CSE585/EE555: Digital Image Processing II

Computer Project # 2:

Homotopic Skeletonization and Shape Analysis

Yanxi Yang, Jiuchao Yin, Hongjie Liu

Date: 02/17/2020

A. Objectives

- Implement homotopic skeletonization through repetitively thinning of target images by 8 structural element pairs.
- Perform shape analysis, including size distribution, pecstrum, and complexity.
- Realize pattern recognition by determining the smallest pecstral distance between the reference object and test object.

B. Methods

1. Flowchart (Figure 1):

- Convert the original RGB image to binary image.

1.1 For the homotopic skeletonization task (left part of flowchart)

- Build the structuring element set as given, with foreground and background as each pairs of structuring element set.
- Apply hit-or-miss transform to build thinning operation function. Iteratively run the thinning operation until we get the skeleton results.
- Show the superposition plotting with intermediate results as well as the final skeleton result.

1.2 For the shape analysis (right part of flowchart)

- Isolate distinct object from binary image using MATLAB built-in connected component label functions.
- Find the bounding box enclosing each distinct object.
- Build functions of size distribution, pecstrum, complexity and distance and run each function to compute respectively for each object.
- Observe and compare the result of complexity to find the most complex object.
- For the matching task, compare the pecstral distance and find the minimum to get optimal matching for each test objective and reference objective.

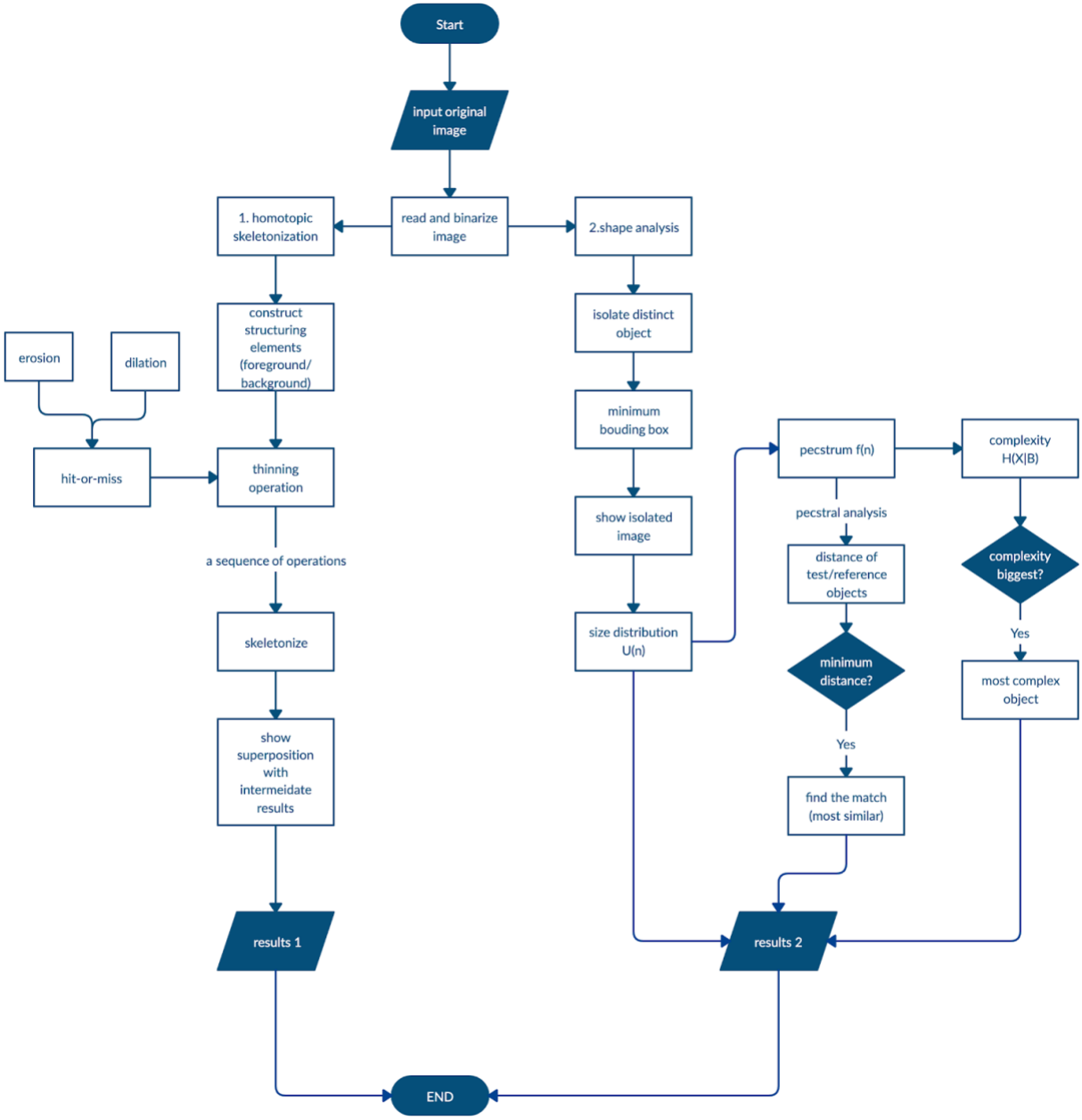

Figure 1. Flowchart

2. Algorithms

2.1 Homotopic Skeletonization

Homotopic skeletonization is based on an algorithm that implement the "grass fire" transformation to generate the homotopy of the target image. This algorithm is to perform thinning of target image X by 8 pairs of structural elements. Thinning is defined P&V eq. (6.10.1) as

$$X \bigcirc B_i = X - X \circledast B_i \tag{1}$$

B_i here represent a pair of structural elements B_{if} and B_{ib}, which probes the foreground and background respectively. Figure 2 demonstrates the 8 3 x 3 B_is we used in our project. For each Bi, ● represents the origin; the 4 shaded pixels are 1 in B_{if}, and the 3 white pixels are 1 in B_{ib}. The other 2 are always 0.

$X \circledast B_i$ is the hit-or-miss transformation defined as

$$X \circledast B_i = (X \ominus B_i^f) \cap (X^c \ominus B_i^b) \tag{2}$$

The algorithm for homotopic skeletonization is to iteratively implement thinning of the target object X by the 8 pairs of structuring elements. Each pair of the structuring elements strip off a layer of pixels from the target image. B_1 will thin off the top layer, B_2 thins off the top-right corner, B_3 thins off the right layer, etc. In this way, each iteration will burn off a layer of X. This process will continue until X does not change any more. The left over will be the skeleton based on "grass fire" algorithm.

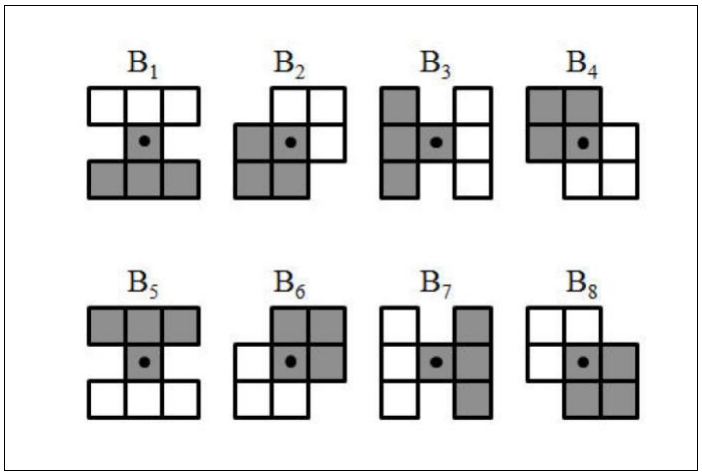

Figure 2 Structuring elements

2.2 Shape Analysis and Pattern Recognition

We use a 3 x 3 square B as the structuring element in both shape analysis and pattern recognition. As there are several objects in the same image, we first isolate each object by applying the MATLAB built-in functions bwconncomp and regionprops, and identify the bounding box for each object from the result returned by bwconncomp and regionprops.

2.3.1 Shape Analysis

Shape analysis is performed after all the objects are separated, including size distribution, pecstrum, and complexity. Size distribution is computed based on P&V eq. (6.11.3)

$$U(r) = m(X_{rB}),\ r \geq 0. \tag{3}$$

Here, rB is the structural element produced by Minkowski set addition of r number of B. X_{rB} is the result of X opening by rB. m (X_{rB}) is the total area covered by X_{rB}.

Pecstrum analysis is computed with the result of size distribution based on P&V eq. (6.11.7).

$$f(r) = \frac{-\frac{dU(r)}{dr}}{m(X)},\ r \geq 0. \tag{4}$$

In discrete space, $-dU(r)/dr = U(r) - U(r+1)$, and m(X) is the total number of pixels.

Lastly, complexity is computed with the result of f (r) based on Maragos-Shafer eq. (40):

$$H(X|B) = -\sum_{i=0}^{N} f(i)\log f(i). \quad (5)$$

The bigger the number of H(X|B) is, the more complex an object is. Therefore, the most complex object is the one that has the biggest H(X|B).

2.3.2 Pattern recognition

To determine which reference object best matches a test object, we calculate the pecstral distances for all of them. The reference object with the smallest pecstral distance will be considered to be the most similar one to the test object.

We need to first compute the pecstrum out based on equation (4) and (5) above. Pecstral distance is calculated according to P&V eq. (6.11.10)

$$d_i = [\sum_{n=0}^{N-1} C_n(f(n) - f_{R_i}(n)^2]^{1/2}. \quad (6)$$

$$\arg\{\min_i d_i\} = i_{min.} \quad (7)$$

i_{min} here is the reference object that most resembles the test object. Note that C_n here are weights chosen by ourselves depending on the components we would like to emphasize. We use different weights when we work on "match3" and "shadow1rotated" because we want to emphasize different components. For "match3", we think that the r ∈ [3,6] matters most for discriminating the four objects, so we set C_3 to C_6 to 50, and all the other C_n to 1 to emphasize components with r ∈ [3,6]. For "shadow1rotated", we consider r = [0, 5] to be most critical for discriminating the four characters. Therefore, we set C_0 to C_5 to be 100, and all the other C_n to 1. This is to emphasize the importance of components with r ∈ [0, 5].

3. Structure of code

3.1 main.m

main.m is the main function where the program starts. This function implements homotopic skeletonization, shape analysis, and pattern recognition. It calls 7 other functions: structure_E.m, skeletonize.m, sizeDistribution.m, Pecstrum.m, Complexity.m, distance.m, distance_cartoon.m.

3.2 Homotopic Skeletonization

3.2.1 structure_E.m

Function structure_E.m does not take any input. It simply creates the 8 pair of structural elements used for homotopic skeletonization, and outputs them as two cell arrays: one for foreground structural elements, and one for corresponding background structural elements.

3.2.2 skeletonize.m

Function skeletonize.m takes one 2-D binary image, one cell array of 3 x 3 origin-centered structural elements probing the foreground, and one array of corresponding 3 x 3 origin-centered structural elements probing the background. This function will return a cell array that contains resulted images after each thinning iteration. skeletonize.m will call thin.m for thinning operation.

3.2.3 thin.m

Function thin.m computes thinning operation. Its input includes a target 2-D binary image X, a 3 x 3 origin-centered foreground structural element B_f, and a 3 x 3 origin-centered background structural element B_b. It

outputs the thinning result of X by B_f and B_b. This function calls Hit-Or-Miss.m for the hit-or-miss transformation.

3.2.4 Hit_Or_Miss.m

Function Hit_Or_Miss.m performs hit-or-miss transformation. It takes three inputs: one 2-D binary image X, one 3 x 3 origin-centered structuring element B_f, and one 3 x 3 origin-centered structural element B_b. It will return the hit-or-miss transformation based on equation (2). This function needs to call erosion.m and dilation_s.m.

3.2.5 erosion.m

The purpose of this function is to compute the erosion of an input X with respect to a structuring element A. The input of erosion is a 2D binary image X and a structuring element A. The output is the result image X' after X eroded with respect to A. One restriction in this function is that the number of rows and columns in the structuring element A must be odd, and A must center at the origin. This function does not call other functions.

3.2.6 dilation_s.m

The purpose of this function is to compute the dilation of an input image X with respect to a structuring element A. The output is the result after X dilated by A. There is also a restriction in this function that the number of rows and columns in the structuring element A must be odd, and A must be origin-centered. This function does not call any other functions.

3.2.7 bitand_s.m

The purpose of this function is to compute the bit-and operation of two input matrices. This function takes two binary images with the same size as input, and the output is a matrix of the bit-and result of the two input images.

3.3 Shape Analysis and Pattern Recognition

3.3. 1 sizeDistribution.m

Function sizeDistribution.m computes size distribution of an input image X with respect to a 3 x 3 origin-centered structural element B. This function returns a vector of size information for different radii, and the maximum radius of X with respect to B. This functions calls erosion.m and dilation_s.m.

3.3.2 Pecstrum.m

Pecstrum.m takes a vector of size distribution information of an image X, and computes the Pecstrum of this image based on the size distribution information. No other functions are called inside this function.

3.3.3 Complexity.m

Complexity.m takes a vector of Pecstrum information of an image X, and returns a complexity score. This function does not call any other functions.

3.3.4 distance.m

Function distance.m is to calculated the pecstral distance of two objects. Its input should be two vectors containing the Pecstrum information of the two objects. It will output a distance value. Note that in this function, we set C_3 to C_6 to be 50, and all other C_n to be 1 to emphasize components with $r \in [3,6]$. This function is used for comparing objects in match1 and match3. It does not call other functions.

3.3.5 distance_cartoon.m

distance_cartoon.m has the same function with distance.m. The only difference is that in this function, C_0 to C_5 are 100, and all other C_n are 1. The purpose is to emphasize components with $r \in [0, 5]$. This function is used when we compare shadow1 and shadow1rotated. This function does not call other functions.

4. Results

4.1 Homotopic Skeletonization

(a) Task1: bear

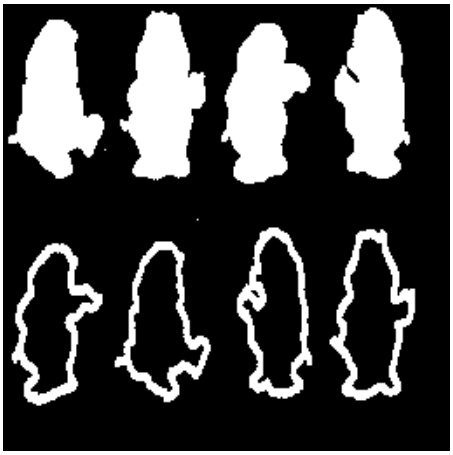

Figure 3 Original image: bear

First, we used the bear.gif image (Figure 3) to get the skeletons of objects in the figure. After the implementation of a sequence of thinning, which depends on Equation (1) and (2), we get the skeletons of those objects in the figure, which is shown in Figure 4. Figure 5 showed the intermediate results (X2, X5, X10) and the final result in the superposition for a better presentation (red shade). The structuring element is shown in Figure 2.

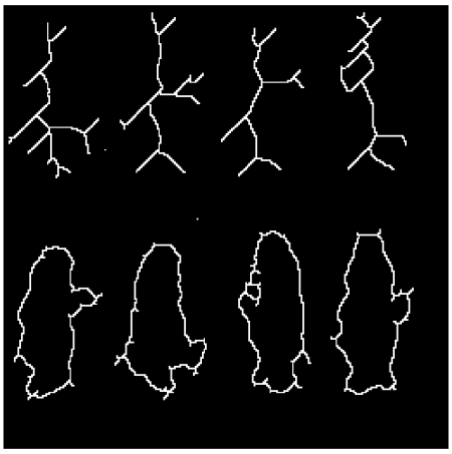

Figure 4 Skeletonized image

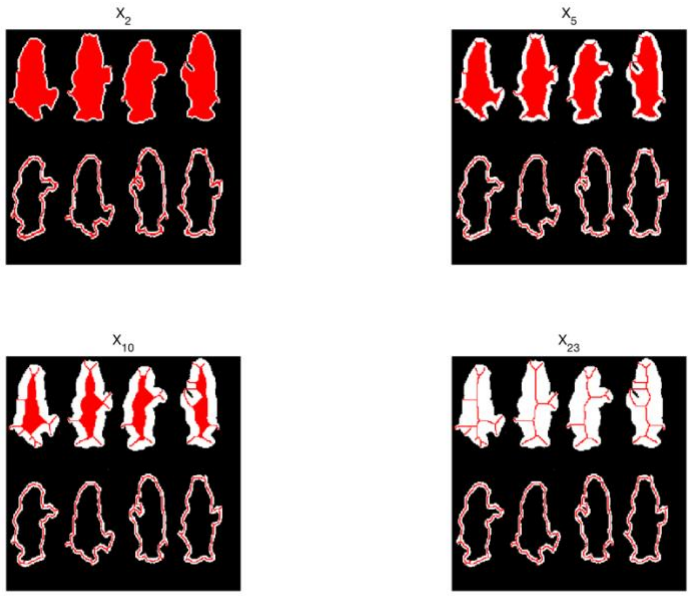

Figure 5 Intermediate results of thinning X2, X5, X10 and final skeleton (red shade)

(b) Task2: Penn256

Then we used penn256.gif to repeat the same operations. The original image is shown in Figure 6 and the intermediate results (X2, X5) and final results are shown in Figure 7. Because the content of this image is simpler than that of bear.gif, so the whole process is finished after 8 loops, and the intermediate results are only X2 and X5.

Figure 6 Original image: penn256

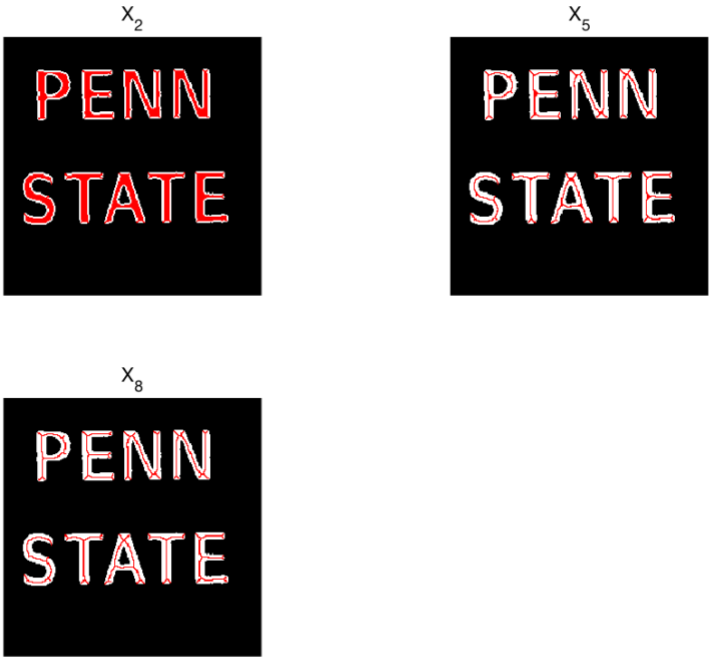

Figure 7 Intermediate results of thinning X2, X57 and final skeleton (red shade)

4.2 Shape Analysis

4.2.1 Part (a)

(a) Task1: match1

First, we split four objects in match1.gif and labeled them with four numbers. The original image match1.gif is shown in Figure 8 and the split result is shown in Figure 9.

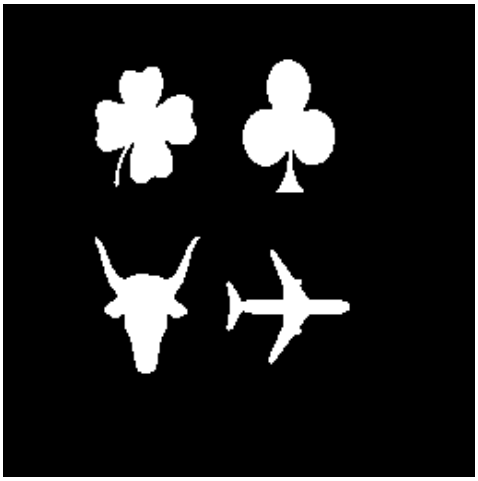

Figure 8 Original image: match1

Fig 9 Isolate and label each distinct object in match1

Then we compute the size distribution, pecstrum and complexity of each object. The computation result is shown in Table 1, Table 2, and Table 3.

Table 1 The computation results of size distribution: match1

Object	**r=0**	**r=1**	**r=2**	**r=3**	**r=4**	**r=5**	**r=6**	**r=7**	**r=8**	**r=9**	**r=10**	**r=11**
1	2125	2082	2075	2025	1995	1900	1864	1773	1697	746	651	
2	1511	1488	1332	1215	1155	1127	1081	1015	924	873	663	621
3	1003	990	850	647	346	225						
4	1945	1930	1900	1852	1773	1746	1734	1667	1590	1072	821	

Table 2 The computation results of pecstrum: match1

Object	**r=0**	**r=1**	**r=2**	**r=3**	**r=4**	**r=5**	**r=6**	**r=7**	**r=8**	**r=9**	**r=10**	**r=11**
1	0.0202	0.0033	0.0235	0.0141	0.0447	0.0169	0.0428	0.0358	0.4475	0.0447	0.3064	
2	0.0152	0.1032	0.0774	0.0397	0.0185	0.0304	0.0437	0.0602	0.0338	0.1390	0.0278	0.4110
3	0.0130	0.1396	0.2024	0.3001	0.1206	0.2243						
4	0.0077	0.0154	0.0247	0.0406	0.0139	0.0062	0.0344	0.0396	0.2663	0.1290	0.4221	

Table 3 The computation results of complexity: match1

Object	Complexity
1	1.5694
2	1.9642
3	1.6062
4	1.6386

From Table 3, we can conclude that the most complex object is steer (object 2).

(b) Task2: match3 shape analysis

Image match3.gif is the rotated version of math1.gif. In order to check which object in "match3" best matches each of the objects in "match1", we also split four objects in match3 first. The original image is shown in Figure 10, and the split objects are shown in Figure 11.

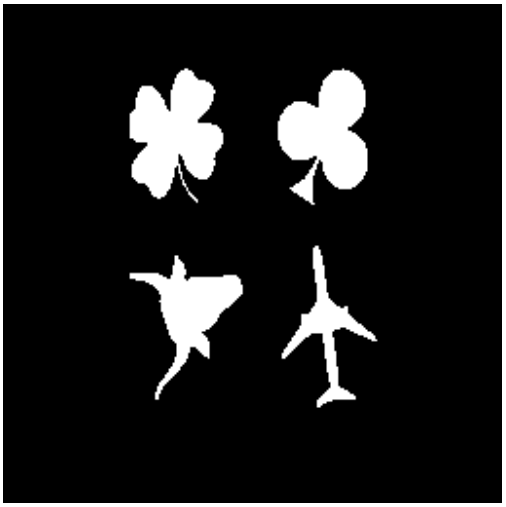

Figure 10 Original image: match3

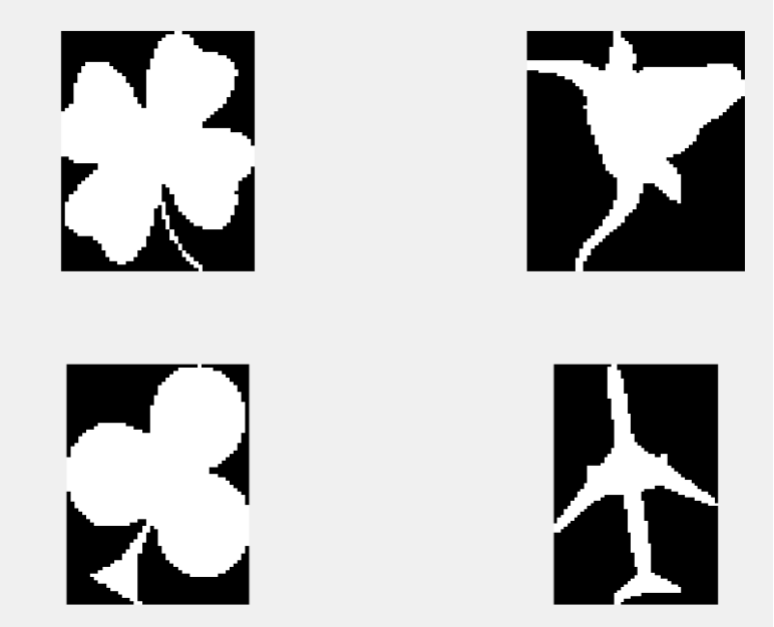

Fig 11 Isolate each distinct object in match3

Then we compute the pecstrum values of each object in match3 and the distances between match1 and match3. We set C_3 to C_6 to be 50, and all other C_n to be 1 to emphasize components with $r \in [3,6]$ as mentioned in section 3.3.4. Finally, we give the matches objects in match3 to the objects in match1 by finding out the smallest pecstral. Table 4 shows the values of pecstrum in match3.gif and Table 5 shows the distances between those objects. 12 shows the final matches of 4 objects. From tables below and Figure 12, we can easily conclude that our matches are correct. Note that the order of objects is changed, but they matched correctly to the objects in match1.

Table 4 The computation results of pecstrum: match3

Object	r=0	r=1	r=2	r=3	r=4	r=5	r=6	r=7	r=8	r=9	r=10	r=11	r=12
1	0.0156	0.0015	0.0141	0.0268	0.0331	0.0565	0.0599	0.3765	0.1374	0	0.2786		
2	0.0251	0.0666	0.1088	0.0408	0.0064	0.0165	0.0573	0.0787	0.0852	0.0408	0.0623	0.4116	
3	0.0099	0.0135	0.0115	0.0364	0.0245	0.0245	0.0213	0.0370	0.0562	0.3378	0.0115	0.0125	0.4034
4	0.0218	0.1848	0.4076	0.0253	0.1033	0.2572							

Table 5 The distances between objects in match1 and match3

	match1.clover	match1.sheer	match1.airplane	match1.spade
match3.clover	0.5593	0.7911	2.7494	0.5732
match3.sheer	0.9159	0.2860	2.6095	0.8405
match3.spade	0.7383	0.7699	2.8384	0.6719
match3.airplane	3.2818	2.9313	2.4399	3.3452

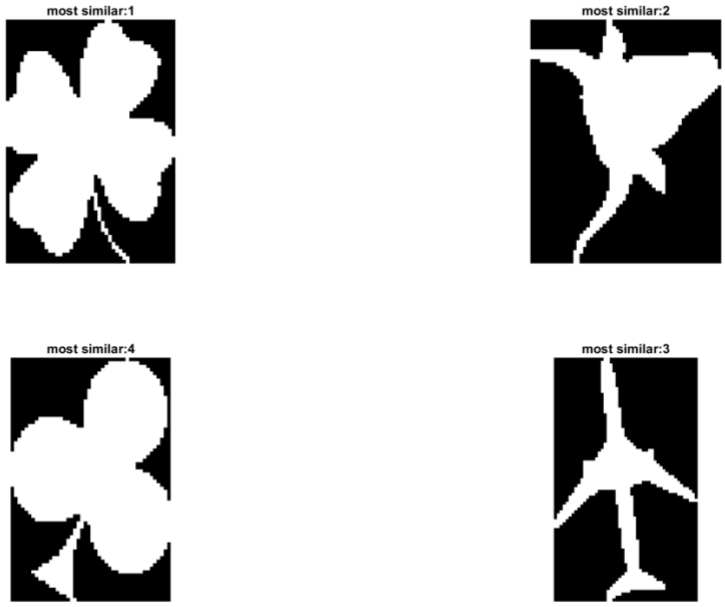

Figure 12 Final matches of objects in match3.gif

4.2.1 Part (b)

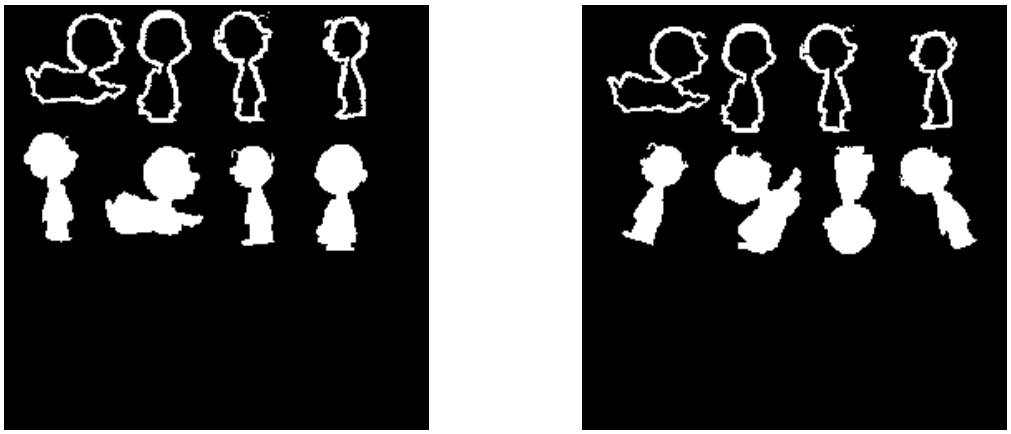

Figure 13 Original image Shadow1(left) and Shadow1rotated(right)

Figure 13 shows the original image of shadow1.gif (on the left) and shadow1rotated.gif (on the right). In this task, we only used four solid objects to complete the match. Four solid shapes in shadow1.gif are the referenced objects and four solid shapes in shadow1rotated.gif are the test objects. In order to do matching, we first split four shapes in shadow1.gif and shadow1rotated.gif. Then we compute pecstrums and distances between objects in two images. We set C_0 to C_5 to be 100, and all other C_n to be 1. The purpose is to emphasize components with $r \in [0, 5]$ as mentioned in section 3.3.5. Finally, we got the match results of those four shapes. Figure 14 shows the split of objects in two images. Table 6 shows the pecstrum value of four objects in shadow1.gif. Table 7 shows the pecstrum value of four objects in shadow1rotated.gif. Table 8 shows the distances between those objects in two images. Again, we matched the objects in shadow1rotated to the object in shawdow1 which has the shortest pecstral distance

(emphasized with underline in Table 8). Figure 15 shows the final matches of objects in shadow1rotated.gif, from which we could see we correctly matched each object to the correct ones in shadow1.

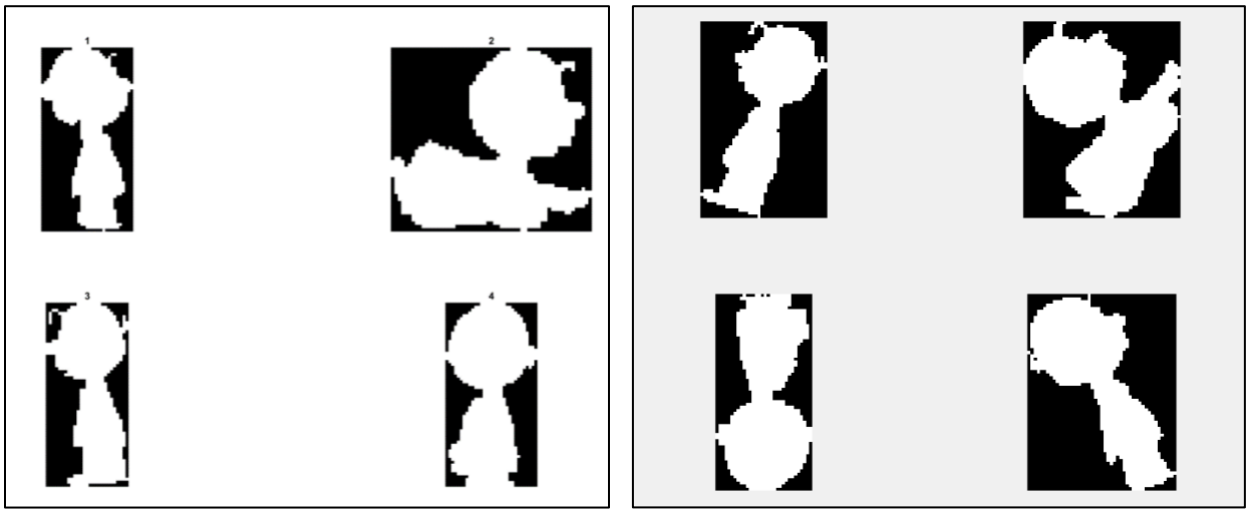

Figure 14 Split result of Shadow1(left) and Shadow1rotated(right)

Table 6 The computation results of pecstrum: shadow1

Object	**r=0**	**r=1**	**r=2**	**r=3**	**r=4**	**r=5**	**r=6**	**r=7**	**r=8**	**r=9**	**r=10**
1	0.0102	0.0102	0.0085	0.0529	0.0845	0.1826	0.1894	0.0256	0.0290	0.4070	
2	0.0118	0.0124	0.0399	0.0169	0.0281	0.0326	0.0837	0.0258	0.1612	0.2787	0.3090
3	0.0209	0.0262	0.0220	0.0377	0.0502	0.1444	0.3431	0	0.3556		
4	0.0046	0.0123	0.0046	0.0331	0.0293	0.0547	0.0847	0.0601	0.3151	0.0616	0.3398

Table 7 The computation results of pecstrum: shadow1rotated

Object	**r=0**	**r=1**	**r=2**	**r=3**	**r=4**	**r=5**	**r=6**	**r=7**	**r=8**	**r=9**	**r=10**	**r=11**
1	0.0214	0.0398	0.0183	0.0530	0.0571	0.2110	0.2596	0.3425				
2	0.0067	0.0333	0.0338	0.0283	0.0255	0.0255	0.0499	0.1592	0.2546	0.0211	0.0483	0.3189
3	0.0076	0.0030	0.0122	0.0327	0.0213	0.0708	0.0838	0.0586	0.3123	0.0305	0.3671	
4	0.0110	0.0245	0.0178	0.0440	0.0821	0.0863	0.2453	0.0135	0.0465	0.4289		

Table 8 The distances between objects in shadow1 and shadow1rotated

	shadow1.1	shadow1.2	shadow1.3	shadow1.4
shadow1rotated.1	0.6757	0.8460	0.5512	0.7690
shadow1rotated.2	0.9454	0.5726	0.5936	0.5811
shadow1rotated.3	0.9100	0.4553	0.6166	0.1535
shadow1rotated.4	0.2255	0.7676	0.6403	0.8235

Figure 15 Final matches of objects in shadow1rotated.gif

5. Conclusion

1) We got familiar with skeletonization and thinning operations and their close relationship, as well as their applications in shape analysis and pattern recognition through this project.

2) Also, we applied hit-or-miss function realized by dilation and erosion operations in project 1, which is also a deeper understanding of previous work.

3) We tried different C_n scalar parameters when running the distance.m and distance_cartoon.m function to get the optimal matching result.