VIRTUAL SHAPE RECOGNITION USING LEAP MOTION

David Lavy and Dung Pham

Boston University
Department of Electrical and Computer Engineering
8 Saint Mary's Street
Boston, MA 02215
www.bu.edu/ece

May. 03, 2015

Technical Report No. ECE-2015-03

Contents

List of Figures

1 Introduction

Leap Motion is a new gesture-capturing sensor hardware system that can take hand and finger gestures as input at a very high accuracy level compared to other devices such as Microsoft Kinect. Its capabilities give us a big range of applications that it can support. In this project, we want to extend Leap Motion so that it can recognize hand drawing gesture of numerical letters and output the corresponding values. This new function can replace traditional keyboards or other inputting methods in some human-computer interaction systems where accessing to a physical keyboard is forbidden.

2 Literature Review

Numerous papers have been written addressing handwriting recognition using image processing. A comprehensive review of this approach can be found in [1]. In different stages of the project, many approaches can be used independently. First, to transform 3D input data to 2D, many methods are available. *RANSAC* is one of the potential candidates for such task. However, according to [2], *RANSAC* runtime grows very fast if there are more input sample data, thus to have a faster running time while keeping appropriate accuracy level, we choose *least square fitting method* as the final decision, which has faster computing time.

Secondly, in our work, we used the approach mentioned in the paper called online handwriting recognition, which recognizes the characters while they are being written. Several approaches are identified in [1], and after discussions we decide to use shape-descriptor features. Shape-descriptors are a popular technique quite used for different applications and we concluded that this approach is computationally fast to achieve the real time classification we wanted to. We implemented this in the proposed system. In [3] several shape features extraction techniques are explained. In Figure 1 we have a quick overview of these techniques. After further review we decided to implement Fourier Descriptors as we only need to extract a very small percentage of this data to analyze the shape of the characters.

Finally, we need to select a good algorithm for classification. In [4] a set of classifiers are used for the MNIST dataset, a very popular and large dataset of handwritten digits and the results are very promising with the lower error rate being 1.19% using a 3-NN classifier. Other papers like [5] and [6] and tutorials online like [7] show also the efficiency of this classifier.

3 Problem Statement

Our approach to solving the problem (recognize a number drawing gesture from Leap Motion) will be broken into 4 main parts: Acquire Leap Motion's 3D data, then transform them into 2D data as pre-processing stage with *Least Square Fitting method*, extract features of that shape using *Fourier Descriptor Shape*, and finally, apply *K*-

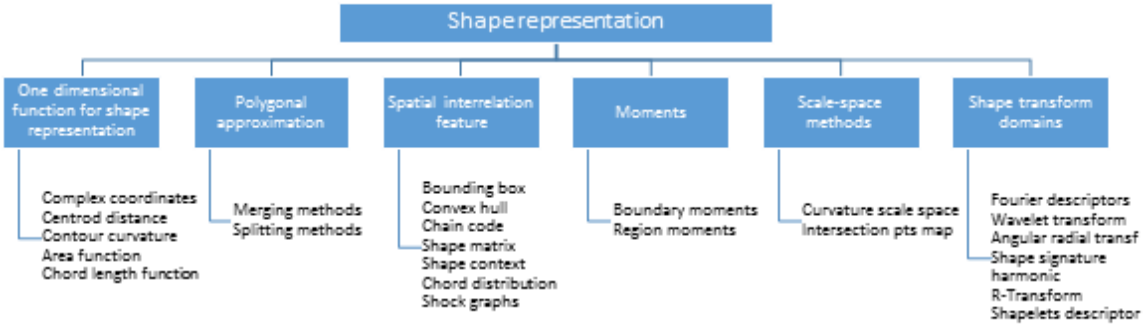


Figure 1: Overview of techniques for Shape representation

Nearest-Neighbor classification algorithm with those features to determine the closest number, which is the final output.

3.1 Acquisition

3D input data are easily acquired from Leap Motion with the provided Leap Motion SDK [8]. The provided SDK's data structure is very comprehensive with a large amount of information, in which the most useful data segment for this particular problem is finger's coordinates. We assumed that when a person draws something, he/she will likely use their index finger, either left or right hand. As a result, an index finger's tip bone is used as a point of data for each frame Leap Motion can detect. Note that: Leap Motion's capability of capturing image frames is substantially high, hence giving input data of each drawing sample up to more than 200 pixels per sample.

3.2 3D-2D Preprocessing

Least Square Fitting is the method that we used for this task. This method is actually performed in 2 phases of the solution. Firstly, in the training phase (collecting sample data set for later classification purpose) and secondly, transform real time testing sample when running the actual recognition program. In any of these two phases, a data transformation will be executed using LSF. As described in [9], the calculation

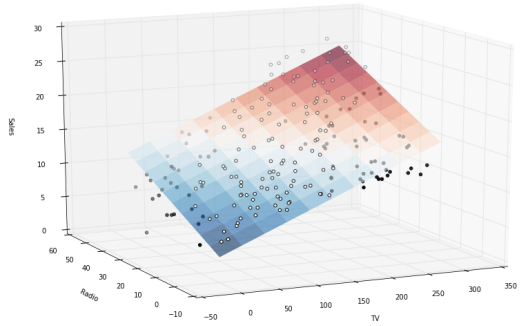


Figure 2: Least Square Fitting method visualization

of the best fit plane is derived using matrix operation with all 3D data point:

$$Z = \begin{pmatrix} \sum_{i=1}^{m} x_i^2 & \sum_{i=1}^{m} x_i y_i & \sum_{i=1}^{m} x_i \\ \sum_{i=1}^{m} x_i y_i & \sum_{i=1}^{m} y_i^2 & \sum_{i=1}^{m} y_i \\ \sum_{i=1}^{m} x_i & \sum_{i=1}^{m} y_i & \sum_{i=1}^{m} 1 \end{pmatrix}^{-1} \times \begin{pmatrix} \sum_{i=1}^{m} x_i z_i \\ \sum_{i=1}^{m} y_i z_i \\ \sum_{i=1}^{m} z_i \end{pmatrix} \quad (1)$$

z_1, z_2, z_3 are the normal vector of that best fit plane. By doing projections into 2 basic vectors of that plane with all 3D points, the new set of 2D points will be the coordinates of the new plane using those 2 basic vectors. In this phase of projection, a random vector is chosen to simplify the calculation and avoid unnecessary rotation of the plane, we assume that the input 3D point will be always in parallel with the ground, i.e., perpendicular with z axis. A detailed high level overview of how LSF is performed can be seen in Figure 2. Once the data has been processed to 2D, a normalization of the data has been applied using the formula $x_i = (x_i - mean(x))/std\_dev(x)$ and also for y so that all input sample have the same upper and lower bound, which make it easier for later feature extraction methods.

3.3 Feature Extraction

This feature extraction method was based on one of the ideas expressed in [10]. The idea is that Fourier descriptor is an efficient and effective way to record and compress a closed boundary because it only needs few low-frequency coefficients to reconstruct the contour of the original shape.

3.3.1 Basic concept of Fourier Descriptor of Shape (FDS)

FDS consists of basically two stages: constructing the signature shape and discrete fourier transform. An additional third step is included which is reconstructing the image, but this isnt a formal step for our project. Reconstruction is only done to show the efficiency of using FDS.

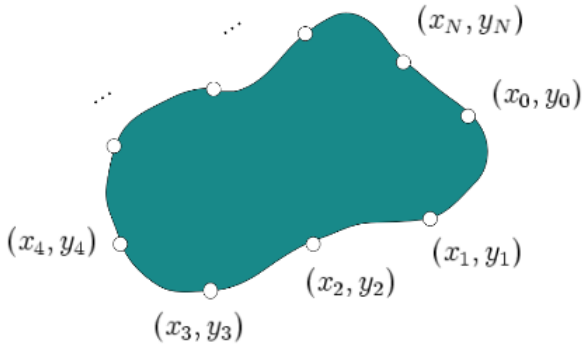


Figure 3: Set of points (x, y) on the contour are assumed to be ordered

- *First Step*: Signature Shape. After preprocessing our 3D data, we get a 2D contour based on a set of points (x_i, y_i), $\forall x = 1 \dots n$, an example is shown in Figure 3. We will extract each coordinate x and y separately and create a new signal U, this new signal is called the signature shape:

$$\mathbf{U_k} = \begin{pmatrix} x_0 + iy_0 \\ x_1 + iy_1 \\ \vdots \\ x_N + iy_N \end{pmatrix} \tag{2}$$

- *Second step*: Apply 1D DFT. The resulting DFT coefficients are called the Fourier Descriptors (FD) which describe the shape of our contour.

$$\mathbf{F}_\mu = FFT[\mathbf{U}] = \sum_{k=0}^{N-1} \mathbf{U_k} exp(-\frac{2\pi i}{N} k\mu) \tag{3}$$

  FD can be used as a representation of 2D closed shapes independent of its location, scaling, rotation and starting point. For example, we could use $M < N$ FDs corresponding to the low frequency components of the boundary to represent the 2D shape. The reconstructed shape based on these FDs approximate the shape without the details (corresponding to the high frequency components susceptible to noise).

  Although different variations exists on how to define the shape signature process [11, 12], we used the conventional Fourier descriptor as described in [13].

- *Third step*: Reconstruction of the contour. Only with a few FD coefficients it is possible to reconstruct the shape of the original contour by applying an inverse DFT:

$$\mathbf{U} = IFFT[\mathbf{F}_\mu] = \sum_{k=0}^{M-1} \mathbf{F}_\mu exp(\frac{2\pi i}{N} k\mu), M < N \tag{4}$$

3.3.2 Fourier Descriptors for Non-closed segments

The method described before is mainly used for closed boundaries. However, for a non-closed segments (something very frequent for character recognition), non-adjacents end points results in discontinuity and when the DFT is applied large oscillations will happen. This is explained due to the Gibbs effect. Because DFT is an implementation of the DSFT the method will work for a periodically extended signal but will present strange behaviors in the second case.

As an alternative, we corrected this method by using DCT rather than DFT on the signature shape. The approach is to extend the signal by a mirror and then take the DFT.

3.3.3 Efficient Feature extraction

When extracting feature for comparison we have to chose the ones that provides most of the information of the shape, and also, this features must have some essential properties such as:

- *Translation, rotation and scale invariance*: The location, the rotation and the scaling changing of the shape must not affect the extracted features.

- *Identifiability*: shapes which are found perceptually similar by human eye have the same features that are different from the others.

- *Noise resistance*: Features must be as robust as possible against noise, i.e., they must be the same whichever be the strength of the noise in a given range that affects the pattern.

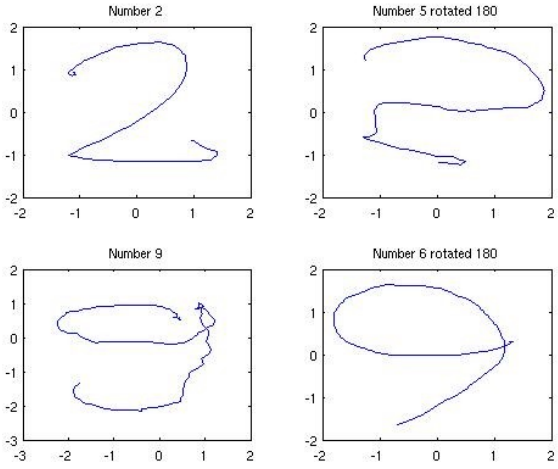
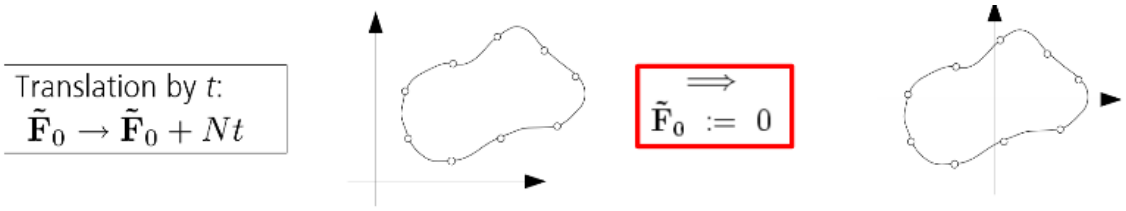


Figure 4: Similarity of numbers due to rotation

For this project we consider only translation and scale invariance from the first point. This is because the similarity of numbers when they are rotated 180 degrees. Figure 4 shows how the rotation of two numbers will give similar shape for another number.

1. *Translation invariance*: It is proved that DC gain contains all the information about rotation. To achieve this invariance we only need to set that value to 0



2. *Scale invariance*: Different people write in different sizes, a need to standardize the size of the contour is achieved by normalizing the FD coefficients by dividing all of them with a constant number. Assuming that the contours are traced counterclockwise and describe a non-zero area, we can rely on the face that the second Fourier descriptor is nonzero. Therefore we can divide all Fourier descriptors by the magnitude of the second Fourier descriptor to obtain a scale invariant vector.

3. *Noise resistance*: Usually when people write using Leap has a shaky start and end; also some people takes longer to write the number which will introduce noise in the whole contour. One approach to reduce noise is to take out the first and last 20 points of the set of data, as they are very likely to contain noise.

Once we finished the feature extraction, we will have one set of complex features that we will separate and put them together as a joint vector. This will be our final vector of features that will serve for our classification algorithm that will be explained next.

3.4 Classification algorithm (KNN)

kNN is a non parametric lazy learning algorithm. By parametric we mean that it does not make any assumptions on the underlying data distribution. This can be really helpful, because in the real world practical data does not obey theoretical assumptions. By lazy algorithm we mean that it does not use the training data

points to make any generalization. It also keeps all the training data during the testing phase.

It is also important to note that kNN assumes that the data is in a feature space. The data are feature vectors in a multidimensional space. Since the points are in feature space, they have a notion of distance This need not necessarily be Euclidean distance although it is the one commonly used.

Each of the training data consists of a set of vectors and class label associated with each vector. We are also given a single number "k". This number decides how many neighbors (where neighbors is defined based on the distance metric) influence the classification. This is usually an odd number, but mostly depends on error and trial to find the optimal value of k. If $k = 1$, then the algorithm is simply called the nearest neighbor algorithm.

Nearest Neighbor

- Given query instance x_q, first locate nearest training example x_n, then estimate $\hat{f}(x_q) \leftarrow f(x_n)$

k-nearest neighbor

- Given x_q, take vote among its k nearest neighbors (if discrete-valued target function)
- Take mean of f values of k nearest neighbors (if real valued)
  $f(x_q) = \frac{1}{k} \sum_{i=1}^{k} f(x_i)$

In terms of the metrics, KNN can use a wide range of metrics. Specifically for our system which uses numeric features, the set of following can be used:

- L^n-norm $L^n(x_1, x_2) = \sqrt[n]{\sum_{i=1}^{\#dim} |x_{1,i} - x_{2,i}|^n}$, Euclidean ($L^2$-norm), Manhattan
- Normalized by: range, standard deviation

For our project we used NN and kNN depending on the number of feature vectors, and as metrics we used Euclidean distance. This distance is the one used by default in OpenCV.

4 Implementation

Our final implementations come in 2 programming languages: Matlab and C++ (see Appendix). Initially, Matlab is the main language to verify the functionality of all algorithms, then all were converted to a single program in C++ which can import all pre-processed 2D data by Matlab to become training data, capture input gestures using Leap Motion SDK data structure, running 3D-2D transformation, Fourier Desciptor Shape and finally KNN.

4.1 Acquisition phase

Leap Motion SDK in C++ provides a wide range of input data, the most interesting data that we decided to use is the tip of index finger coordinates. At every frame, the Leap Motion controller will invoke a data structure for using, we extract them by using this data structure:

`controller.frame().hands().leftHand().fingers().index().bone('Distal').nextJoint().x (y & z)`

and store them into a vector structure. After finishing inputting the gesture sample (signify by a stop command from the terminal), a `2d_transform` function will be performed with that vector.

4.2 `2d_transform` function

`2d_transform` function apply LSF algorithm, which is also implemented in Matlab for converting training data, to process real time data. All matrix operations are done without any extension library since 3D data is trivial and implementing natively 3D operation is much faster than a library, such as cross product, dot product, 3×3 matrix multiplication and inverse matrix. The final output of `2d_transform` will be another vector that is ready for later phase of processing. There are corner case where output data is reversed (negative values) which distort the image. A simple counting variable is used to keep track of such abnormalities, and a flipping operation is introduced to solve that problem.

4.3 FDS & KNN

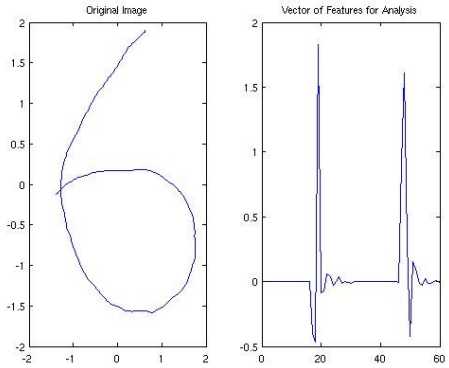


Figure 5: Number 6 and its corresponding feature vector

One of the first implementations for feature descriptors was based in the first experimental results we did using kNN. This was to treat the contour as an 60×60 pixel image. This was done by plotting a line approximation based on the points and

then calculate the feature descriptors. However, we found that it was much simpler to only works with the set of 2D points and use shape descriptors.

Overall, DFT and DCT functions were implemented in Matlab & C++ as well as a working KNN algorithm in C++.

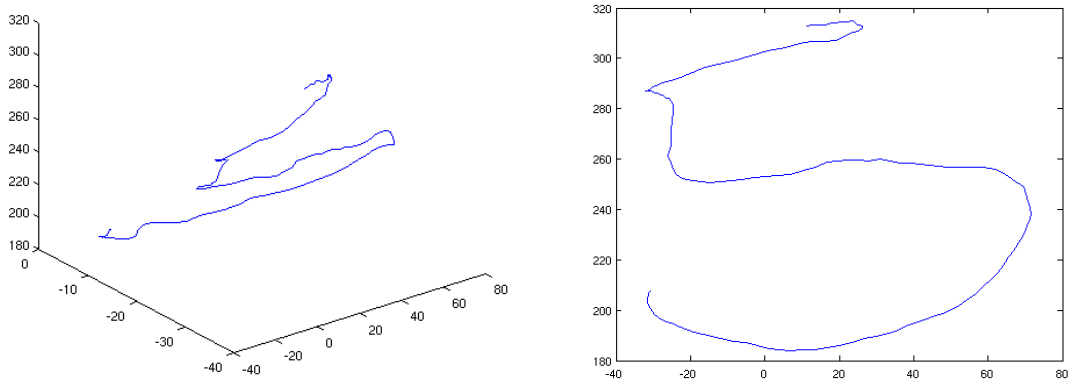
4.4 UI Program

Our final program in C++ was done and can behave independently, as well as having all mentioned functions which go from input Leap Motion data to output actual values. Generally, at start up, it will load in a set of training data (number of samples specified by user), then ask user to input a gesture, wait for a 'stop' signal (a 'Enter' key) and starts processing 3D to 2D data, extract features using FDS and finally apply KNN to calculate the output values. It is user friendly and functioning.

5 Experimental Results

5.1 Least Square Fitting

LSF performs fairly well in terms of running time in C++ (not restrain the processing of other incoming data frame from Leap Motion). After capturing 3D testing data, 2D data is processed instantly and ready for using Matlab implementation can finish processing all training data in reasonable time. In terms of quality, all sample results have good output, all corner cases are handled to avoid bad interpretations.



(a) 3D Input (b) 2D Output from LSF

Figure 6: LSF Performance

5.2 Fourier Descriptor Shape

In our project we found that we can reconstruct most of our digits by only using 2.5% of the FD. Figure 7 shows the original image, their x and y components as well as for the reconstructed image, and their complex components of the FDs, implemented in Matlab.

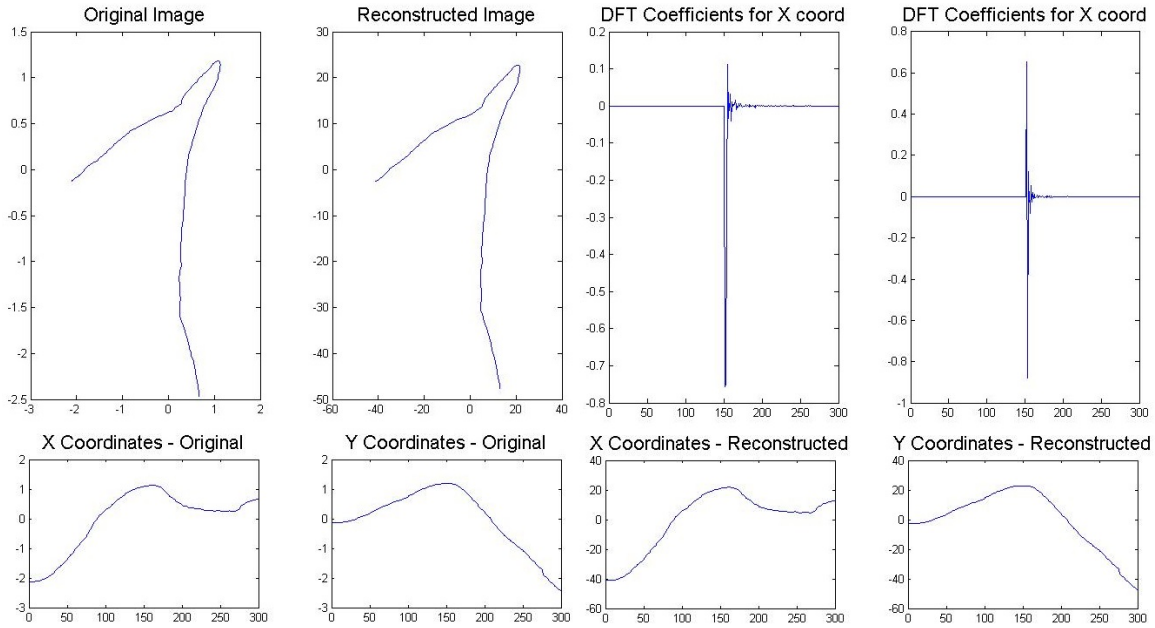


Figure 7: Detailed analysis of character 1 using 2.5% FD coefficients with DFT

As another way to improve feature extraction, Figure 8 shows an example of what happens when using DFT and Figure 9 shows how the reconstructed shape doesnt have those oscillations at the end by using DCT and also how our FD become symmetric.

5.3 KNN

As a first implementation of our system, we did an offline trial using the MNIST dataset. For this a set of 2500 characters were selected (250 characters per number). The feature vector was a 400×1 vector pixel values from the 20×20 pixel image from the dataset. The following confusion matrix shows the performance of this trial. Our performance test was based on $k = 5$ number of nearest neighbors, Eucledian distance metric and LOOCV (Leave One Out Cross Validation), this was also used for all of our other tests. Figure 10 gives a basic result using that dataset.

For the project using real data taken from the Leap, we gathered real data from different people and divided it in two different testing sets. The first set was using a set of 40 samples (4 different people will write numbers from 0 to 9), using NN. The second set was 150 samples (15 different people, 10 numbers each) and we used a 5-NN classifier. We show the confusion matrix for these tests in Figure 11 and 12.

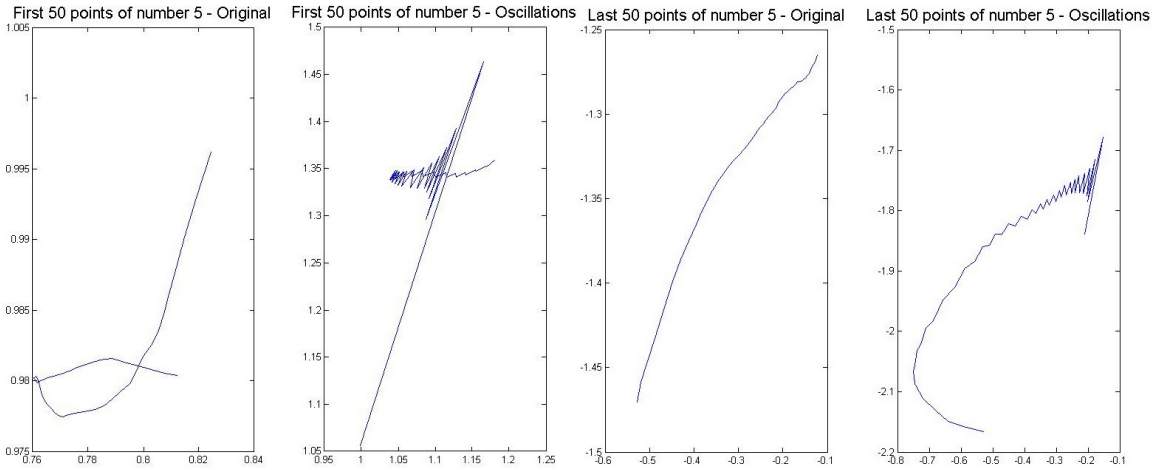


Figure 8: Comparison of reconstructed number using DFT

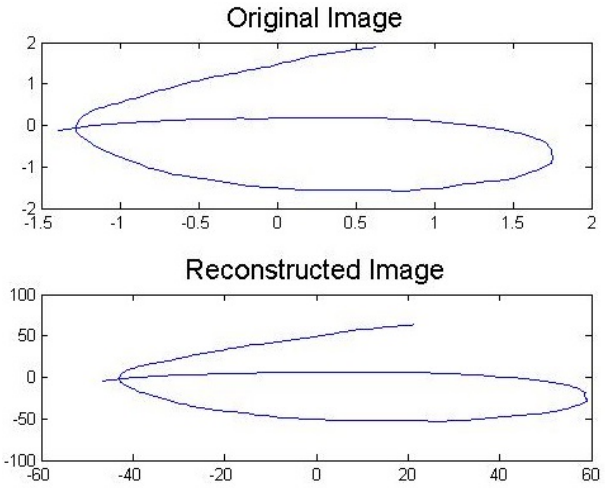
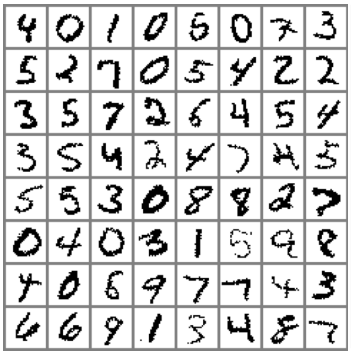


Figure 9: Comparison of reconstructed numbers using DCT



(a) MNIST Dataset

	1	2	3	4	5	6	7	8	9	0
1	95.25	0.29		0.20		0.31	0.97		0.50	
2	0.09	92.14	0.20				0.68	0.10		0.20
3			97.58				0.19	0.31	0.62	0.10
4		0.10		94.87				0.31	0.79	
5					96.66	0.32		0.21	0.50	
6	0.09	0.10		0.10	0.11	97.64			0.10	0.20
7		0.68	0.50				98.36	0.10		
8		0.10	0.20			0.10	0.10	97.03	0.50	0.20
9			0.20	0.92			0.68	0.31	95.39	
0		0.10			0.11	0.21		0.31	0.10	94.54

(b) Confusion Matrix

Figure 10: MNIST Dataset Testing

	1	2	3	4	5	6	7	8	9	0
1	73.26			5.23			12.11			
2		75.12			2.25			5.55		
3	1.11		64.25					13.25	11.25	
4				78.58						
5		2.25			82.15					
6	0.01					59.32			3.31	10.12
7							74.84			
8			2.25		5.11			80.63		
9						4.55			77.25	
0			1.58			6.30				79.21

Figure 11: Confusion Matrix for 40 Samples, Euclidean distance, $k = 1$(NN)

	1	2	3	4	5	6	7	8	9	0
1	24.25		8.12	5.23		3.21	8.54		6.65	
2		25.45		6.65	8.58		7.78	2.25	9.65	5.25
3	8.85		19.65		5.32		5.54		10.45	
4	5.23		3.36	31.56		8.68		6.65		4.28
5		10.76			36.58		5.55			
6	5.35		9.32			28.21		10.02		19.65
7		6.22		8.29	6.65		23.88		9.25	7.45
8		9.87	6.89		12.85	11.25		25.87	7.85	
9			6.65		8.85	10.25			12.32	10.66
0			15.21		8.65	12.25		6.94		10.27

Figure 12: Confusion Matrix for 150 Samples, Euclidean distance, $k = 5$(NN)

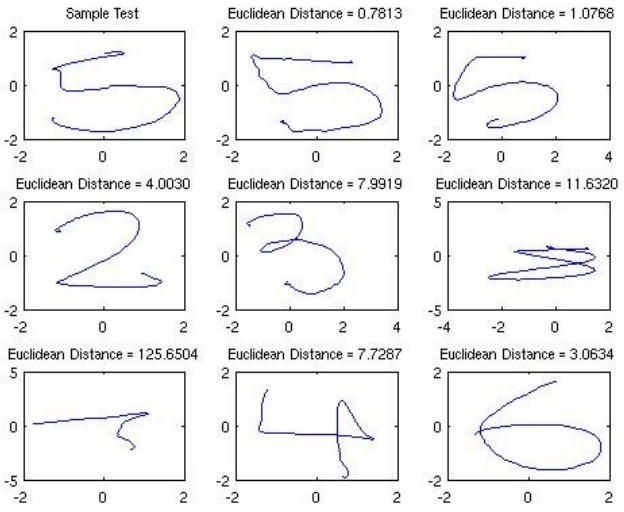


Figure 13: Euclidean distances for number 5 with a small set

For a good visualization of how the systems performs, we show a small example of the metrics between a 5 and other numbers, and highlight the smallest distances that will classify 5. From Figure 13, we can see that the smallest distances happens

to be for all the numbers 5, this will make our classifier work good.

6 Conclusion

In conclusion, we have successfully implemented independent modules (in Matlab and C++) that can perform assigned functions with determined results. Least Square Fitting method can transform 3D data in reasonable time and with reliable results for both training dataset and real time testing data. The Fourier descriptor method (only) was able to perform character recognition with a Detection rate of 75% by using NN in a 40 samples training set (4 per digit) and a Detection rate of 25% by using 5-NN in a 150 samples training (5 per digit). This was implemented in C++ using OpenCV to run it in real time. Additional to this, a simple MATLAB script was written to visualize and analyze in an easier way our data and their FD.

Due to time constraints, we were unable to add more features to improve the performance of our algorithm. Some potential set of additional features that we believe will improve this performance are:

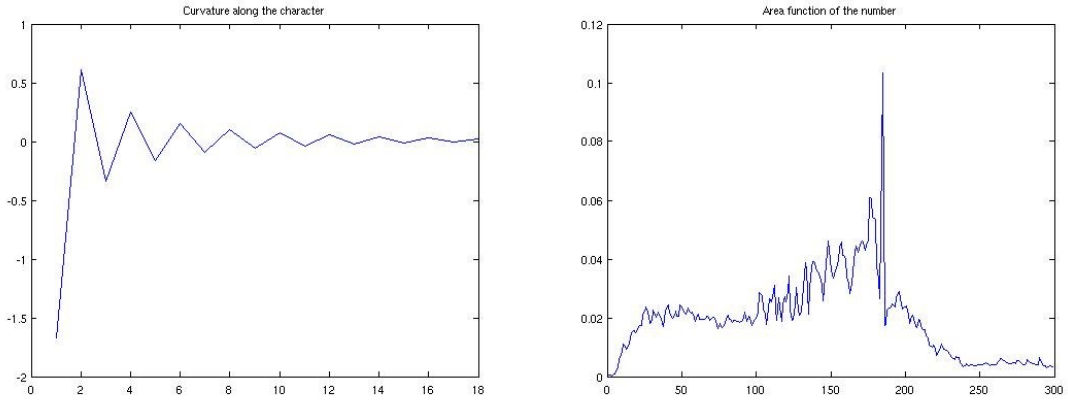
- *Contour curvature*: Curvature is a very important boundary feature for human to judge similarity between shapes. It also has saliend perceptual characteristics and has proven to be very useful for shape recognition [14]. In order to use $K(n)$ for shape representation, we quote the function of curvature, $K(n)$ as:

$$K(n) = \frac{\dot{x}(n)\ddot{y}(n) - \dot{y}(n)\ddot{x}(n)}{(\dot{x}(n)^2 + \dot{y}(n)^2)^{\frac{3}{2}}} \tag{5}$$

- *Area function*: When the boundary points change along the shape boundary, the area of the triangle formed by two successive boundary points and the center of gravity also changes. This forms an area function which can be exploited as shape representation. Figure 14 shows the contour curvature and area function for number 6. The area function is linear under affine transform. However, this linearity only works for shape sampled at its same vertices.

Also, we think that our metrics for the classification algorithm should be improved. OpenCV does the metrics under the hood and does not leave the option to change it. Another KNN implementations like [15, 16, 17] can be better as we can implement the metrics we desire.

Finally, a single user-friendly C++ program was done and functioning as planned, i.e., recognize Leap Motion hand gesture drawing and translate it into numerical values. However, due to unexpected large amount of hand gesture abnormalities, the accuracy level of results need to be improved by changing to a better algorithm in many phases of the project as mentioned above.



(a) Curvature function (b) Area function

Figure 14: Additional Features for FDS

References

[1] R. Plamodon and S. Srihari. Online and offline handwriting recognition: A comprehensive survey. 2000.

[2] Least squares. ransac. hough transform.

[3] Y. Mingqiand, K. Kidiyo, and R. Joseph. A survey of shape feature extraction techniques. 2008.

[4] M. Wu and Z. Zhang. Handwritten digit classification using the mnist data set. 2006.

[5] M. Bernard, E. Fromont, and A. Habard. Handwritten digit recognition using edit distance-based knn. 2012.

[6] Y. Lee. Handwritten digit recognition using knn, radial basis function and back-propagation neural network.

[7] OpenCV.

[8] Leap Motion Inc.

[9] David Eberly.

[10] J. Ding. Anti-symmetric fourier descriptor for non-closed segments. 2012.

[11] D. S. Zhang and G. Lu. Content based retrieval using different shape descriptors: A comparative study. 2001.

[12] D. S. Zhang and G. Lu. A comparative study of fourier descriptors for shape representation and retrieval. 2002.

[13] G. H. Grandlund. Fourier preprocessing for hand print character recognition. 2009.

[14] I. Siddiqi and N. Vincent. Combining contour based orientation and curvature feature for writer recognition.

[15] Weighted k nearest neighbor alorithms.

[16] Gpu-fs-knn: a software tool for fast and scalable knn computation using gpus.

[17] Mlpack.