Master Project Report:

Efficient Implementation of Shape Matching and Object Recognition Using Shape Contexts

Author: Andreas Kartawidjaja Advisor: George Kollios
Department of Computer Science
Boston University

1. Introduction

Objects that have dissimilar appearances are not necessarily different. The most common example would be human handwriting. A handwritten letter 'a' of one person is usually different from that of another person. Despite the difference, we are still able to identify that it is a letter 'a'. However, this dissimilarity poses problem for computers to identify similar objects using L2 norms comparison of pixel brightness values.

Figure 1. The above image is taken from [1] to illustrate shapes similarity that has different appearances.

[1] proposes a method for category-level recognition. This method can be broken down into three stages.

1. Solve the correspondence problem between the two shapes,
2. Use the correspondences to estimate an aligning transform, and
3. Compute the distance between the two shapes as a sum of matching errors between corresponding points, together with a term measuring the magnitude of the aligning transformation.

They provide a robust and simple algorithm for finding correspondences between shapes. In this algorithm, shapes are represented by a set of points sampled from the shape contours and shape descriptor is introduced to describe the coarse distribution of the rest of the shape with respect to a given point on the shape. In this case, solving the correspondence problem between two shapes would be equivalent to finding for each sample point on one shape the sample point on the other shape that has the most similar shape descriptor / shape context.

Given the correspondences at sample points of the two shapes, we then estimate an aligning transformation that maps one shape onto the other. By aligning shapes, we are able to define a simple measure of shape similarity. The dissimilarity between the two shapes can now be computed as a sum of matching errors between corresponding points, together with a term measuring the magnitude of the aligning transform. They have demonstrated object recognition in a wide variety of settings, such as the MNIST data set, which we will use as our data set in this project.

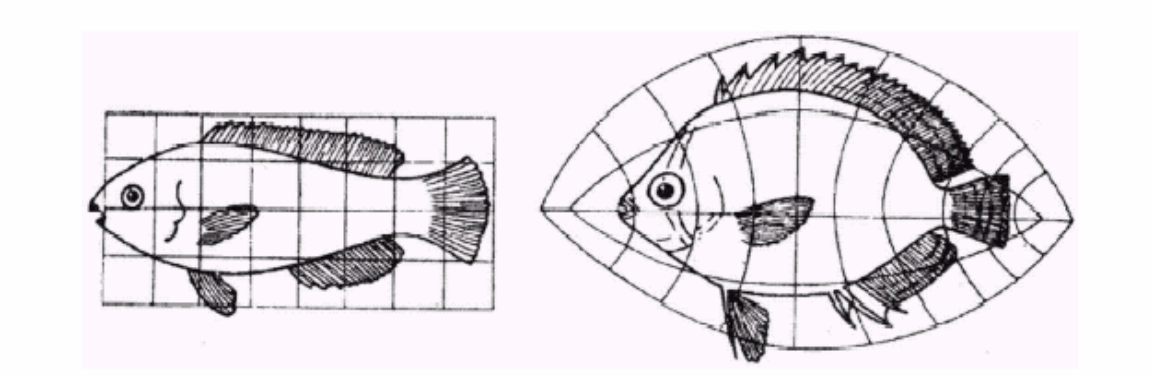

Figure 2. The above image is taken from [1] to illustrate coordinate transformations.

2. Our Contribution

In [1], they have implemented a system for shape matching and object recognition using shape contexts in MATLAB. The goal of this project is to improve the performance of the system developed by [1]. Since MATLAB is an interpreted programming language, there are overheads in the interpretation of programs. By eliminating interpretive overhead and managing memory efficiently, C/C++ code may run significantly faster than equivalent M-file functions. We can get significant speed increases especially for M-files containing for or while loops. Due to certain limitation, we will only implement those functions in C/C++ that take significant amount of execution time in the system. These functions include the Hungarian function and Shape Context computation function. The equivalent C/C++ functions can then be compiled to mex-file function that can be called by MATLAB, just like an M-file function.

We describe the algorithms for the functions that we implemented below.

2.1 Hungarian

Hungarian algorithm is used to solve bipartite graph matching in order to maximize similarities and enforcing uniqueness between the two shapes. Given the set of costs C_{ij} between all pairs of points p_i on one shape and q_j on the other, we want to minimize the total cost of one-to-one matching,

$$H(\pi) = \Sigma\, C(p_i, q_{\pi(i)}), \text{ where } \pi \text{ is a permutation.}$$

Algorithm:

Input : a square cost matrix
Output : optimal assignment and the cost of the optimal assignment

Step 0: *Create an nxm matrix called the cost matrix in which each element represents the cost of assigning one of n workers to one of m jobs. Rotate the matrix so that there are at least as many rows as columns and let k=min(n,m).*

Step 1: *For each row of the matrix, find the smallest element and subtract it from every element in its row. Go to Step 2.*

Step 2: *Find a zero (Z) in the resulting matrix. If there is no starred zero in its row or column, star Z. Repeat for each element in the matrix. Go to Step 3.*

Step 3: *Cover each column containing a starred zero. If K columns are covered, the starred zeros describe a complete set of unique assignments. In this case, Go to DONE, otherwise, Go to Step 4.*

Step 4: *Find a noncovered zero and prime it. If there is no starred zero in the row containing this primed zero, Go to Step 5. Otherwise, cover this row and uncover the column containing the starred zero. Continue in this manner until there are no uncovered zeros left. Save the smallest uncovered value and Go to Step 6.*

Step 5: *Construct a series of alternating primed and starred zeros as follows. Let Z_0 represent the uncovered primed zero found in Step 4. Let Z_1 denote the starred zero in the column of Z_0 (if any). Let Z_2 denote the primed zero in the row of Z_1 (there will always be one). Continue until the series terminates at a primed zero that has no starred zero in its column. Unstar each starred zero of the series, star each primed zero of the series, erase all primes and uncover every line in the matrix. Return to Step 3.*

Step 6: *Add the value found in Step 4 to every element of each covered row, and subtract it from every element of each uncovered column. Return to Step 4 without altering any stars, primes, or covered lines.*

2.2 Shape Context Matching:

In this function, we want to find the best matching point between point p_i on the first shape and point q_j on the second shape. We will have the distribution over relative positions of the sample points as the descriptor. For a point p_i, we compute a coarse histogram h_i, which is the shape context of p_i, of the relative coordinates of the remaining n–1 points,

$$h_i(k) = \#\{ q \neq p_i : (q - p_i) \in \text{bin}(k) \}$$

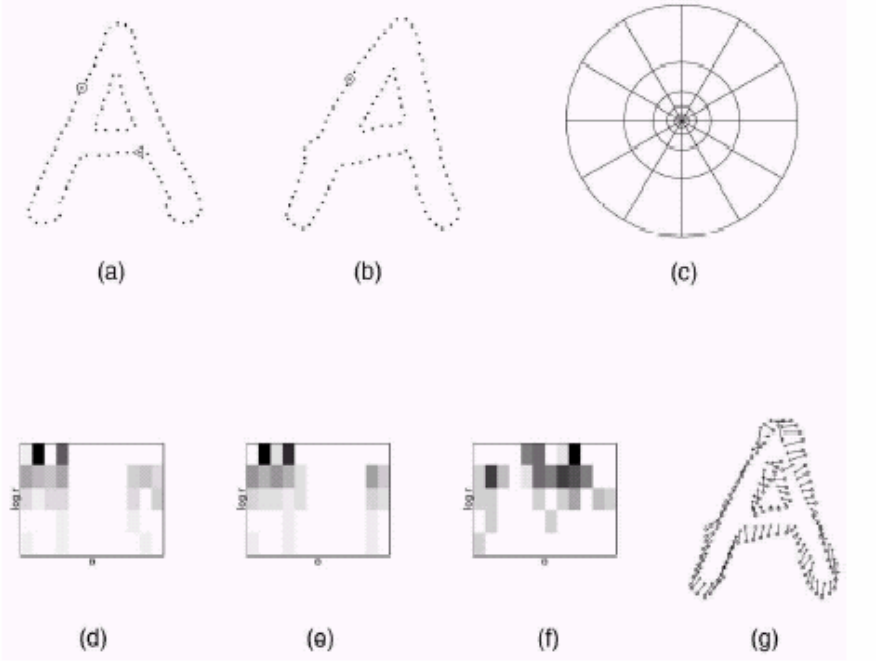

Figure 3. The image is taken from [1] to illustrate shape context computation and matching. (a) and (b) sample edge points of two shapes. (c) Diagram of log-polar histogram bins used in computing the shape contexts. We use five bins for log r and 12 bins for θ. (d), (e) and (f) Example shape contexts for reference samples marked by ○, ◇ and ◁ in (a) and (b). Each shape context is a log-polar histogram of the coordinates of the rest of the point set measured using the reference point as the origin. (Dark=large value). Note the visual similarity of the shape contexts for ○ and ◇, which were computed for relatively similar points on the two shapes. By contrast, the shape context for ◁ is quite different. (g) Correspondences found using bipartite matching, with costs defined by the χ^2 distance between histograms.

Algorithm:

Input : Coordinates of sample points, tangent theta for each point, and outliers
Output : Shape context for each point

1. *Compute the Euclidean distance matrix for each sample point to every other sample points and normalize the distance by the mean distance.*
2. *Compute the angle for each point relative to other points, where all angles are between 0 and* π
3. *Create bins / histograms that are uniform in log-polar space*
4. *For each point* p_i*, put point* q_j *in the appropriate bin in accordance to the distance and the angle between* p_i *and* q_j

Other than the Hungarian and Shape Context functions, we also implemented the histogram cost and the points sampling function since they took the longest running time.

3. Experimental Result

We conduct our experiments using MNIST database, handwritten digits, as our data set on Sun Ultra Enterprise 450 with four CPUs running the Solaris operating system.

Profile Summary

Generated 20-Apr-2004 19:56:46

Number of files called: 172

Filename	File Type	Calls	Total Time	Time Plot
demo_2	M-script	1	121.000 s	
hungarian	M-function	6	97.780 s	
hungarian/hmreduce	M-subfunction	853	81.910 s	
demo_2a	M-script	1	22.900 s	
hist_cost_2	M-function	12	7.950 s	
griddata	M-function	2	4.300 s	
newplot	M-function	178	4.140 s	
griddata/linear	M-subfunction	2	4.110 s	
get_samples_1	M-function	4	3.880 s	
newplot/ObserveAxesNextPlot	M-subfunction	178	3.880 s	
/cs/...ab/graphics/private/clo	M-function	60	3.810 s	
delaunayn	M-function	2	3.440 s	
im	M-function	8	2.700 s	
sc_compute	M-function	12	2.360 s	
colorbar	M-function	20	2.350 s	
Hungarian_mex	MEX-function	6	1.990 s	
hungarian/hminiass	M-subfunction	6	1.230 s	
dist2	M-function	68	1.210 s	
allchild	M-function	72	1.190 s	
/cs/...polyfun/private/qhullmx	MEX-function	2	0.950 s	
axis	M-function	92	0.920 s	
imresize	M-function	4	0.910 s	
hungarian/hmflip	M-subfunction	175	0.760 s	
sc_compute_mex	MEX-function	12	0.750 s	

Figure 4. Experiment run on MNIST data set sampled for 100 points for each object.

In figure 1, demo_2 is the original system implemented by [1] using MATLAB and it calls the M-file functions hungarian and sc_compute, whereas demo_2a is a slightly modified version of demo_2 that calls the mex-file functions Hungarian_mex and sc_compute_mex, which are implemented in C/C++. The performance difference between the two systems is evident that demo2_a runs significantly faster than demo_2 by a factor of $^{121}/_{123} = 5.26$, since mex-file functions have much better execution time than the corresponding M-file functions.

Then, we experimented using a query on a dataset of 30 handwritten digits from MNIST dataset. The following illustrates the improvement in the running time of the program.

Filename	File Type	Total Time
Total query running time	M-script	3434.61
Hungarian	M-function	3020.55
get_samples_1	M-function	67.99
sc_compute	M-function	63.91

Figure 4. Query using original MATLAB code

Filename	File Type	Total Time
Total query running time	M-script	624.66
Hungarian	M-function	83.11
get_samples_1	M-function	27.1
sc_compute	M-function	20.34

Figure 5. Query using MEX-files functions

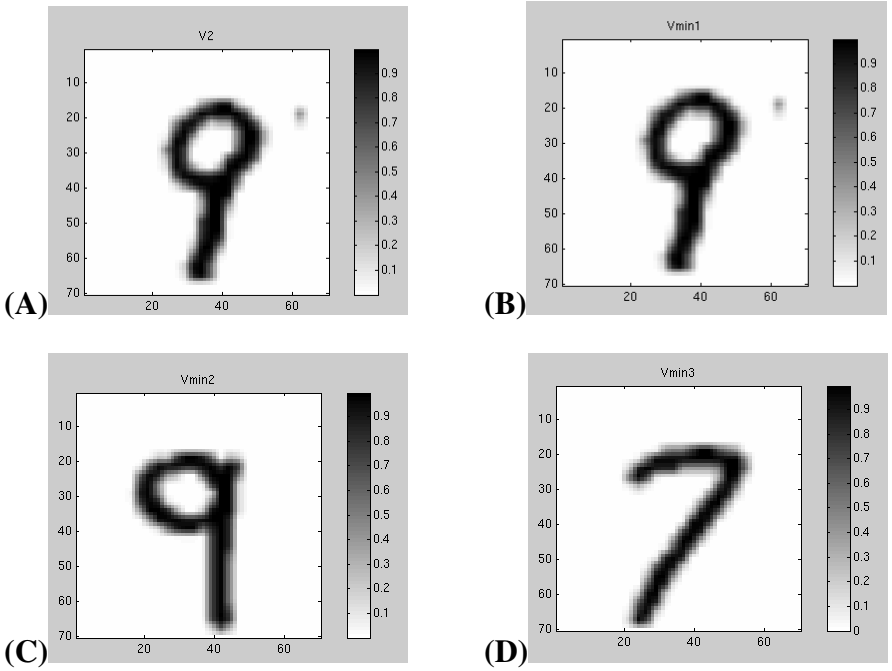

Figure 6. Experiment on 30 digits of MNIST dataset where (A) is the query and (B), (C) and (D) are the top 3 most-similar shape.

Improvement in performance is evident in Figure 4 and Figure 5, where the running time of the program with MEX-files functions run 5 times faster than the original program.

Acknowledgement

Hungarian algorithm – http://www.public.iastate.edu/~ddoty/HungarianAlgorithm.html

Reference

[1] S. Belongie,J. Malik, J. Puzicha"Shape Matching and Object Recognition Using Shape Contexts "