Comparison of Fourier Descriptor Methods for Shape Classification

Jonathan Gill
Electrical Engineering
University of Arizona
Tucson, AZ, United States
jtgill@email.arizona.edu

***Abstract* - Shape classification is an important topic in fields ranging from Biology to Astronomy. There are many methods to execute shape classification, and none of them constitute a panacea. Most are robust for a well-defined application, but perform poorly outside of that application. In this paper we examine the accuracy of handwritten digit classification using two flavors of scale and rotationally invariant Fourier Descriptors: centroid distance functions and polygonal approximation. Testing and training is performed using Cross Validation on the BU Handwritten Digit Database. For simplicity, classification is achieved by employing Matlab's built-in K-Nearest Neighbors libraries.**

I. Introduction

This paper compare the accuracy of handwritten character classification using two Fourier Descriptor methods of shape analysis. The first is the Discrete Fourier Transform of the distances between the pixels constituting the object's contour and the object's centroid at finite timesteps. The second method treats the pixels in an object's contour as corners of a polygon. Pixel coordinates are converted into complex numbers, and neighboring corners are connected using parametrized line segments. The resulting contour is decomposed into a Fourier Series.

The first N coefficients from each method are used as the shape's identifying descriptors [modify if you take magnitudes or whatever]. A class label is assigned to each set of descriptors, and both are used to build the classification space of KNN. Testing data is decomposed using the same two methods, and its separate descriptors are fed into the separate KNN classifiers, from which a class label is produced. The accuracy of each descriptor method for a particular segment of cross-validation is defined as the number of correct class predictions divided by the total number of predictions. The cumulative performance over all tests is then represented as the average of all test outcomes including standard deviation.

This paper will explain the theory behind each descriptor method, show the implementation details using pseudocode, present the results, and compare the average performance of each method for accuracy and precision of class predictions.

II. Theory

The common goal for the following methods is to exploit the periodicity of an object's contour to build descriptors out of Fourier series. This is accomplished by building the object's signature, itself periodic, which can be information about its curvature, arc area, distance metric, etc, and collecting that signature's Fourier coefficients. Ideally these coefficients contain some unique features of the shape in question and can therefore be used to classify a given object.

This paper focuses on building these descriptors for objects in discrete space, so a small grain of salt must be taken before reading the theory: it is assumed that the object is large enough that rotating the object in discrete space does not appreciably modify the object's shape.

Centroid Distance Function

One method for building a contour's shape signature involves the use of a distance function on the shape's outermost contour [1]. The object's centroid is located, and the distance from the centroid to every point on the object's outer contour is calculated as the contour is traversed, as shown in Figure 0.

The distance function is periodic because the contour is closed, which means that the distance function can be represented with a Fourier series. The coefficients obtained from distance function's Fourier series are then used as the object's descriptors.

This paper focuses on the classification of shapes created from digital images, meaning that an object's contour is an ordered list of pixel locations with an arbitrary starting pixel. These locations are written out as a vector,

$$\vec{s}(n) = x_n\hat{x} + y_n\hat{y}$$

where $n \in [0, N-1]$ is the nth pixel encountered along the contour, and $N-1$ is the total number of pixels in the contour. It is worth noting that the discrete contour function is periodic in time; $\vec{s}(n) = \vec{s}(n+N)$. We define the contour's centroid as the contour's center of mass, with every pixel along the contour contributing equal weight.

$$\vec{c}_o = \frac{1}{N}\sum_{n=0}^{N-1}\vec{s}(n)$$

The distance function is the distance from the object's centroid to every point along the object's contour,

$$d(n) = |\vec{s}'(n) - \vec{c}_o|.$$

The distance function is discrete in time, which means that its Fourier Series coefficients are obtained using a discrete sum

$$a_k = \frac{1}{N}\sum_{n=0}^{N-1} d(n)e^{-jk\omega n} \tag{Eq 1}$$

where $\omega = 2\pi / N$ is the distance function's fundamental frequency and $d(n) = d(n+N)$. In order to use these coefficients as object descriptors, we must assess their volatility to the geometric transformations of translation, rotation, and scaling, as well as the effect of choosing an arbitrary starting point on the contour.

A. *Starting Point Variation*

Let the contour that is obtained by starting at a timestep m be named $\vec{s}'(n)$ where $\vec{s}'(n) = \vec{s}(n+m)$. The centroid is unaltered, and the new distance function, $d'(n)$ is

$$d'(n) = |\vec{s}'(n) - \vec{c}_o| = |\vec{s}(n+m) - \vec{c}_o|.$$

Meaning that the new distance function has the same values as the original distance function, just shifted to the left along the time axis,

$$d'(n) = d(n+m),$$

and the Fourier series coefficients take on a complex phase factor,

$$a'_k = a_k e^{-jk\omega m}$$

due to the time delay property of Fourier series decomposition.

B. *Translation*

Let the object be translated in the xy plane by some discrete-valued vector $\vec{p}$. The contour function that results from this translation will be called $\vec{s}'(n)$, and it is given by

$$\vec{s}'(n) = x_n\hat{x} + y_n\hat{y} + \vec{p}.$$

The centroid of the translated contour, $\vec{c}_o{}'$ is then

$$\vec{c}'_o = \frac{1}{N}\sum_{n=0}^{N-1}\vec{s}'(n) = \frac{1}{N}\sum_{n=0}^{N-1}(\vec{s}(n) + \vec{p}).$$

Since $\vec{p}$ is a constant vector with no dependence on n, the sum simplifies to

$$\vec{c}'_o = \vec{p} + \frac{1}{N}\sum_{n=0}^{N-1}\vec{s}(n) = \vec{p} + \vec{c}_o.$$

And the distance function from translation is the same as the distance function before translation.

$$d'(n) = |\vec{s}'(n) - \vec{c}'_o| = |\vec{s}(n) - \vec{c}_o| = d(n). \tag{Eq 2}$$

So the Fourier series coefficients are immune to translation of the object.

C. *Rotation and Scaling*

Though the contour we're describing is discrete, the following analysis operates under the assumption that rotating the contour by any angle preserves relative pixel locations. Let the object be rotated by an angle θ and isotropically scaled by a factor h,

$$\vec{s}'(n) = h\begin{pmatrix}\cos\theta & \sin\theta\\ -\sin\theta & \cos\theta\end{pmatrix}\vec{s}(n)$$

$$= h(x_n\cos\theta + y_n\sin\theta)\hat{x} + h(-x_n\sin\theta + y_n\cos\theta)\hat{y}.$$

Let R be the rotation matrix. The centroid becomes

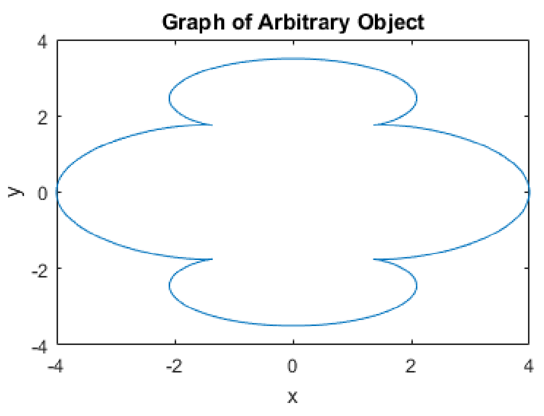
$$\vec{c}'_o = \frac{1}{N}\sum_{n=0}^{N-1}\vec{s}'(n) = \frac{1}{N}\sum_{n=0}^{N-1} hR\vec{s}(n) = hR\vec{c}_o$$

which is simply a scaled and rotated version of the original centroid. Using this same notation, the distance function for the rotated and scaled contour becomes

$$d'(n) = |\vec{s}'(n) - \vec{c}'_o| = h|R(\vec{s}(n) - \vec{c}_o)|.$$

The magnitude of a vector being acted on by a rotation matrix is simply the magnitude of the unrotated vector, which means that

$$d'(n) = h|R(\vec{s}(n) - \vec{c}_o)| = hd(n)$$



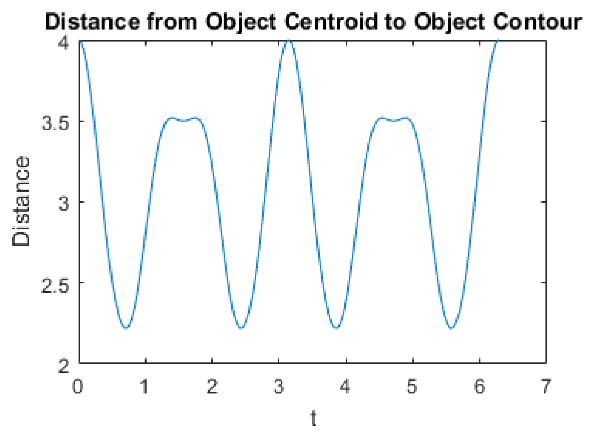


Figure 1
An arbitrary object is shown on the left. On the right is the object's centroid distance function. Note the four valleys corresponding to the distances at the sharp corners of the object.

and the new distance function is simply the old distance function multiplied by a scale factor. This leads us to the intuitive conclusion that the Fourier series coefficients will be multiplied by the same scale factor.

$$a'_k = ha_k \qquad \text{(Eq 3)}$$

D. Descriptors

At worst the entire object we wish to describe will have elements of all of these transformations, meaning that it will have an arbitrary starting point, and be translated, rotated, and scaled. Ultimately the Fourier series could have coefficients of the form

$$a'_k = ha_k e^{-jk\omega m}$$

which are not scale or starting point invariant. The task now becomes using these coefficients to build a transformation invariant description of the object's contour. Possible descriptors of the form

$$b_k = \left|\frac{a'_k}{a'_0}\right| = \left|\frac{ha_k e^{-jk\omega m}}{ha_0}\right| = \left|\frac{a_k}{a_0}\right| \qquad \text{(Eq 4)}$$

come to mind most easily; the magnitude removes the phase factor and dividing by the first zeroth coefficient undoes any scaling. These b_k values are therefore invariant to translation, scaling, rotation, and starting point variation, and will be used to describe our objects. We will neglect the term b_0, as it will always be one.

Polygonal Approximation

Unlike the centroid distance function, the polygonal approximation has its descriptors created from Fourier decomposition in continuous space. Instead of taking the distances between the centroid and every pixel on the object's contour, we convert the contour's pixel locations from real-valued vectors to coordinates in the complex plane [2]. Pixel locations along the contour are labeled as "corners", and adjacent corners are connected in the complex plane with parametrized line segments. The result is a piecewise continuous contour in the complex plane (Figure 0).

The polygon consists of N total corners, which are written as complex values z_k. Since the contour is closed, $z_{k+N} = z_k$. The contour is composed of segments of the form

$$f_k(t) = a_k t + b_k, \qquad t_{k-1} \le t \le t_k$$

where t is the parametrization, and a_k and b_k are complex with z_{k-1} and z_k determining their values:

$$a_k = \frac{z_k - z_{k-1}}{t_k - t_{k-1}} = v_k\ ,$$
$$b_k = z_k - v_k t_k.$$

The value v_k can be thought of as a complex velocity whose magnitude prescribes the total amount of time it will take to pass from z_{k-1} to z_k. To find appropriate values for the t_k's, we will assume that all faces of the contour are traversed with a constant speed S, the time of traversal around the entire contour is 2π, and that $t_1 = 0$. Let L represent the length of the entire contour, and our constraints provide the following simplifications:

$$|v_k| = S\ \ \forall k$$
$$\frac{L}{S} = t_N - t_1 = 2\pi \therefore S = \frac{L}{2\pi}$$

where

$$L = \sum_{k=1}^{N} |z_k - z_{k-1}|.$$

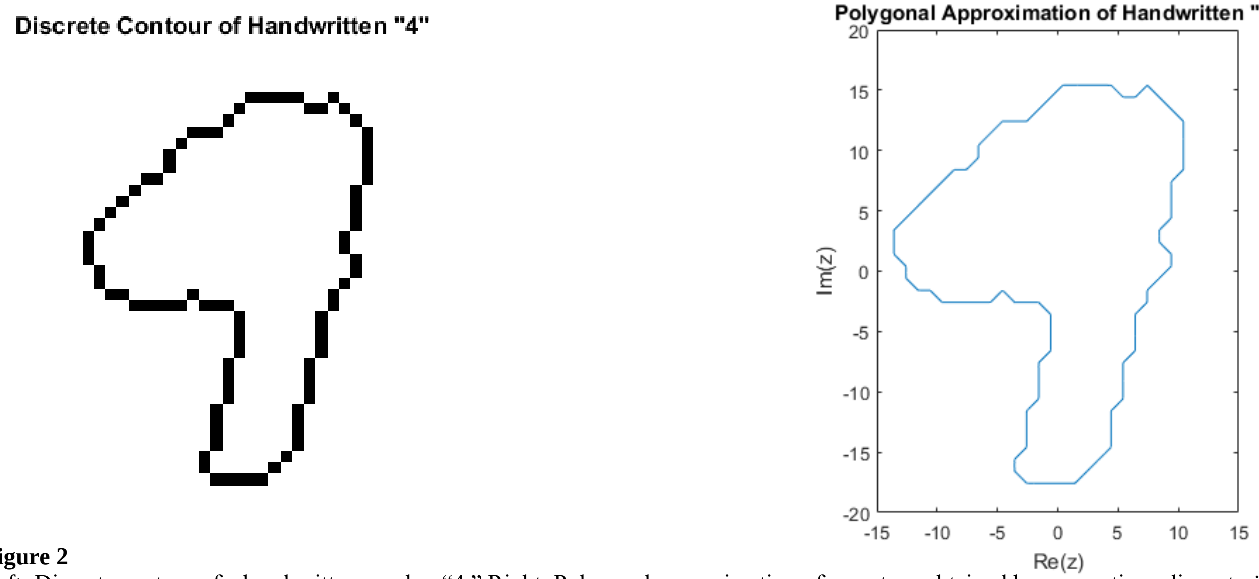


Figure 2
Left: Discrete contour of a handwritten number "4." Right: Polygonal approximation of a contour obtained by connecting adjacent pixels in the discrete contour.

This allows us to build a recurrence relation for the values of t_k,

$$|v_k| = \frac{L}{2\pi} = \left|\frac{z_k - z_{k-1}}{t_k - t_{k-1}}\right|,$$

assuming that time increases linearly and not as "a big ball of wibbly wobbly, timey wimey stuff", we know that $t_k > t_{k-1}$ $\forall k$, meaning that the absolute value bars around the time difference can be removed, and

$$t_k = \frac{2\pi|z_k - z_{k-1}|}{L} + t_{k-1}. \qquad \text{(Eq 5)}$$

The periodicity of the contour and the previous constraint that we placed upon time instants ensures that.

$$t_{k+N} = t_k + 2\pi \qquad \text{(Eq 6)}$$

The complete contour is then,

$$g(t) = \bigcup_{k=1}^{N} f_k(t),$$

and we fully know the time instants where the contour traversal reaches a corner. Moreover, we know that $g(t + 2\pi) = g(t)$ because the contour is closed. Similar to the centroid distance function, we seek to build descriptors out of this contour using Fourier series.

$$c_n = \frac{1}{2\pi}\int_0^{2\pi} g(t)e^{-jnt}dt = \frac{1}{2\pi}\sum_{k=1}^{N}\int_{t_{k-1}}^{t_k} f_k(t)\, e^{-jnt}dt$$
$$= \frac{1}{2\pi}\sum_{k=1}^{N}\left[\frac{j}{n}\left(f_k(t_k)e^{-jnt_k} - f_k(t_{k-1})e^{-jnt_{k-1}}\right) - \int_{t_{k-1}}^{t_k}\frac{jv_k}{n}e^{-jnt}dt\right]$$
$$= \frac{1}{2\pi}\sum_{k=1}^{N}\left[\frac{j}{n}\left(z_k e^{-jnt_k} - z_{k-1}e^{-jnt_{k-1}}\right) + \frac{v_k}{n^2}\left(e^{-jnt_k} - e^{-jnt_{k-1}}\right)\right]$$

Since the contour is periodic the $1/_n$ terms will cancel each other out as the sum is evaluated, leaving the sum over the $1/_{n^2}$ term.

$$c_n = \frac{1}{2\pi}\sum_{k=1}^{N}\left[\frac{v_k}{n^2}\left(e^{-jnt_k} - e^{-jnt_{k-1}}\right)\right]$$
$$= \frac{1}{2\pi n^2}\sum_{k=1}^{N}(v_{k-1} - v_k)e^{-jnt_k}, \quad n \neq 0 \qquad \text{(Eq 7)}$$

The last result can be obtained by expanding the sum and collecting terms with like powers of e^{-jn}. The DC term of the Fourier series represents the polygon's centroid.

$$c_0 = \frac{1}{2\pi}\sum_{k=1}^{N} f_k\left(\frac{t_k + t_{k-1}}{2}\right)(t_k - t_{k-1}). \qquad \text{(Eq 8)}$$

We will now examine these coefficients' susceptibility to geometric transformations. By observation it's easy to see that our choice of starting point will not affect the coefficients. This can easily be demonstrated by shifting the beginning and end points in the sum over k and noting that $v_{k+N=}v_k$, and $t_{k+N} = t_k + 2\pi$.

E. Translation

The polygon can be translated in complex space by adding the translation $w \in \mathbb{C}$ to each of the corners and recalculating the coefficients for the $f_k(t)$ line segments. As before, the translated parameters will be denoted with a dash.

$$v'_k = \frac{z'_k - z'_{k-1}}{t'_k - t'_{k-1}} = \frac{(z_k + w) - (z_{k-1} + w)}{t'_k - t'_{k-1}}$$
$$v'_k = \frac{z_k - z_{k-1}}{t'_k - t'_{k-1}}.$$

The polygon's perimeter will not be affected by translation, and we will impose the same constraint on the speed of the contour traversal as in (Eq 6).

$$|v'_k| = |v_k| = \frac{L}{2\pi} = \left|\frac{z_k - z_{k-1}}{t'_k - t'_{k-1}}\right|$$

and

$$t'_k - t'_{k-1} = \frac{2\pi|z_k - z_{k-1}|}{L} = t_k - t_{k-1},$$

so primed velocities are exactly equal to the unprimed velocities. This means that the Fourier series coefficients for non-zero frequencies are unaffected by translation, but the DC offset is translated by w.

F. Rotation and Scaling

Rotating and scaling the object in the complex plane is as simple as multiplying every point on the object by a term $he^{j\theta}$. We multiply the corners of the polygon by the scale and rotation term and propagate the results to the connecting line segments. Polygon corners that undergo scaling and rotation are denoted with primes.

$$z'_k = hz_k e^{j\theta} \rightarrow$$
$$v'_k = \frac{z'_k - z'_{k-1}}{t'_k - t'_{k-1}} = he^{j\theta}\frac{z_k - z_{k-1}}{t'_k - t'_{k-1}}$$

Continuing the imposition of the time and speed constraints in (Eq 6) we know that

$$|v'_k| = \frac{h|z_k - z_{k-1}|}{t'_k - t'_{k-1}} = \frac{L'}{2\pi}, \qquad \text{(Eq 9)}$$

where the polygon's perimeter L' can be written in terms of the unscaled perimeter as

$$L' = \sum_{k=1}^{N}|z'_k - z'_{k-1}| = \sum_{k=1}^{N} h|z'_k - z'_{k-1}| = hL.$$

As an unfortunate side effect to the scaling operation, the speed with which the contour is traversed must increase to compensate for the time restrictions. However, we retrieve an endearing result:

```
for each row and column of I(r,c):
    if I(r,c) is foreground:
        start_pixel = (r, c)
    end
end

num_pixels = R*C
current_pixel = start_pixel
contour = new list
first_loop = 1
num_iterations = 0

while first_loop || current_pixel != start_pixel:

    for each neighbor_pixel = 8-connected neighbor of current_pixel:
        for each sub_neighbor = 4-connected neighbor of neighbor_pixel:
            if I(sub_neighbor) == background /
            && neighbor_pixel not in last 3 entries of contour[]:
                current_pixel = neighbor_pixel
                contour[num_iterations] = current_pixel
                break
            end
        end
    end
    first_loop = 0
    num_iterations++
    if num_iterations >= num_pixels:
        print "Number of iterations exceeded number of pixels in entire image"
        break
    end
end
```

Figure 3
Pseudocode for generating an ordered list of points describing a binary image's contour

$$t'_k - t'_{k-1} = \frac{2\pi|z_k - z_{k-1}|}{L} = t_k - t_{k-1}.$$

This indicates that the time instants to reach corners are not changed with scaling or rotation. Moreso, the primed velocities are simply scaled and rotated versions of the unprimed velocities.

$$v'_k = he^{j\theta}\frac{z_k - z_{k-1}}{t_k - t_{k-1}} = he^{j\theta}v_k$$

The remaining challenge is to examine the behavior of the Fourier coefficients under the transformation.

$$c'_n = \frac{1}{2\pi n^2}\sum_{k=1}^{N}(v'_{k-1} - v'_k)e^{-jnt'_k}$$
$$= \frac{he^{j\theta}}{2\pi n^2}\sum_{k=1}^{N}(v_{k-1} - v')e^{-jnt_k} \qquad \text{(Eq 10)}$$
$$= he^{j\theta}c_n, \;\; n \neq 0.$$

Unfortunately, the coefficients are not immune to the effects of scaling and rotation of the object. However, as shown with centroid distance functions (Eq 4), these coefficients can still be manipulated to become transformation-invariant shape descriptors.

G. *Descriptors*

Consider a descriptor of the form

$$d_k = \left|\frac{c'_k c'_{-k}}{c'_1 c'_{-1}}\right|, \qquad |k| > 2$$

The magnitude of the fraction is taken to make the elements amenable to KNN analysis in Matlab. We can simplify this in terms of the unscaled and unrotated Fourier coefficients:

$$d_k = \left|\frac{h^2 e^{j2\theta} c_k c_{-k}}{h^2 e^{j2\theta} c_1 c_{-1}}\right| = \left|\frac{c_k c_{-k}}{c_1 c_{-1}}\right|, \qquad |k| > 2 \qquad \text{(Eq 11)}$$

and we have created descriptors that are transformation-invariant.

III. Practical Issues

The core of both algorithms requires knowledge of an object's contour. Of particular difficulty is obtaining a contour in discrete space. It is insufficient to simply obtain a list of points along the contour, the list of points must be ordered in such a way that traversal of the list equates to a clockwise or counterclockwise traversal of the contour, and the direction of traversal must be consistent across every training contour in the data set.

The handwritten digit data set is preprocessed so that the written character is denoted with foreground pixels, and everything else is considered background. The pseudocode for finding the contour of a segmented image is shown in Figure 3. Unfortunately, this algorithm encounters errors when the contour being traversed is part of a "thin" object. More formally, this method of obtaining a discrete contour fails when the object's thickness is less than one pixel.

To mitigate failure of contour tracking from thin objects, each input image is dilated by a small amount with a cross-shaped structuring element bounded by a 3x3 box. This simple step reduced the number of contour tracking failures from over 50% to under 20%. Unfortunately, this reduction in failure introduced a separate error where contour tracking would fail if a particular sequence of foreground pixels was encountered.

The contour list developed a propensity for oscillating between foreground pixels until the iteration failure threshold was exceeded. The reason this happened was due to the order in which neighboring pixels of the current contour pixel were checked; originally, neighboring pixels were inspected left-to-right, top-to-bottom, meaning that diagonal neighbors were inspected before adjacent neighbors. By forcing the contour tracker to first inspect neighbors that were directly adjacent to the current pixel, and diagonal neighbors second, the overall contour tracking error for this dataset dropped from 20% to less than 2%.

A minor issue was encountered when translating the work of Robert [2] into Matlab code. His derivation of the Fourier coefficients for a polygon were zero-indexed, making it difficult to translate the various sums and coefficient calculations into vector operations. The theory has been reformulated in this paper to accommodate future researchers' own implementations.

IV. Implementation

The handwritten digit data is pre-processed with foreground pixels representing a number from 0 – 9. Handwritten digits occupy 32×32 square pixel spaces, and are represented as a series of ASCII data in one continuous text file. Each 1024 series of ASCII values, along with its label, are read into the main Matlab script. The ASCII representation of the digits is converted into an image format, and that image is dilated using a cross-shaped, zero-centered structuring element with bounding dimensions of 3×3.

The contour for every digit is computed, and the descriptors for each dilated digit are created from the contour data. Currently, digits who fail to have their contours created are not rejected from the training data. Only 25 out of the 1934 training data experience a contour traversal failure, amounting

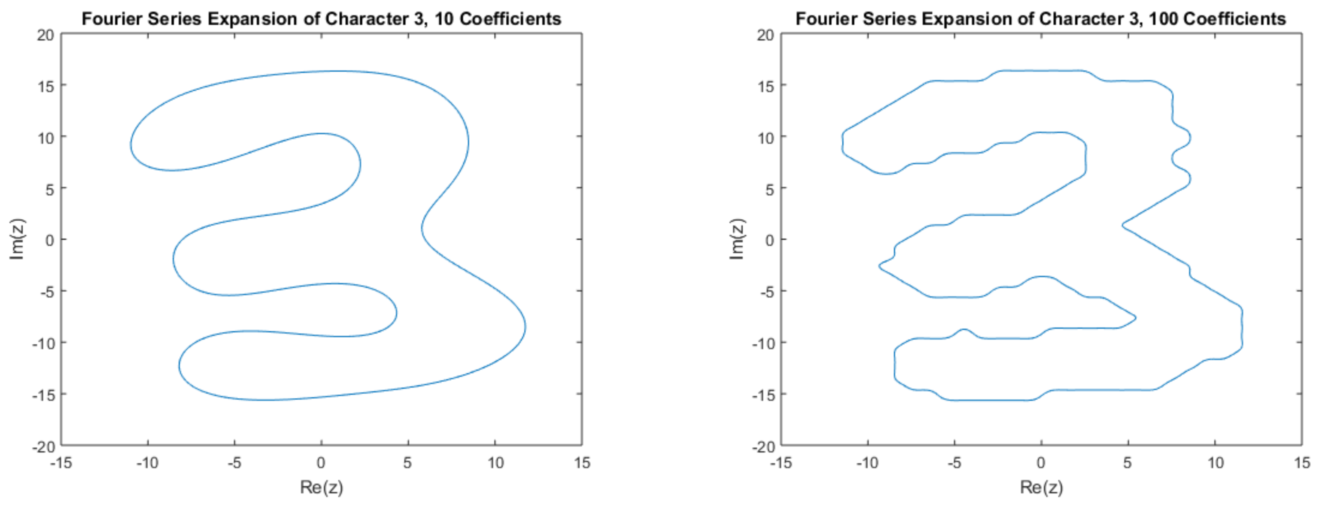


Figure 4
Fourier series representation of a hand-written three for (left) ten principal frequencies and (right) 100 principal frequencies.

to 1.2% of the training data.

In order to verify that the majority of the contours contain the correct pixel values, the pixel locations contained in the contours are plotted on top of the dilated digit images. The plots are inspected manually to verify that the contour points match up with the actual contours of the dilated digits. Not only does the contour have to contain the correct digits, but the digits must also be in the correct order.

```
convert contour pixel coordinates into complex coordinates, z(k)

for k = 2 to num_contour_points:
    x(k-1) = z(k)-z(k-1)
end
x(k) = z(1)-z(k)
x(k+1) = x(1)

for k = 1 to num_contour_points:
    L += abs(x(k))
    %L will be the polygon's total perimeter
end

for k = 1 to num_contour_points+1:
    v(k) = L*x(k)/(2*pi*abs(x(k)))
end

time(1) = 0
for k = 2 to num_contour_points+1:
    time(k) = 2*pi*abs(x(k-1))/L + time(k-1)
end

for n = 1 to num_coefficients:
    for k = 1 to num_contour_points + 1:
        c_pos(n) += (v(k-1)-v(k))*exp(-j*time(k)*n)/(2*pi*n^2)
        c_neg(n) += (v(k-1)-v(k))*exp(-j*time(k)*-n)/(2*pi*n^2)
    end
end

for n = 2 to num_coefficients:
    descriptors(n) = abs(c_pos(n)*c_neg(n)/c_pos(1)*c_neg(1))
end
```

Figure 6a
Pseudocode for generating shape descriptors from the polygonal approximation of the object contour.

```
centroid = [0 0]
for each element n of contour:
    centroid += contour{n}
end
centroid /= length(contour)

for each element n of contour:
    distances(n) = (centroid(1)-contour{n}(1))^2
    distances(n) += (centroid(2)-contour{n}(2))^2
    distances(n) = sqrt(distances(n))
end

coefficients = fft(distances)
descriptors = abs(coefficients/coefficients(1))
```

Figure 6b
Pseudocode for generating centroid distance function and shape descriptors based on the distance function’s Fourier transform. (Curly braces denote that the object is a list).

The contour tracking algorithm was slowed sufficiently to allow a human to view which pixel location was recently added to the contour. The human would then determine the current pixel’s candidacy as the next point along the object’s contour, and verify that the contour was be tracked in a clockwise direction. This process was repeated for several objects. Later, a Matlab script was implemented to verify that the traversal order of each contour was clockwise.

Descriptors from both methods are implemented partially from hand-written code, and partially with the use of Matlab’s built-in methods. The Discrete Fourier Series coefficients for the distance function are computed using the built-in “fft()” function. The Fourier coefficients for the polygonal approximation are computed with Matlab scripts written by the author. Pseudocode for both methods is listed in Figures Figure **6**a and Figure **6**b.

To ensure that the correct Fourier coefficients were being obtained for the polygonal approximation, the Fourier representation of the closed contour was plotted in complex space, without the DC term (Figure 4). The reconstructed curves are out of phase with the originals by $\pi/2$, so each term is multiplied by a factor of $e^{-j\pi/2} = -j$ to retrieve the correct orientation of the polygonal contour. With the verification of the Fourier coefficients, the object descriptors can then be created according to (Eq 4) and (Eq 11).

After the descriptors for either method have been generated, all of the descriptors are passed to a KNN classifier. Ten-fold cross-validation is run using multiple values for hyperparameters K, the number of nearest neighbors to check, and N, the number of descriptors used for classification, in order to obtain the optimal configuration for the classifier. The results of the cross-validation tests are shown in Table 2 and Table 2. These tests were performed for $5 \leq N \leq 15$ and $2 \leq K \leq 5$; the reason for the choice of range for N is using a large number of descriptors introduces noise into the classification, yielding poor results.

The optimized KNN parameters are used to determine which training labels have the best and worst classification results for a small set of hold-out data. We carry this out by re-training KNN with the optimized parameters on all but the last 100 data points, and testing on the held-out data.

V. Results

As stated earlier, the results of the cross-validation tests are shown in Table 1 and Table 2. Surprisingly, the centroid-distance function method out-performed the polygonal approximation for every combination of classifier parameter that was used. Given the ability of the polygonal approximation to exactly reproduce the contour it describes, it was hypothesized that the polygonal approximation would show better results for descriptor classification than centroid-distance functions.

Also worth noting is how the last 100 data points performed for each method. Using centroid-distance functions, 74% of the last data points were correctly classified (see Figure 7), with the number “7” having the highest number of misclassifications. Perhaps not surprising, the classifier most often mistook sevens for nines. An example of a few of these misclassifications is shown in Figure 8.

Comparatively, the results for classifying the last 100 data

Polygonal Approximation Classification Error				
	K=2	**K=3**	**K=4**	**K=5**
N=5	0.392968	0.355895	0.346381	0.344829
N=6	0.390279	0.348811	0.345605	0.344054
N=7	0.388831	0.348656	0.344674	0.344105
N=8	0.385988	0.34969	0.345915	0.340693
N=9	0.385419	0.350259	0.347518	0.342347
N=10	0.387435	0.35	0.34395	0.340848
N=11	0.385936	0.349535	0.348035	0.339452
N=12	0.384695	0.346381	0.343382	0.337952
N=13	0.383402	0.350982	0.345191	0.338831
N=14	0.384023	0.345863	0.342089	0.337746
N=15	0.384643	0.347777	0.343692	0.342089

Table 2
KNN classification error for various choices of number of neighbors (K), and number of descriptors (N) for polygonal approximation based descriptors.

Centroid-Distance Function Classification Error				
	K=2	**K=3**	**K=4**	**K=5**
N=5	0.318046	0.279783	0.273526	0.265357
N=6	0.278283	0.245863	0.24183	0.230455
N=7	0.252172	0.220734	0.221975	0.211479
N=8	0.246949	0.211996	0.213185	0.204343
N=9	0.237125	0.207394	0.20486	0.197622
N=10	0.237384	0.205739	0.203671	0.194829
N=11	0.23423	0.205481	0.205222	0.194002
N=12	0.236815	0.205222	0.203878	0.194054
N=13	0.234953	0.207394	0.201603	0.194623
N=14	0.237022	0.205946	0.201034	0.192554
N=15	0.230972	0.205016	0.200982	0.194623

Table 2
KNN classification error for various choices of number of neighbors (K), and number of descriptors (N) for centroid-distance function-based descriptors.

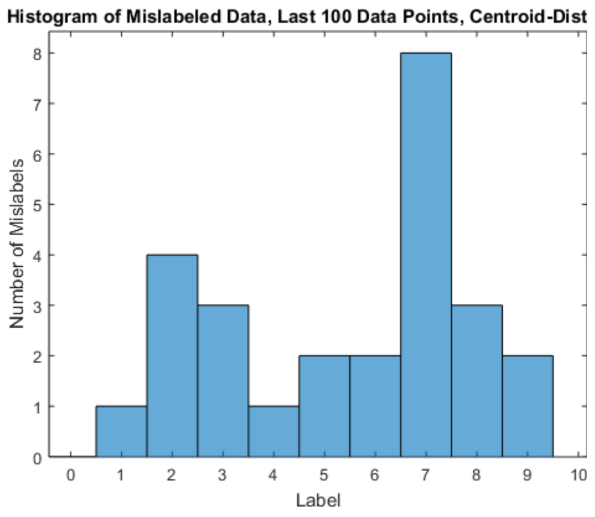


Figure 7
Histogram of mislabeled data points (using centroid-distance) in the training set’s last 100 data points.

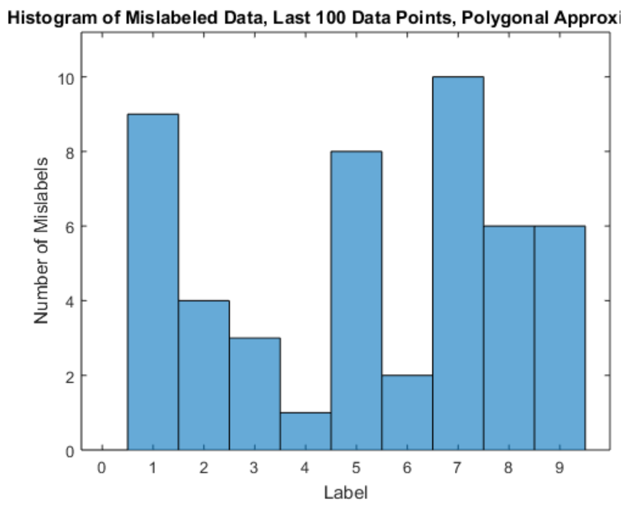


Figure 9
Histogram of mislabeled data points (using polygonal approximation) in the training set’s last 100 data points.

points using the polygonal approximation only managed a 51% accuracy (see Figure 9), even when the classification maximizing parameters were used for KNN. Though the polygonal approximation had dismal results, both descriptor methods showed a maximum misclassification of sevens in the last 100 data points. It is unclear at this point whether this correlation has any statistical relevance, but it does warrant further investigation.

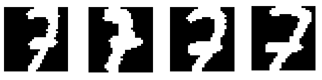


Figure 8
Misclassified sevens from centroid-distance descriptors. From left to right, misclassified as: “9”, “5”, “9”, and “3”

Ultimately, descriptors built out of centroid distance functions have proven themselves to be more robust than descriptors made from polygonal approximations. However, only one method of classification was attempted. These results do not preclude the possibility of the polygonal approximation (or other descriptor methods) outperforming centroid distance functions when other supervised classification methods are employed. These results might also be a result of the particular similarity function being used by Matlab's implementation of KNN.

In particular, the paper by Granlund [3] utilized statistical methods to determine if a particular set of shape descriptors belonged to a given class. However, it should be noted that a limited data set forced him to use the same data for training and testing.

VI. Discussion

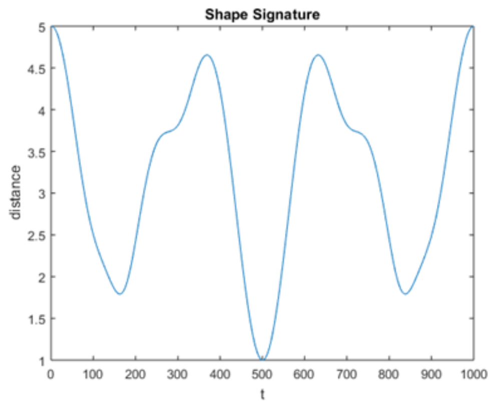
Several potential improvements for classification are mentioned in [1], [4], [5]. In particular, [4] suggests that pixels along the contour be sampled at regular distance intervals. They further suggest that subsampling the number of contour pixels into integer powers of two to optimize the efficacy of Fast Fourier Transform algorithms. In their example, they subsample a contour with a much larger perimeter than any of the training data used in this paper. In this particular implementation the polygonal approximation could be utilized to provide an estimate for the locations of contour pixels. Furthermore, it's easy to see that contour points that are spaced regularly along the object's perimeter are also spread apart by finite time intervals as the contour is traversed. To show this, recall that the contour is traversed with at a constant speed (Eq 9).

Shape classification might also be improved by using different terms of the Fourier series expansion to build invariant descriptors. Granlund [3] uses a combination of asymmetric Fourier coefficients such that "angular information can be defined." He also blatantly states that the coefficients he used were chosen "after merely qualitative considerations." However, we have a computational advantage over Granlund; it is possible for various combinations of algebraically combined Fourier coefficients to be tested for classification accuracy. Care must be taken to ensure that the descriptors retain their transformation invariance, if so desired.

The classification accuracy of the centroid-distance method can potentially be increased by altering the method by which the centroid is calculated. Instead of using just the contour pixels to determine the object's centroid, the object's entire mass could be used. In particular, this could drastically affect the signatures for numbers "9", "8", "4", and "0" in this data set. The motivation behind this change is providing an opportunity for the object's interior to have an effect on the shape signature. By simply observing contours it's easy to imagine that numbers such as "0" and "8" could have similar descriptors, if written sloppily enough. Taking the interior structure into account can translate the centroid, making parts of the contour nearer or farther away from the centroid. The net effect is that the distance signature can be altered (see Figure 10; notice the asymmetry that's been introduced into the signature function as a result of translating the centroid).

This implementation could be additionally improved by employing a contour tracking algorithm that doesn't require dilation of the original object. Even though the structuring element used in the dilation process is small, its relative size to the training images puts it at risk for removing shape information. In particular, it could close regions that should normally be open, making the contour data unreliable. The contour tracking algorithm in this work could be modified to work with "thin" contours, or more sophisticated techniques such as "snakes" could be employed.

A known issue for this implementation of the centroid-distance function is that *every* discrete jump between pixels is assumed to occur with a constant timestep. This is accurate when contour pixels are directly adjacent to each other, but diagonally neighboring pixels are separated by a unit timestep multiplied by $\sqrt{2}$. By assuming that all pixels are separated by a unit timestep, we effectively "squish" the distance function together for times corresponding to regions of neighboring pixels on the contour with diagonal adjacency. A possible way to overcome this is to combine the polygonal approximation with the centroid distance function so that the contour can be easily broken up into segments with equal length.



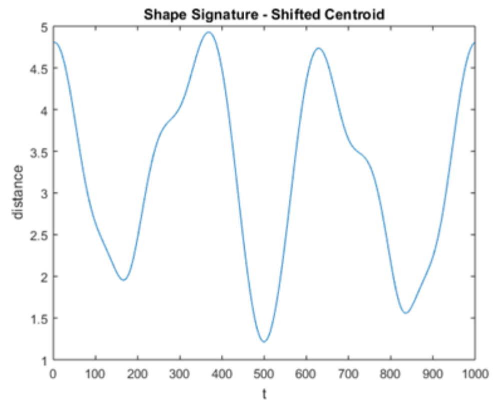


Figure 10
Top – shape signature of a closed contour with a centered centroid.
Bottom – shape signature of a closed contour with the centroid shifted slightly.

VII. References

[1] Y. Hu and Z. Li, "An Improved Shape Signature for Shape Representation and Image Retrieval," *Journal of Software,* vol. 8, no. 11, 2013.

[2] A. Robert, "Fourier Series of Polygons," *The American Mathematical Monthly,* pp. 420-428, 1994.

[3] G. H. Granlund, "Fourier Preprocessing for Hand Print Character Recognition," *IEEE Transactions on Computers,* vol. 21, no. 2, pp. 195 - 201, 1972.

[4] D. Zhang, *A Comparative Study on Shape Retrieval Using Fourier Descriptors with Different Shape Signatures,* Churchill.

[5] C. T. Zahn and R. Z. Roskies, "Fourier Descriptors for Plane Closed Curves," *IEEE Transactions on Computers,* Vols. C-21, no. 3, pp. 269-281, 1972.

[6] C. C. Lin and R. Chellappa, "Classification of Partial 2-D Shapes Using Fourier Descriptors," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vols. PAMI-9, no. 5, pp. 686-690, 1987.