Digits 3-D with Leap Motion

Juan Manuel Castillo Nievas (0518896), Cesar Lobo Martín (0519772), Guillermo Grijalvo (0519277). Lappeenranta University of Technology. December, 11th 2017

***Abstract*—Leap Motion is a sensor that is used for writing digits as free-hand strokes in the air with the index finger. In this project we are going to develop a classifier that is able to identify handwritten digits with this sensor using Matlab.**

***Index Terms*—Pattern recognition, Handwritten digits, Leap Motion, Preprocessing, Feature extraction, Classification, K-nearest neighbor**

I. Introduction

In order to develop our classifier, we have 1000 samples as training data, 100 samples for each digit (0-9). Each sample is stored into a Matlab file called stroke_DIGIT_NUM.mat, where DIGIT is the digit label and NUM is the number of the stroke. First, we have done a data preprocessing. Then, we have done a feature extraction. Finally, we have used the k-nearest neighbor method for doing the digit classification.

II. Data preprocesing

Data preprocessing is implemented in the file called *preprocessing.m*. We have ignored the third point (z axis) of each digit because it is not really necessary, because it does not have any effect when classifying the digits. Preprocessing follows 3 steps.

First step is related to the position of the digit [1]. This step is used for centering the position of the digit because sometimes we started writting the digits in a position that is not near to the center of the device. We can say that we start in a random position. The main idea is to calculate a find rectangular bound of the digit and translate it to the center of that bound. Due to the 2-Dimentions of our digits, we have a n x 2 matrix, where the first column is the x axis and the second column is the y axis. First, we calculate the minimum and maximum of the matrix [1], so they are the limits of our rectangular bound.. Then, we calculate the center point of our bound for both axis. Finally, we subtract the center point from each point of the stroke [1].

Second step is related to the scale of the digits [1]. We want to resize every digit, so that every digit has a 30x300 size. We have calculated a ratio in order to rescale our digit. First, we have calculated the width and height of our rectangular bound. Then, we have divided the size that we want by the width and height of our rectangular bound. Finally, we multiply the ratio that we have obtained by our digit points.

Third step is related to the amount of data. The number of points that Leap Motion can recognize depends on the velocity which the user writes with. Some points are very closely to each other, so these points are not going to be useful for us. We consider that a point is useless when the distance with the next point is smaller than the average of the distance between every consecutive point.

III. Feature Extraction

The second step is the feature extraction. It is implemented in the file called *feature_extraction.m*. The idea of this [2] is to divide the digit into 4 equal zones, so every zone has a 15x150 size. Then, we are going to see which are the directions that our digit follows. Every zone has different points. The line that connects two consecutive points can be classified into one of the directions shown in Figure1.

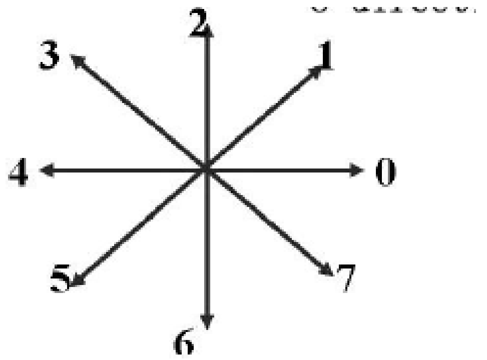

Figure 1. 8-direction

We have calculated the angle between every line that connects every consecutive points and the x axis. Once we have calculated the angle, we can say the direction that follows that line. We have followed the following rules:

Angles between 15º and -15º follow the direction number 0.

Angles between 15º and 75º follow the direction number 1.

Angles between 75º and 105º follow the direction number 2.

Angles between 105º and 165º follow the direction number 3.

Angles between 165º and -165º follow the direction number 4.

Angles between -165º and -105º follow the direction number 5.

Angles between -105º and -75º follow the direction number 6.

Angles between -75º and -15º follow the direction number 7.

The idea is to sum the angles that have the same direction in every zone. So we will have a 8x4 (*n* x *m*) matrix for every digit; each component will contain the total sum of the angles that follow the *n* direction in the zone number *m*.

IV. Classifier

The last step is to use a classifier in order to classify the digit, once we have preprocessed the data and extracted the features. We have selected the k-nearest neighbor classifier. This classifier has an easy implementation and it is non parametric classifier. It is implemented in the file called *knn.m*. The algorithm is described as follows:

1. For each training data, compute the substraction between the training data and the test data. Compute the mean of the result matrix.
2. Sort the means in ascending orders and select the *k* first neighbors, that is, select the training data that have the smallest difference comparing with the test data.
3. Select the most frequent class between the *k* neighbors.

So the main idea is to select the train digit that has the smallest difference between the direction that follows the written digit. For example, when writing number 2 we first start going to the right and then we go down. On the other hand, when writing number 5, first we go to the left, then we go down, and finally we make a semicircle. We follow different directions.

V. Results

We have used the *leave-one-out approach* in order to test our classifier. We use one data for testing and the rest of the data for training. We have done 20 iterations, 2 iterations for each digit and these are our results:

Iteration	**Percentage of success**	**Percentage of error**
1 (test data = 0)	91.6917	8.3083
2 (test data = 0)	91.6917	8.3083
3 (test data = 1)	91.7918	8.2082
4 (test data = 1)	91.7918	8.2082
5 (test data = 2)	91.8919	8.1081
6 (test data = 2)	91.8919	8.1081
7 (test data = 3)	91.9920	8.0080
8 (test data = 3)	91.9920	8.0080
9 (test data = 4)	92.0921	7.9079
10 (test data = 4)	92.0921	7.9079
12 (test data = 5)	92.1922	7.8078
13 (test data = 5)	92.1922	7.8078
14 (test data = 6)	92.2923	7.7077
15 (test data = 6)	92.2923	7.7077
16 (test data = 7)	92.3924	7.6076
17 (test data = 7)	92.3924	7.6076
18 (test data = 8)	92.4925	7.5075
19 (test data = 8)	92.4925	7.5075
20 (test data = 9)	92.5926	7.4074
21 (test data = 9)	92.5926	7.4074

We can say that we have a good classifier.

References

[1] Wen-Li Wang & Mei-Huei Tang, A Normalization Process to Standardize Handwriting Data Collected from Multiple Resources for Recognition. Conference Organized by Missouri University of Science and Technology 2015-San Jose, CA [online]. Available: https://ac.els-cdn.com/S187705091503001X/1-s2.0-S187705091503001X-main.pdf?_tid=3165a398-ddb6-11e7-ad17-00000aacb35e&acdnat=1512916204_33dabe12d995d05742e75862583c07ef

[2] Qian You, Xichang Wang, Huaying Zhang, Zhen Sun and Jiang Liu, Recognition Method for Handwritten Digits Based on Improved Chain Code Histogram Feature. School of Management Science and Engineering, Shandong Normal University, Lixia District, East of Culture Road 88,250014 Jinan, China.