7695 15.8097 15.8097 15.7816
15.7455 15.6853 15.6251 15.5569 15.5127 15.4967 15.5007 15.5368];
plot(x1,y1);
hold on
% plot(x1,y1,'ButtonDownFcn', @startDragFcn)
% C= get(gcf, 'CurrentPoint'); %I don't know what these 2 lines do
% title(gca, ['x1,y1) = (',num2str(C(1,1)),',',num2str(C(1,2)),')']);
x2 = [11.7847 11.7847 11.7847 11.7727 11.7606 11.7566 11.7365 11.7085
11.6643 11.5921 11.5439 11.5199 11.5159 11.5159 11.5199 11.5319];
y2 = [15.28  15.28 15.272  15.3081 15.3482 15.4405 15.597 15.4405
15.597 15.7735 15.93 16.0384 16.0985 16.1106 16.1026 16.0544];
plot(x2,y2);
% plot(x2,y2,'ButtonDownFcn',@startDragFcn)
x3 = [12.1217 12.1217 12.1137 12.1057 12.0937 12.0616 12.0134 11.9733
11.9412 11.9251 11.9372 11.9532 11.9853 12.0054 12.0295 12.0656];
y3 = [15.3924 15.3924 15.3603 15.3562 15.3603 15.3883 15.445 15.5408
15.6411 15.7495 15.8377 15.8939 15.934 15.9461 15.9501 15.93];
plot(x3,y3);
```

```
% plot(x3,y3,'ButtonDownFcn',@startDragFcn)
x4 = [12.1538 12.1538 12.1498 12.1498 12.1619 12.1699 12.1819 12.21
12.2301 12.2662 12.2983 12.3063 12.2903 12.2582 12.221 12.206 12.206
12.2261 12.2662 12.3264 12.3665 12.3866 12.3946 12.3906 12.3826 12.3665
12.3384 12.3264 12.3424 12.3665 12.4227 12.523];
y4 = [15.7254 15.7254 15.7174 15.7134 15.7093 15.7053 15.6933 15.6853
15.6853 15.7013 15.7294 15.7695 15.8177 15.8458 15.8698 15.8739 15.8658
15.8578 15.8297 15.7735 15.7294 15.7013 15.6893 15.6933 15.6973 15.7214
15.7575 15.8016 15.8377 15.8698 15.8779 15.8418];
plot(x4,y4);
% plot(x4,y4,'ButtonDownFcn',@startDragFcn)
x5 = [12.5832 12.5832 12.5792 12.5832 12.5992 12.6153 12.6554 12.6915
12.7116 12.7196 12.7076 12.6795 12.6434 12.5872];%% go in a clock wise
fashion when drawing vertical line
y5 = [15.3442 15.3442 15.3282 15.3282 15.3442 15.3683 15.4365 15.5408
15.6612 15.8056 15.926 15.9942 16.0544 16.0825];
plot(x5,y5);
pause
% plot(x5,y5,'ButtonDownFcn',@startDragFcn)
%{

disp('List 1:')
str1 = '';
[rows,cols] = size(x1);
for k = 1:cols
    %%temp = str1;
    str1 = strcat(str1, '(', num2str(x1(k)), ' , ' ,
num2str(y1(k)),')');
end
disp(str1)

disp('List 2:')
str2 = '';
[rows,cols] = size(x2);
for k =1:cols
    str2 = strcat(str2, '(', num2str(x2(k)), ' , ', num2str(y2(k)),
')');
end
disp(str2)
```

```
disp('List 3:')
str3 = '';
[rows, cols] = size(x3);
for k =1:cols
    str3 = strcat(str3, '(',num2str(x3(k)), ',', num2str(y3(k)), ')');
end
disp(str3)

disp('List 4:')
str4 = '';
[rows,cols] = size(x4);
for k = 1:cols
    str4 =strcat(str4, '(', num2str(x4(k)), ',', num2str(y4(k)), ')');
end
disp(str4)

disp('List 5:')
str5 = ' ';
[rows, cols] = size(x5);
for k = 1:cols
    str5 = strcat(str5, '(', num2str(x5(k)), ',' , num2str(y5(k)),
')');
end
disp(str5)
}



%1st Trace
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x1pt= x1(1,1);%draw a circle
y1pt= y1(1,1);
aH = axes('Xlim', [0,20], 'Ylim', [0,20]);
h = plot(x1pt,y1pt,'x','ButtonDownFcn',@startDragFcn);% defined
%%toc(Tstart)%%testing to see if value Tstart works

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%axis([0 20 0 20])
```

```
axis([10 15 14 16])%%original to the left just be more accurate on the
arrays x1 y1
set(f,'WindowButtonUpFcn', @stopDragFcn);

Tstart = tic;
function startDragFcn(varargin)
set(f, 'WindowButtonMotionFcn', @draggingFcn)
end

function draggingFcn(varargin)
pt = get(aH, 'CurrentPoint');%take mouse point and store into pt
%axis([0 20 0 20])
axis([10 15 14 16])% concatenates new pts of the array, empty array
starts to fill up
set(h, 'XData', pt(1,1), 'YData', pt(1,2));% updates mouse point
x1NEW = [x1NEW pt(1,1)];% x location of mouse
y1NEW = [y1NEW pt(1,2)]; %y location of mouse
end



Tstop = toc(Tstart);%%feeding in results into global variables
%disp(Tstop)
disp(['Elapsed time is...',num2str(Tstop),'!'])
function stopDragFcn(varargin)
set(f, 'WindowButtonMotionFcn', '');
end
pause
clf


%% 2nd trace
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x2pt =x2(1,1);%
y2pt =y2(1,1);
aH = axes('Xlim', [0,20], 'Ylim', [0,20]);
h = plot(x2pt,y2pt,'x','ButtonDownFcn',@startDragFcn2);

axis([10 15 14 16])%%original to the left just be more accurate on the
```

```
arrays x1 y1
set(f,'WindowButtonUpFcn', @stopDragFcn2);

Tstart = tic;
function startDragFcn2(varargin)
set(f, 'WindowButtonMotionFcn', @draggingFcn2)
end

function draggingFcn2(varargin)
pt = get(aH, 'CurrentPoint');%take mouse point and store into pt
%axis([0 20 0 20])
axis([10 15 14 16])
set(h, 'XData', pt(1,1), 'YData', pt(1,2));% updates mouse point
x2NEW = [x2NEW pt(1,1)];% x location of mouse
y2NEW = [y2NEW pt(1,2)]; %y location of mouse
end

%temp = toc; %surpresses whatever line is doing
%Times =cat( 1, Times,temp);


Tstop = toc(Tstart);%%feeding in results into global variables
%disp(Tstop)
disp(['Elapsed time is...',num2str(Tstop),'!'])
function stopDragFcn2(varargin)
set(f, 'WindowButtonMotionFcn', '');
end
pause
clf

%3rd Trace
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x3pt =x3(1,1);%Update these to the right starting point for this trace.
y3pt =y3(1,1);
aH = axes('Xlim', [0,20], 'Ylim', [0,20]);
h = plot(x3pt,y3pt,'x','ButtonDownFcn',@startDragFcn3);

axis([10 15 14 16])%%original to the left just be more accurate on the
arrays x1 y1
```

```
set(f,'WindowButtonUpFcn', @stopDragFcn3);

Tstart = tic;
function startDragFcn3(varargin)
set(f, 'WindowButtonMotionFcn', @draggingFcn3)
end

function draggingFcn3(varargin)
pt = get(aH, 'CurrentPoint');%take mouse point and store into pt
%axis([0 20 0 20])
axis([10 15 14 16])% concatenates new pts of the array, empty array
starts to fill up
set(h, 'XData', pt(1,1), 'YData', pt(1,2));% updates mouse point
x3NEW = [x3NEW pt(1,1)];% x location of mouse
y3NEW = [y3NEW pt(1,2)]; %y location of mouse
end

%temp = toc; %surpresses whatever line is doing
%Times =cat( 1, Times,temp);


Tstop = toc(Tstart);%%feeding in results into global variables
%disp(Tstop)
disp(['Elapsed time is...',num2str(Tstop),'!'])
function stopDragFcn3(varargin)
set(f, 'WindowButtonMotionFcn', '');
end
pause
clf

%4th Trace
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x4pt =x4(1,1);%Update these to the right starting point for this trace.
y4pt =y4(1,1);
aH = axes('Xlim', [0,20], 'Ylim', [0,20]);
h = plot(x4pt,y4pt,'x','ButtonDownFcn',@startDragFcn4);

axis([10 15 14 16])%%original to the left just be more accurate on the
arrays x1 y1
```

```
set(f,'WindowButtonUpFcn', @stopDragFcn4);

Tstart = tic;
function startDragFcn4(varargin)
set(f, 'WindowButtonMotionFcn', @draggingFcn4)
end

function draggingFcn4(varargin)
pt = get(aH, 'CurrentPoint');%take mouse point and store into pt
%axis([0 20 0 20])
axis([10 15 14 16])% concatenates new pts of the array, empty array
starts to fill up
set(h, 'XData', pt(1,1), 'YData', pt(1,2));% updates mouse point
x4NEW = [x4NEW pt(1,1)];% x location of mouse
y4NEW = [y4NEW pt(1,2)]; %y location of mouse
end

%temp = toc; %surpresses whatever line is doing
%Times =cat( 1, Times,temp);


Tstop = toc(Tstart);%%feeding in results into global variables
%disp(Tstop)
disp(['Elapsed time is...',num2str(Tstop),'!'])
function stopDragFcn4(varargin)
set(f, 'WindowButtonMotionFcn', '');
end
pause
clf

%5th Trace
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x5pt =x5(1,1); %Update these to the right starting point for this
trace.
y5pt =y5(1,1);
aH = axes('Xlim', [0,20], 'Ylim', [0,20]);
h = plot(x5pt,y5pt,'x','ButtonDownFcn',@startDragFcn5);

axis([10 15 14 16])%%original to the left just be more accurate on the
```

```
arrays x1 y1
set(f,'WindowButtonUpFcn', @stopDragFcn5);

Tstart = tic;
function startDragFcn5(varargin)
set(f, 'WindowButtonMotionFcn', @draggingFcn5)
end

function draggingFcn5(varargin)
pt = get(aH, 'CurrentPoint');%take mouse point and store into pt
%axis([0 20 0 20])
axis([10 15 14 16])% concatenates new pts of the array, empty array
starts to fill up
set(h, 'XData', pt(1,1), 'YData', pt(1,2));% updates mouse point
x5NEW = [x5NEW pt(1,1)];% x location of mouse
y5NEW = [y5NEW pt(1,2)]; %y location of mouse
end


Tstop = toc(Tstart);%%feeding in results into global variables
%disp(Tstop)
disp(['Elapsed time is...',num2str(Tstop),'!'])
function stopDragFcn5(varargin)
set(f, 'WindowButtonMotionFcn', '');
end
pause
clf


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%tic
plot(x1NEW,y1NEW)
hold on %saves the previos plot and plots new
%toc %% temp storing number that clock returns asks clock what it's
doing and returns number total amount of time toc was executed
plot(x2NEW, y2NEW)
plot(x3NEW,y3NEW)
plot(x4NEW, y4NEW)
plot(x5NEW,y5NEW)
```

```
axis([10 15 14 16])
pause
clf
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x1Total = [x1NEW x2NEW x3NEW x4NEW x5NEW];
y1Total = [y1NEW y2NEW y3NEW y4NEW y5NEW];
hplot = plot(x1Total,y1Total);
axis([10 15 14 16])
hSlider = uicontrol('Style', 'slider',...
        'Min',1,'Max',size(x1Total,2),...
        'SliderStep', [1 1], 'Value',1,...
        'Position', [10 10 420 20],...
        'Callback', @slider1_Callback);

function slider1_Callback(source,callbackdata)
    a = get(source,'Value');
    plot(x1Total(1:a),y1Total(1:a))
    axis([10 15 14 16])
end
pause

end
```

In the end, I was able to finally view a replication of a symbolic expression from the user's end. However, it was not perfect because everyone has their own unique handwriting.

Chapter 6: Conclusion

In reflection of my Senior Project, my goal was to create a Matlab/InkML library that would replicate a symbolic expression based using an InkML document. My task required me to take a digitized symbolic expression and replicate the symbol into handwriting, graphically speaking. It was imperative for me to understand why basic image storage would not be sufficient for recognition purposes. By necessity, I needed to study the trace arrays in the InkML documents to understand where the points originally derived. By studying the arrays closely, I could pursue the construction of the graphical user interface of the expression. I had to consider the stroke order of each trace, which brought intuition into the values behind the x and y coordinates from the InkML document. The key term that was discussed in one of my earlier chapters was called a determining point, in which Steven Watt used to describe a position of all the points in the individual's handwriting. I discovered that this task is difficult since human handwriting is indeed unique. After studying the values, I had to think of an algorithm that would best fit the approach to tackling this problem.

In regards to future work, my project could be expanded to a further degree. Since I was able to accomplish constructing a Matlab /InkML library, there were still imperfections that came into my final result. Indeed, everyone has his or her own unique handwriting, but technology should have the ability to perfect this issue. As discussed earlier, Matlab, powerful image-processing software, should have the ability to perfect stroke order to its highest standard. Since I was given nine months to complete my Senior Project, I could accomplish a piece of the puzzle, which was once again to construct a Matlab/InkML library to render the stroke order of a symbol expression. In terms of

future work, there could be signal processing and neural network programming to better enhance the handwriting recognition problem of a symbol expression in my Matlab/InkML library.

Bibliography

Coffin, Drew, ed. "Image Formats: What's the difference Between JPG, GIF,PNG?" www.practicalcommerce.com. Last modified April 15, 2010. Accessed April 27, 2015. http://www.practicalecommerce.com/articles/1821-Image-Formats-What-s-the-Difference-Between-JPG-GIF-PNG-.

"Dealing with “Xerces hell” in Java/Maven?" Stack Overflow. Accessed April 27, 2015. https://stackoverflow.com/questions/11677572/dealing-with-xerces-hell-in-java-maven.

"Digital Image File Types Explained." www.users.wfu.edu. Accessed April 27, 2015. http://users.wfu.edu//matthews/misc/graphics/formats/formats.html.

Galli, A.W., G.T. Heydt, and P.F. Ribeiro. "Exploring the Power of Wavelet Analysis." *Computer Applications in Power* 9, no. 4 (October 1996): 37-41. Accessed April 27, 2015. doi:10.1109/67.539845.

"JPEG compression algorithim implementation in Matlab." www.mathworks.com. Last modified August 23, 2014. Accessed April 27, 2015. http://www.mathworks.com/matlabcentral/answers/152071-jpeg-compression-algorithm-implementation-in-matlab.

"Modifying zlib and libpng for lossy image compression." Accessed April 27, 2015. http://membled.com/work/apps/lossy_png/.

Pradeep, J. "Diagonal Based Feature Extraction for Handwritten Alphabets Recognition System using Neural Network." *International Journal of Computer Applications* 8, no. 9 (October 2010): 17-22.

Read, Janet C., Stuart MacFarlane, and Matthew Horton. *The Usability of Handwriting Recognition for Writing in the Primary Classroom*. Vol. 1 of *People and Computers XVIII - Design for Life*. London, UK: Springer London, 2005.

"Read image from graphics file." www.mathworks.com. Accessed April 27, 2015. http://www.mathworks.com/help/matlab/ref/imread.html.

"Reading XML document and return Document Object Model node." www.mathworks.com. Accessed April 27, 2015. http://www.mathworks.com/help/matlab/ref/xmlread.html.

Sukanya, Y., and J. Preethi. "Analysis of Image Compression using Wavelet Transform with GUI in Matlab." *International Journal of Research in Engineering and Technology* 2, no. 10 (October 2013): 595-603.

"Supported File Formats for Import and Export." www.mathworks.com. Accessed April 27, 2015. http://www.mathworks.com/help/matlab/import_export/supported-file-formats.html.

Tran, Dat, Wanli Ma, and Dharmendra Sharma. "Handwriting Recognition Applications for Tablet PCs." *www.fujipress.com*. Last modified May 22, 2007. Accessed April 27, 2015. http://www.fujipress.jp/finder/preview_download.php?pdf...PRE...

Watt, Steven. "Ink Markup Language (InkML)." www.w3.org. Last modified September 2011. Accessed April 27, 2015. http://www.w3.org/TR/InkML/.

"Polynomial Approximation in Handwriting Recogntion." Paper presented at Proceedings of the 2011 International Workshop on Symbolic-Numeric Computation, San Jose, Ca, June 7, 2011.

Watt, Steven, Phd, and Rui Hui. "Representation, Recogntion and Collaboration with Digital Ink." University of Western Ontario, 2013.

"Web Graphics." www.mccauslandcenter.sc.edu. Accessed April 27, 2015. http://www.mccauslandcenter.sc.edu/mricro/obsolete/graphics/graphics.html.

Zabala, Alaitz. "Effects of JPEG2000 lossy compression on remote sensing image classification for mapping natural areas." *ACM* 9, no. 93 (July 14, 2005): 1-9.

Zagik, Ziga. "Character Recognition." Unpublished raw data, University of Ljubljana, Ljubljana, Slovenia, n.d.