POLARIZING ROWDY

Pranav A. Bhounsule, Ph.D.
Dept. of Mechanical Engineering
San Antonio, TX, USA 78249
pranav.bhounsule@utsa.edu

Bradley Hammond
Mechanical Engineering Student
San Antonio, TX, USA 78249
bradley.s.hammond@gmail.com

ABSTRACT

The goal of this project was to create a set of parametric equations to define the outermost curve of the mascot for the University of Texas at San Antonio. This was accomplished in three parts: image processing, data processing, and curve fitting. The results were accurate with maximum error under 3% for each parametric curve. The accuracy of the solution depended on time and computing power, and the lack of these prevented the results from having higher accuracy.

1. INTRODUCTION

The majority of the work on this project came from Wolfram Alpha analysts that created curves for objects and people like: Barack Obama, the Pink Panther, and Adele [1]. In order to accomplish the majority of the image processing and data plotting MathWorks was used to provide understanding to the commands used in MATLAB [2]. The data processing was done in both MATLAB and Microsoft EXCEL. The majority of the curve fitting was accomplished using Nutonian's Eureqa software. Information for the general proceedings came from a mixture of prior knowledge and the above mentioned resources.

2. NOMENCLATURE

MATLAB: Matrix Laboratory, software by MathWorks.

EXCEL: Spreadsheet manipulation software, by Microsoft.

Cartesian: Typical x-y-z coordinate system.

3D image: Image file that has three distinct inputs for color values. Commonly RGB.

2D image: Image file that has two distinct inputs for color values. Commonly BW.

Eureqa: Curve fitting software, by Nutonian.

WolframAlpha: Alternative software used by others to fit equations to curves.

3. METHODS

In order to be familiarized with complex curve fitting, the first steps taken were research of a blog performed by Wolfram Alpha analysts that created curves for objects and people like: Barack Obama, the Pink Panther, and Adele [1]. After reading through this blog, it was clear that there are certain steps and flows required for curve fitting. The determined steps were: image processing, data processing, and curve fitting.

Before tackling the Rowdy image, a lot of work was done on a simple curve that was found on the internet. This simple curve was used as a sample in order to understand the concepts of curve fitting and to verify that the equation solver was indeed working correctly. This work was a precursor, but not necessarily a vital part in determining the equations for Rowdy.

Following the first and second steps majorly relied on the use of MATLAB. The first the image for Rowdy was pulled off of the internet from the UTSA webpage [3]. Once the image was then read into MATLAB by the command 'imread()'. In order to verify that the image was properly read, the command 'imshow()' was used to display the image in a figure. The read image can be seen in Figure 1. Once the image was verified to be appearing correctly, the next step was taken.

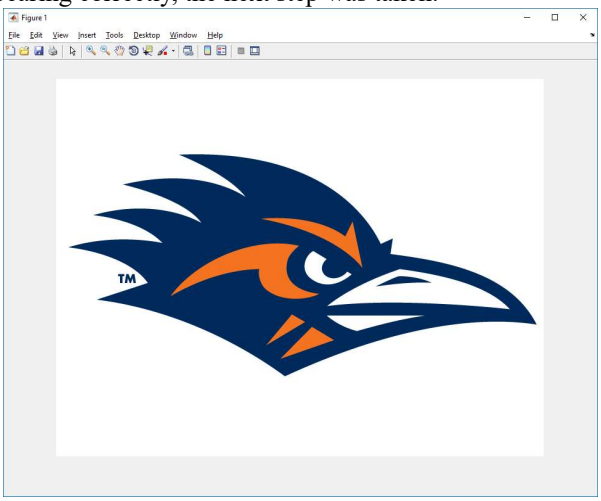

Figure 1: Original Rowdy Image

Originally, a lack of understanding of how to properly trace the image was missing. After various trials and errors, the command 'im2bw()' was used to convert any of the colors that were not white in the original image to black, and to leave any white space in the image. This also compressed the png three dimensional file to a black and white two-dimensional file. This compressed file can be seen in Figure 2.

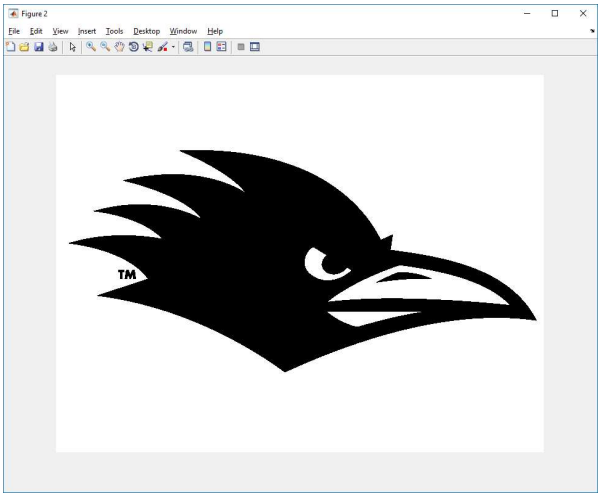

Figure 2: Compressed BW Image

After converting the image to a usable format, the command 'edge()' was used to trace all of the edges in the modified black and white rowdy image as seen in Figure 3.

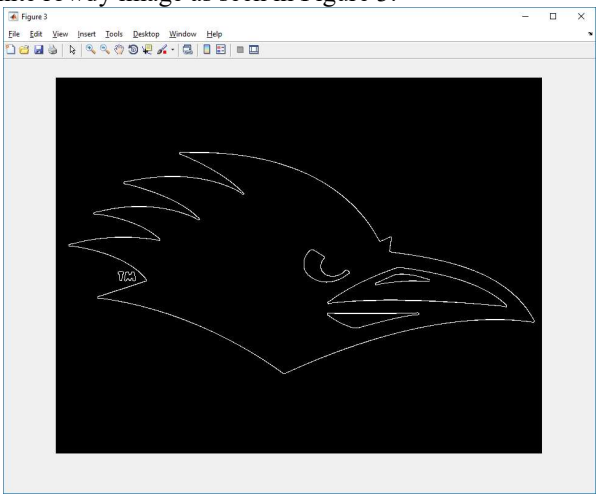

Figure 3: Traced Image

Using the edges of the image, the command 'bwboundaries()' was used to get the coordinates for the traced boundaries in terms of the photograph coordinate system.

Once the edge of Rowdy was determined, the next step was determining data points from the traced edge. This was accomplished by first determining the size to the traced lines. This was done using the command 'size()'. Once the size was determined, the column and row used to start the next trace were determined. In addition, this command converted the image coordinate system into the Cartesian coordinate system.

The command 'bwtraceboundary()' was used to find the data points for the trace shown above. The trace only determined the value for the outermost edges. In order to assure that no other edges were traced, the compressed image was shown with the newly traced data points on top of it, shown in Figure 4.

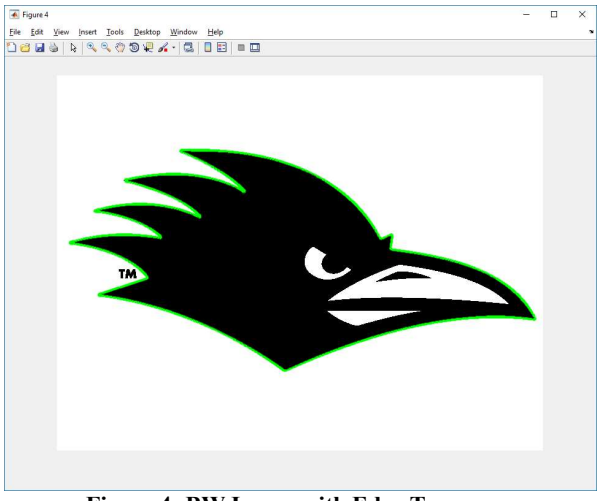

Figure 4: BW Image with Edge Trace

After verifying that the outer edges were traced, and no other edges were included, the image processing phase concluded and the data processing phase began.

The first step in the data processing phase was to get the data points from the edge trace into arrays. This was done using the output of the 'bwtraceboundary()' function used to determine the edges. Columns and rows of the edge were outputs of said command. The data from this action is seen in Figure 5.

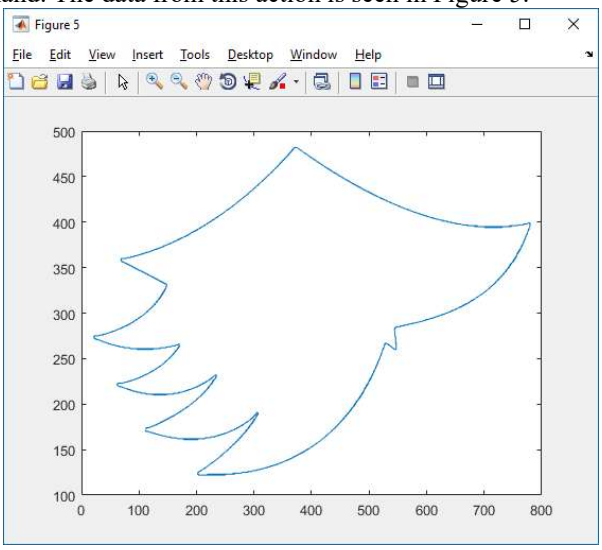

Figure 5: Raw Data Rowdy Outline

The first step after gathering the converted data was to adjust it to the correct orientation, and then to move the correctly oriented Rowdy outline with the origin about the middle of the outline. This was accomplished by finding the maxima and minima of the row and column vectors. Once the minima and maxima were found, these values for the columns were summed and divided by two and the values for the rows were simply subtracted in order to find the location of the middle point of the outline. These new adjusting factors were then used to "move" the outline to the correct position with the middle of the outline at the origin. This new adjusted data is seen in Figure 6.

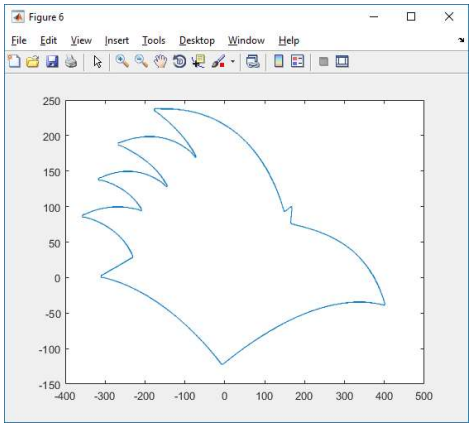

Figure 6: Correctly Oriented Cartesian Rowdy Outline

Once the correct orientation of the data was verified, the next step was to convert the data from Cartesian to polar coordinates. This was done using the command 'cart2pol()'. The data was then shown to be incorrectly oriented once again. This can be seen in Figure 7. The data was converted to polar because the data seems to fit better to polar than Cartesian by inspection.

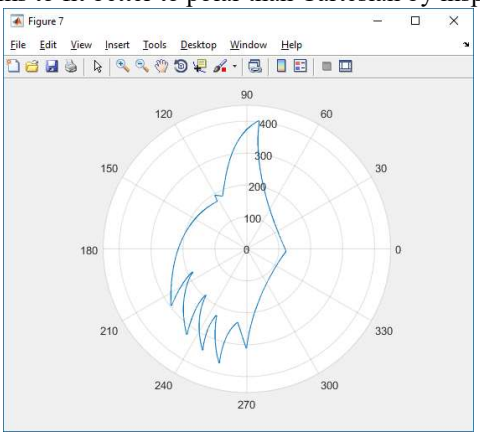

Figure 7: Incorrectly Oriented Polar Rowdy Outline

In order to orient the polar plot correctly, the theta values were subtracted by 90° which is $\frac{\pi}{2}$ radians. This correctly oriented the data, which can be seen in Figure 8.

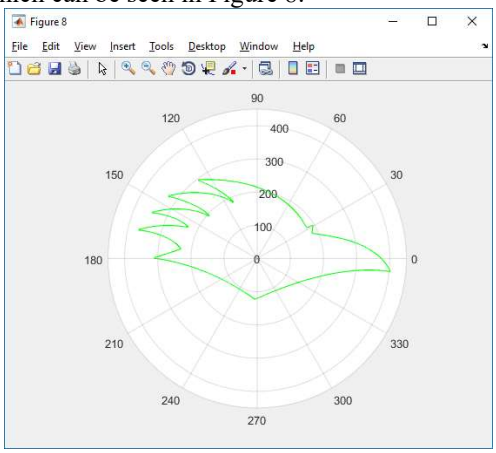

Figure 8: Polar Rowdy Outline

After plotting the polar rowdy, it was seen from a simple plot of theta and rho values versus the simple time step set for parametrization that there was more than one period. This can be seen in Figure 9 and Figure 10.

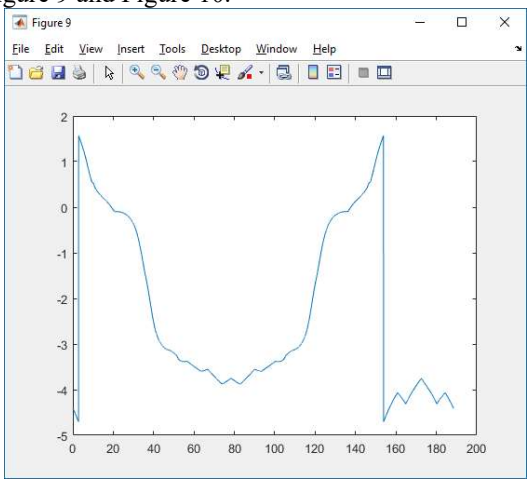

Figure 9: Theta vs. T Before Adjustment

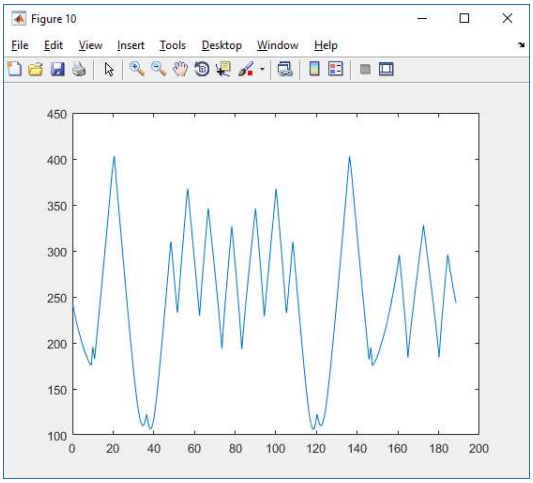

Figure 10: Rho vs. T Before Adjustment

In order to clean and reduce the data to only a single period, the raw data was input into Microsoft EXCEL. From inspection on the time domain plots of rho and theta, the pattern was observed and located in the data by manually searching for the maxima and minima. When the repeating period was removed, it was also observed that the values for rho and theta were discontinuous because of the data manipulation. This was rectified by reordering the data to have only continuous values for rho and theta versus t. The newly adjusted data was then imported back into MATLAB, as well as imported into Nutonian's Eureqa. The final adjusted data is shown in Figure 11 and Figure 12. The final time step is from 0 to 2π.

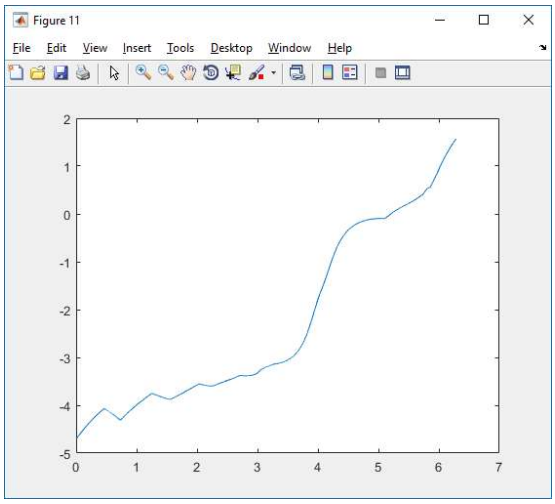

Figure 11: Adjusted Theta vs. T Plot

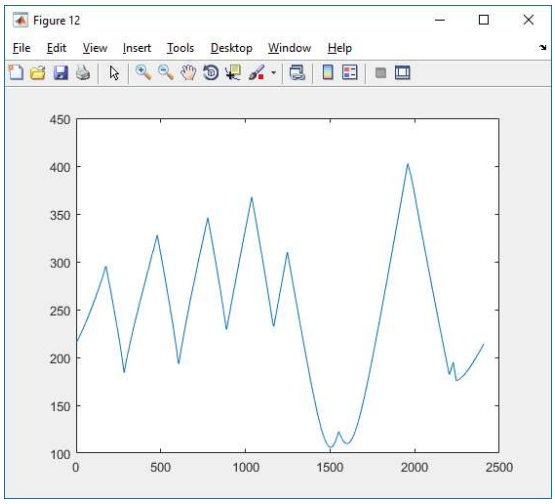

Figure 12: Adjusted Rho vs. T Plot

After the data was adjusted, the radius vector was scaled down by a factor of 20. This can be seen in Figure 13. The reason for sizing down the data was for the equation solver.

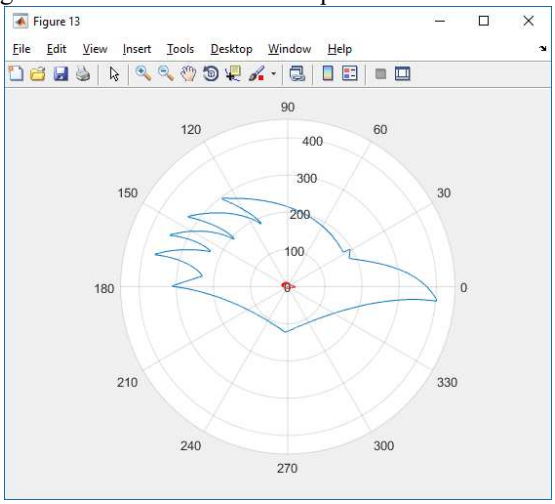

Figure 13: Scaled Down Rowdy Outline

Once the data had been completely conditioned in order to create the "easiest" equation to solve, the data processing was finished and the next phase, curve fitting, was started.

The first step in curve fitting was importing the data into the equation solving software, Eureqa. Once the data was imported, the function for theta and rho was determined. Both theta and rho were made to be only functions of t. This provides for a purely parametric relationship between rho and theta. In addition to setting the form of the function being solved, the formula building-blocks were chosen. These options for which formulae to include came from the blog [1] as mentioned above. The options selected were as follows: constant, input variable, addition, subtraction, multiplication, division, negation, sine, cosine, tangent, square root, step function, sign function, arcsine, arccosine, and arctangent.

Once the formula blocks were chosen, all that could be done was wait. The equation solver was run for 114 hours, 31 minutes, 57 seconds for theta and 116 hours, 11 minutes, 45 seconds for radius. The results of theta versus t are shown in Figure 14 and the results for radius versus t are shown in Figure 15. The equation curve is shown in red.

Figure 14: Eureqa Theta Equation

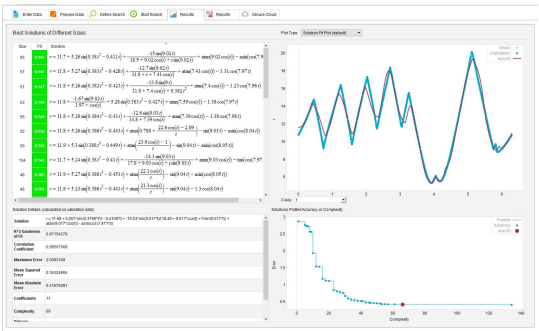

Figure 15: Eureqa Rho Equation

Because the deadline for the project approached before the equation had been fully solved, the equations for theta and rho were stopped prematurely. The functions for each were then input into MATLAB in order to plot the parametric plots of the equations. The final parametric equations plotted are shown in Figure 15. The final result of the parametric equation is shown against the raw data in Figure 16.

4. RESULTS

The results of the curve fitting of the outline of Rowdy are seen in Figure 15. This plot shows the parametric equations only. Figure 16 shows the parametric equations versus the raw data (after manipulation) for reference.

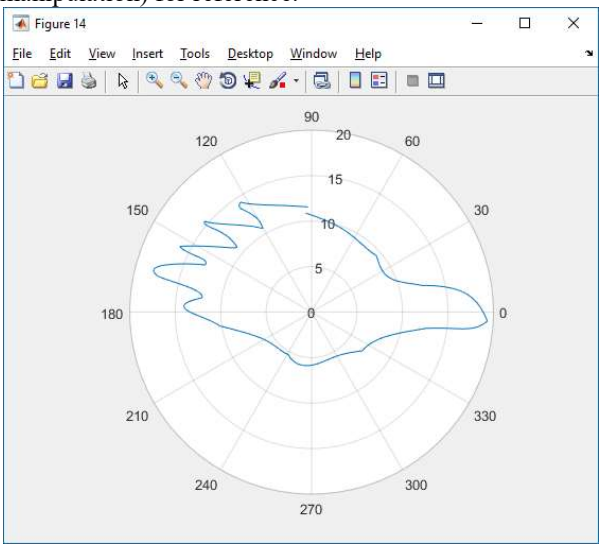

Figure 16: Final Parametric Rowdy Outline

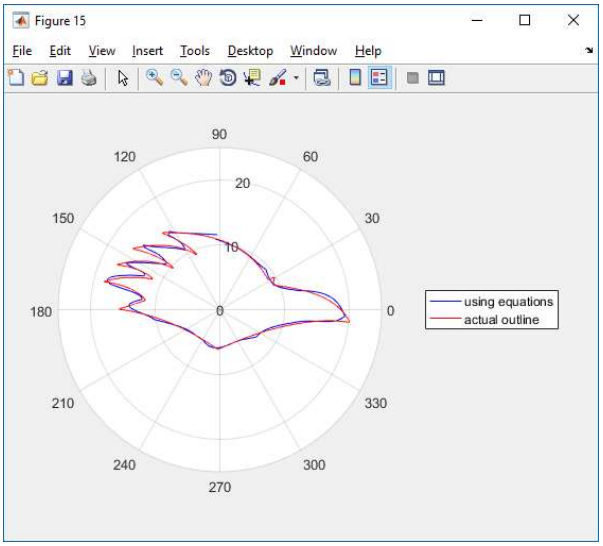

Figure 17: Parametric vs. Raw Data

The equation for theta is below:

θ(t) = 0.556738090065424*atan(t) + 0.010487943000249*t^3*sin(t) + sqrt(0.010487943000249*t)*cos(5.95913927067941 + 8.19679130439689*t) - 1.77704993174192 - 0.195473619171639*cos(5.95913927067941 + 8.19679130439689*t) - 1.83359205127117*atan(11.1783053639771 - 2.75262576041691*t)

The equation for rho is below:

ρ(t) = 66 0.145 "r = 11.6780633156058 + 5.25737866674741*sin(0.379873080930965*t^2 - 0.410797114659848*t) + - 15.0338841955986*sin(9.01717345398399*t)/(18.4868878908716 + 9.01717345398399*cos(t) + t*sin(9.01717345398399*t)) + atan(9.01717345398399*cos(t)) - asin(cos(7.9712359849161*t))

As seen in the figures, the fit that was produced is at least visually recognizable as close to Rowdy when the two are set on top of each other. However, looking at the parametric plot by itself may not lend the observer to immediately think of Rowdy.

The equations that are used for the parametric curve are still basic equations in that no step or sign functions were used to create the difficult curves seen in Rowdy's outline. However, for having only basic equations, such as sine, cosine, tangent, square root, and others the results are surprisingly accurate.

According to the absolute error from the equation solver, the error on the theta equation is a maximum of 0.12187883% and the maximum error on the rho equation is 2.0093349%.

5. DISCUSSION

The results are surprisingly accurate for a first try at curve fitting. The maximum error between the two parametric equations is just over 2%. The fit of the parametric curves over the top of the raw data shows how similar the two plots are. This is a promising curve for the outline of Rowdy.

As seen in Figure 14 and 15, the curve of theta vs t and rho vs t is not perfect. These small deviations seen in the parametric equations compound and show larger errors in the final plot of theta vs t and rho vs t as the inputs to a polar line.

The first reason why the results are not as accurate as they could be is because of the complexity of the curves. Looking at Figures 11 and 12 it can be seen that the curves for both theta and rho are not easily expressed by simple equations. The other complexity is that the Rowdy outline has very harsh curves, not smooth ones, which makes the simple sine or cosine waves not able to be applied to the curve.

The second reason why the curves are not as accurate as they could be is because of time. In any application involving curve fitting, time will always be a great factor. If the curve is very complex, then much more time will be needed compared to a simple curve.

The third reason why the curves are not as accurate as they could be is because the number of cores used to compute the calculations was only 8. The option to double or quadruple the cores was available for a price through Nutonian. However, because of the cost limitations, this option was not explored, and thus the number of searches and computations per second was greatly reduced.

The limitations of this approach are somewhat outlined in the reasons why the final curve was not as accurate as desired. If there had been more time, more computer cores, or simpler curves, the equation for Rowdy's outline surely would have been determined.

6. CONCLUSION AND FUTURE WORK

The project for consisted of three main parts: image processing, data processing, and curve fitting. Each part was accomplished with the help of MATLAB, EXCEL, and Eureqa. While the results were fairly accurate, the results desired were not accomplished. This was mainly due to limitations in time, processing power, and the complexity of the curve. The biggest lesson to be learned from this process is that time and processing power are needed to accomplish the fitting of complex curves. In addition, had the project research and work been started and understood earlier, the results may have been more accurate.

Currently the equation solver is still searching for solutions to the outline of Rowdy. This is because I plan to finish what was started. This project has the basis for many future curve to be fit. The reason why curve equations are useful is because instead of trying to distort the image file of a logo, one can simply increase the sizing factor on an equation like radius and increase the size of the logo without distorting the pixels. Hopefully this project can be completed soon.

ACKNOWLEDGMENTS

Dr. Pranav Bhounsule – for all of his help as my professor this semester

Rebecca Hammond – my wife for being supportive and enduring my long nights working on this project

Michael Mayer – my peer and friend in Engineering for always motivating me to do better and to keep striving for greatness in my studies

REFERENCES

[1] Trott, M. (2009). Making Formulas… for Everything—From pi to the pink Panther to sir Isaac Newton. Retrieved December 13, 2016, from WolframAlpha, http://blog.wolframalpha.com/2013/05/17/making-formulas-for-everything-from-pi-to-the-pink-panther-to-sir-isaac-newton/

[2] MATLAB and Simulink for technical computing - MATLAB & Simulink. (1994). Retrieved December 13, 2016, from MathWorks, https://www.mathworks.com/

[3] Welcome to the university of Texas at San Antonio. (2016). Retrieved December 13, 2016, from UTSA, https://www.utsa.edu/