## CSE585/EE555:  Digital Image Processing II

## Computer Project # 2:

## Morphological Skeleton and Shape Analysis

#### Sweekar Sudhakara, Savinay Nagendra, Nagarjuna Pampana, Prapti Panigrahi

#### Date: 02/26/2021

1. **Objectives**

The Objective of this project is two-fold. The first objective is to implement and understand the process of morphological skeletonization as well as reconstruction of the original image using the obtained skeleton. Skeletonization and Reconstruction use morphological operations like opening, erosion, dilation and set operations like union. The second objective is to understand and implement components of shape analysis – Size distribution, Pecstrum and Complexity of individual connected components (objects) in a binary image. Components of shape analysis provide description of shape features (perimeter, size, compactness, and circularity) of objects in the image. Pecstral analysis is further used for the application of object matching. The goal of this project is to gain better understanding of the above-mentioned concepts and achieve intended results of skeletonization, reconstruction, shape analysis and object matching through their implementation.

1. **Methods**

**Theory and Algorithms:**

The process flow followed in the project implementation (both parts) is as follows:

**PART 1: Skeletonization and Reconstruction**

**Top-level goals** of the method are:

1. Read the image and generate the structuring element.
2. Compute all finite skeletal components and their respective radius/homothetic.
3. Take union of skeletal components to obtain skeleton of the image.
4. Dilate individual skeletal components with their respective homothetic of the skeletal components.

**Details** of the method are:

1. Reading image and generating structuring element: Read the image and create the structuring elements namely, square, rhombus and VEC045.

Also note that for this project white pixels are treated as foreground while the black pixels are treated as the background of the image. For constructing the square, we need to create a array with all elements set to 1 (white). Similarly, for creating a rhombus, change the corners of the square into black pixels. For VEC045, create a square with all black pixels and then change pixels (2,2) and (1,3) into white color.

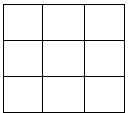
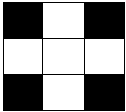
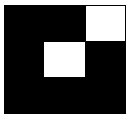
  

Figure 1: Structuring elements: square, rhombus and VEC045.

1. Compute skeletal components: The skeletal components can be computed as follows:

**(1)**

where Sn(X) indicates the nth skeletal component of image ‘X’, ‘B’ indicates the structuring element, the value of ‘n’ ranges from () and (*nB* is a structuring element of radius ‘n’ and *NB* is the largest homothetic of ‘B’ that fits inside of X).

In equation 1, is called the opening operation and is defined as which is a combination of erosion and dilation assuming ‘B’ is symmetric. The erosion and dilation are equivalent to Minkowski subtraction and addition when structuring element is symmetric. But, when the structuring element is not symmetric, performing erosion/dilation with a transpose of ‘B’ () will yield an equivalent result of performing Minkowski subtraction/addition.

For easy of computation, store eroded ‘X’ by structuring element ‘B’ in a dynamic variable ‘X1’ which is eroded by ‘B’ once per iteration. Consider, . Using Associative law, we can write it as . This approach saves us from doing multiple erosion operations for each iteration.

The radius of the skeleton is defined as

Store all the individual skeletal components along with their corresponding radius. These will be helpful in reconstruction of the image. Also record the number of iterations so we know the largest homothetic that fits inside of X.

1. Take union of all individual skeletal components to get the skeleton of ‘X’ following the equation:

Compute union within the iteration loop used for computing skeletal structures as it will not require us make a bigger union at the end.

1. To compute reconstruction of image ‘X’ from skeletal components follow the equation given by

and for partial reconstruction, the equation can be written as follows:

To compute the union, retrieve individual skeletal components stored previously and dilate with nth homothetic of ‘B’, where ‘n’ is the corresponding radius of the skeletal component.

**PART 2: Shape Analysis**

**Top-level goals** of the method are:

1. Compute Minimum Bounding rectangle (MBR) for extracting/isolating all objects (connected components) in a given image.
2. Compute the Size distribution for every object in a given image using the operation of opening with multiple dilations of a structuring element.
3. Compute Pecstrum for every object by using the computed size distributions.
4. Compute Shape Complexity (entropy) for every object by using the computed pecstrums.
5. Use Pecstral analysis for Object Matching between a reference and a test image.

**Details** of the method are:

1. Compute Minimum Bounding Rectangle (MBR):

MBR (bounding box) is an expression of the maximum boundaries of a 2-D object. Connected-component labeling is an algorithmic application of graph theory which is used to detect connected components in binary digital images. In our application, we compute the MBR for every object in the given image to isolate/extract individual connected components for further processing. The function *min\_bounding\_box()* in our code base is called to perform the operation of object extraction. We first call the *bwlabel()* function in MATLAB, which is a connected component labeling function. It takes a 2-D binary image as input and returns a *label matrix* containing labels for 8-connected components present in the image and the *number of connected objects (n)* found. The pixel clusters of each object are found from the matrix using the *find()* function in MATLAB. The minimum and maximum pixel coordinates (x\_min, y\_min) and (x\_max, y\_max) respectively are obtained, which form the bounding box around each object. For extracting a particular object, only the foreground pixels inside the MBR of that object are retained while the rest of the pixels are made into background pixels. This way, we get *n* images of original size but each containing one distinct object.

1. Compute Size Distribution:

This is one of the fundamental approaches to morphological image analysis where the transformed image is measured by its area. From P&V 6.11, there exists a real-valued function , called the size distribution, that gives the measure of . is the family of openings of image with a structuring element of size (radius).

From a physical point of view, the size distribution of gives the area of which can be covered by a disk of radius , when this disk moves inside the image . The size distribution is inversely proportional to the radius, which makes the function strictly monotonic. The function *compute\_size\_distribution()* is used to calculate the size distribution of each object extracted using MBR as mentioned in the above step. With a structuring element of size 3x3 (radius = 1) containing foreground pixels, we compute . Further, we perform the dilation operation on with itself times. At each step, we obtain the structuring element , where . The image of every object was opened iteratively by with being incremented in steps of 1. For all our experiments we used n = 14 as the upper bound for the number of iterations. This was chosen based on our observation that the area obtained above the radius of 12 was zero for all objects. The two additional iterations were performed to showcase the trend of saturating area, which settles at zero. The area is computed by counting the number of foreground pixels in the opened image.

1. Compute Pecstrum:

The normalized negative derivative of the size distribution is called the pattern spectrum or pecstrum. The digital version of the pecstrum is given by P&V 6.11.8:

Here, is the total area of the original object, and are size distributions computed with structuring elements of radius and respectively. Pecstrum is computed using the function *compute\_pecstrum().*

1. Compute Shape Complexity:

Shape complexity is also known as the entropy of an object. The equation for calculating the shape complexity of an object is provided in L6-15 and Maragos-Schafer Eqn 40:

where is the pecstrum with radius . The calculated entropy can be viewed as the average roughness of image relative to the structing element . This is because, the entropy quantifies the shape-size complexity of by measuring its boundary roughness averaged over all depths that can reach. This calculation is implemented in the function *compute\_complexity()*.

1. Object Matching:

From P&V 6.11, Pecstral analysis can be used for pattern recognition and object matching. If is a reference pecstrum and is the pecstrum of a new test image , from P&V 6.11.10, the distance between the reference image and test image is given by:

This distance is used to determine if the new object coincides with a reference pattern. The unknown object in the test image is matched with an object in the reference image with which its distance is minimum. The weights are used to emphasize specific parts of pecstrum differences. For large distances between the two pecstrums are to be emphasized, P&V 6.11.11 suggests choosing weights as follows:

Here, the hyperparameter is application dependent. According to the above equation, in our algorithm, we decrease the weights monotonically with the increase in radius of the structuring element. The idea behind this intuition is that more shape information is lost every time our image is opened by a structuring element of increased radius. We calculate the distance matrix using the function *compute\_distance()*.

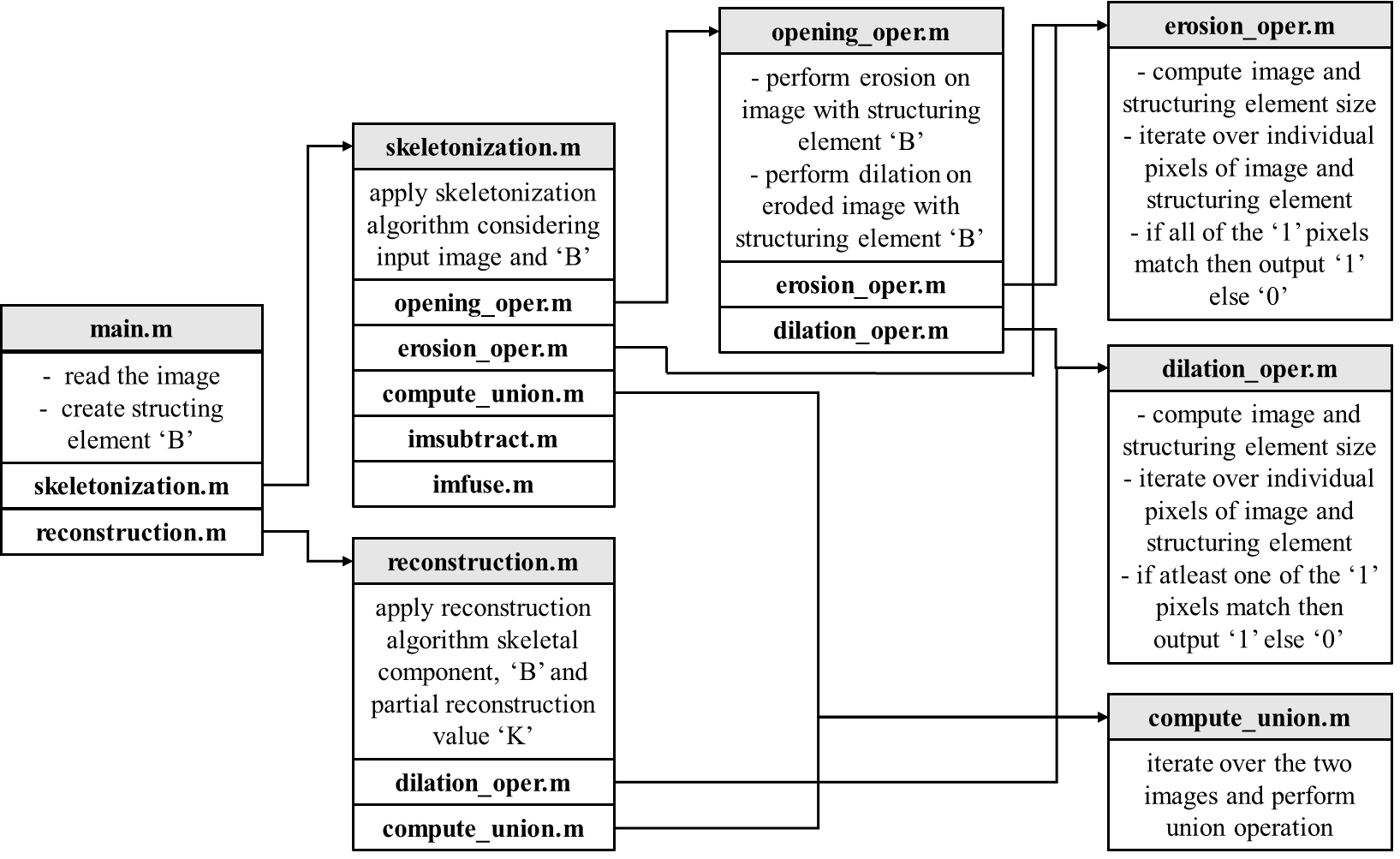
**MATLAB:**

The MATLAB implementation followed in the project (both parts) is as follows:

**PART 1: Skeletonization and Reconstruction**

The files corresponding to Skeletonization & Reconstruction (PART 1) can be found in the folder **EE555\_Project\_2/Part\_1/**. The main code file for the PART 1 is **main.m**.

In Figure 2, we can observe the flowchart followed in the MATLAB implementation for PART 1: Skeletonization and Reconstruction.



**Figure 2**: Flowchart depicting the code implementation of PART 1: Skeletonization and Reconstruction (**main.m**).

**NOTE**: In order to obtain the results as shown in the next section kindly execute/run the **main.m** code file located in **EE555\_Project\_2/Part\_1/**.

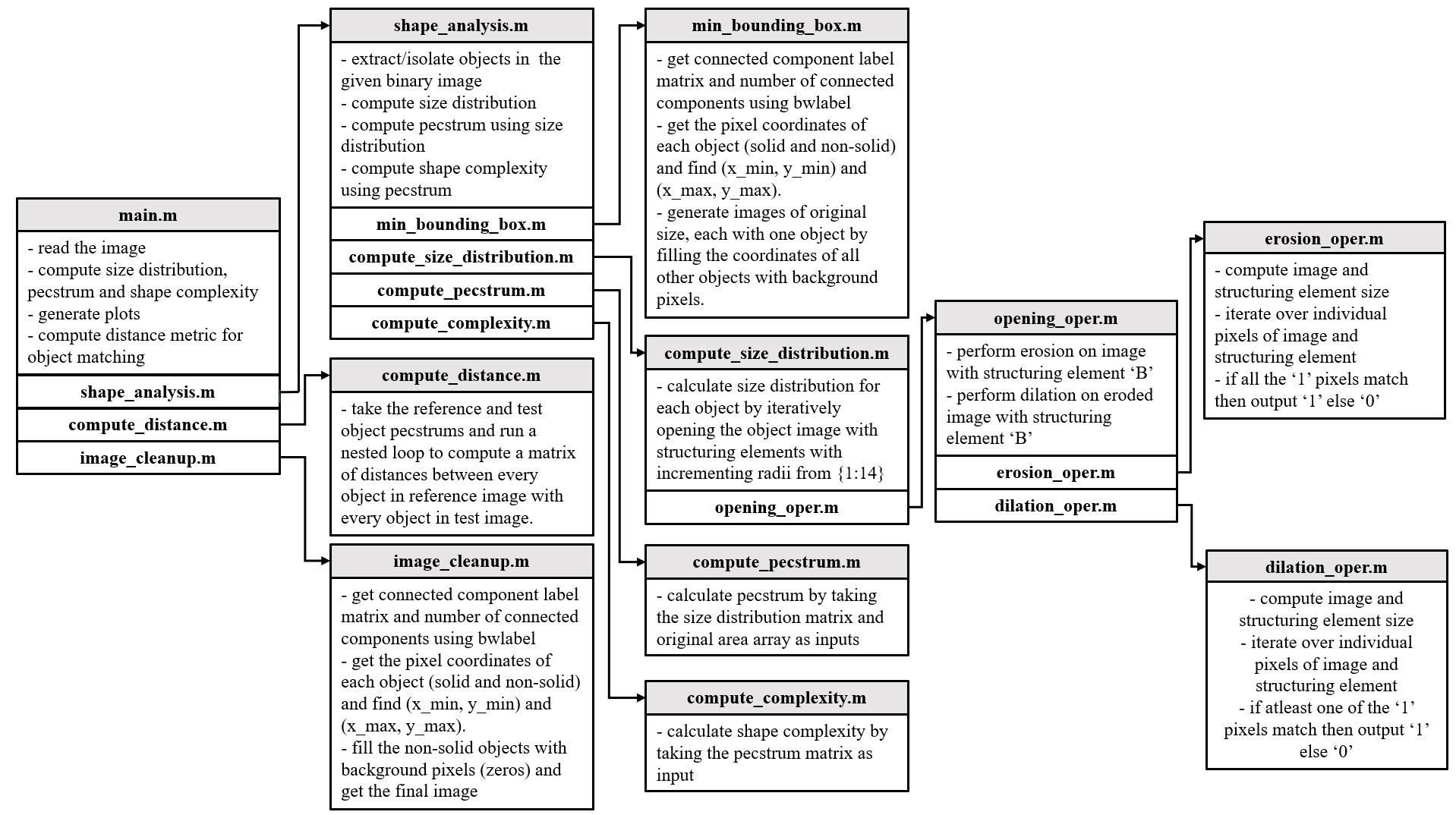
* **main.m** file gives the skeleton of the input image, superimposed skeleton on the original image and its partial reconstruction using three different structuring elements namely, , square, rhombus and VEC045. The functioning is as listed below:

1. It calls **skeletonization.m** function considering image and structuring elements as its input. This function gives us the skeletal components of the image and their union as the output by calling **opening\_oper.m** function.
2. The **opening\_oper.m** function takes image & structuring elements as its inputs and performs opening operation as defined in the previous section. It gives opened image as its output which is returned to **skeletonization.m** function. While in execution this function further calls **erosion\_oper.m** and **dilation\_oper.m** functions.
   1. The **erosion\_oper.m** computes erosion operation on a given image ‘X’ with respect to a given structuring element ‘B’. It performs erosion by moving the structuring element all over the ‘X’ comparing each individual pixel. It checks when B’s pixel is 1 but X’s pixel is 0, implying ‘B’ is not completely inside of X and hence the point to which B has been translated to will be colored black (0) else it will colored white (1).
   2. The **dilation.m** computes dilation in the same way as erosion but checks the condition where at one white pixel from matches with white pixel from X, in that case the point to which X is translated to is colored white (1) else it is black (0).
   3. The **compute\_union.m** function takes two images as input and computes their union by making all white pixels (1) stay from both the images.
3. After the skeletonization.m function is executed we get two outputs within main.m one of them is a cell S\_X containing skeletal components of the input image and the other is their union. Using the first output we perform reconstruction operation. We give the cell S\_X as the input to reconstruction.m function along with a variable which determines the extent of reconstruction or partial reconstruction.
4. The **reconstruction.m** function computes a complete or partial reconstruction of the image after taking its skeletal components as an input. It further calls **dilation\_oper.m** and **compute\_union.m** which returns their outputs to **reconstruction.m**.

**PART 2: Shape Analysis**

The files corresponding to Shape Analysis (PART 2) can be found in the folder **EE555\_Project\_2/Part\_2/**. The main code file for the PART 1 is **main.m**.

In Figure 3, we can observe the flowchart followed in the MATLAB implementation for PART 2: Shape Analysis.



**Figure 3**: Flowchart depicting the code implementation of PART 2: Shape Analysis (**main.m**).

**NOTE**: In order to obtain the results as shown in the next section kindly execute/run the **main.m** code file located in **EE555\_Project\_2/Part\_2/**.

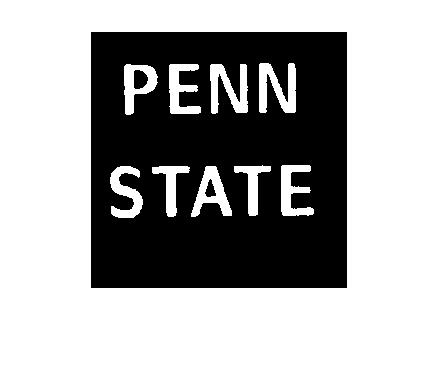
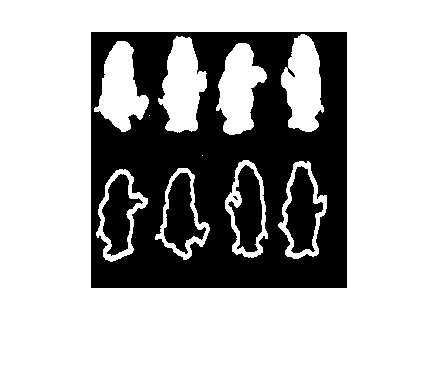
* **main.m** file takes the image file name (*‘match1.gif’*) as input. This input file name is passed into the function called **shape\_analysis.m** which takes as input the file name, a flag value 0 and an image array obtained after correction (only applies to images *‘shadow1.gif’ and ‘shadow1rotated.gif’* and 0 otherwise). The output is the size distribution matrix, the pecstrum matrix and the shape complexity array. Plots of these distributions are generated. The file is also used for object matching between *‘match1.gif’* (reference image) and *‘match3.gif’* (test image). The pecstrum distributions obtained in both the images are passed into the function **compute\_distance.m** which takes as inputs the two pecstrums and the weight array. The output is a distance matrix. For matching images *‘shadow1.gif’* and *‘shadow1rotated.gif’*, the image file names are first passed into the function **image\_cleanup.m** which gives an output the corresponding image arrays after removing non-solid objects. These images are then passed as input to **shape\_analysis.m** with flag value 1 to obtain the pecstrums. The pecstrums are further passed into **compute\_distance.m** with a chosen weight array to obtain the distance matrix. The functioning is as listed below:

1. It calls **shape\_analysis.m** which generates the size distribution matrix, the pecstrum matrix and the shape complexity array. The binary image is read from the file name. Images of original size, each with one object is obtained by calling the **min\_bounding\_box.m** function. This function further calls **compute\_size\_distribution.m**, **compute\_pecstrum.m** and **compute\_complexity.m**.
   1. The **min\_bounding\_box.m** calls bwlabel() function of MATLAB to get connected component label matrix and the number of connected components. The pixel coordinates of each object (solid and non-solid) are obtained using the find() function. The bounding box coordinates (x\_min, x\_max) and (y\_min, y\_max) are obtained to get a tight bound around each object. To generate an image with only one object, the coordinates of all other objects are filled with background pixels. The output is a matrix of binary images, each containing one object.
   2. The **compute\_size\_distribution.m** calculates size distribution for each object by iteratively opening the object image with structuring elements with incrementing radii from {1:14}. The area is calculated using the sum() function of MATLAB. The size of the structuring element for a radius is calculated using the formula . The function calls **opening\_oper.m**.
      1. The **opening\_oper.m** function takes image & structuring elements as its inputs and performs opening operation as defined in the previous section. It gives opened image as its output. While in execution this function further calls **erosion\_oper.m** and **dilation\_oper.m** functions.
         * The **erosion\_oper.m** computes erosion operation on a given image ‘X’ with respect to a given structuring element ‘B’. It performs erosion by moving the structuring element all over the ‘X’ comparing each individual pixel. It checks when B’s pixel is 1 but X’s pixel is 0, implying ‘B’ is not completely inside of X and hence the point to which B has been translated to will be colored black (0) else it will be colored white (1).
         * The **dilation.m** computes dilation in the same way as erosion but checks the condition where at one white pixel from matches with white pixel from X, in that case the point to which X is translated to is colored white (1) else it is black (0).
   3. The **compute\_pecstrum.m** calculates the pecstrum distribution using the formula provided above. It takes as input the size distribution matrix and the original area array.
   4. The **compute\_complexity.m** calculates the shape complexity by taking the pecstrum matrix as input and uses the entropy formula provided above.
2. It calls **compute\_distance.m** which takes the reference and test object pecstrum matrixes as input. The function runs a nested loop to compute a matrix of distances between every object in the reference image with every object in the test image using the distance metric formula mentioned above.
3. It calls **image\_cleanup.m** which takes the image file name as input. It calls bwlabel() function of MATLAB to get connected component label matrix and the number of connected components. The pixel coordinates of each object (solid and non-solid) are obtained using the find() function. The bounding box coordinates (x\_min, x\_max) and (y\_min, y\_max) are obtained to get a tight bound around each object. The non-solid object bounding boxes are found by observation. All the non-solid object coordinates are filled with background pixels (0’s) to get the final binary image array that contains only solid objects.
4. **Results**

In this section we discuss about the results corresponding both parts of the project.

**PART 1: Skeletonization and Reconstruction**

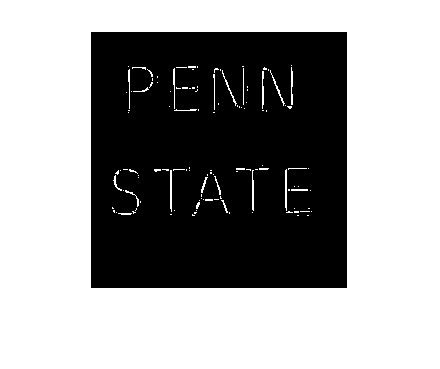
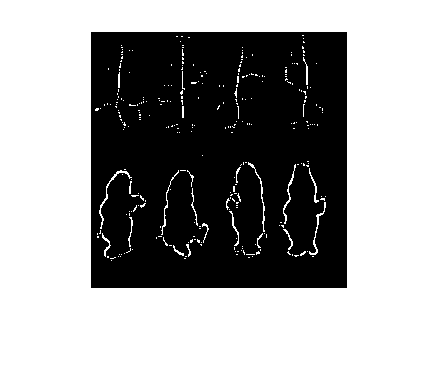
In this project we are given two different images i.e., ‘penn256.gif’ and ‘bear.gif’ as depicted in Figure 4 and 5, respectively. There are two different tasks as part of the project’s PART 1 i.e., 1) to perform the homotopic skeletonization of the two given images and output the skeletonized image superimposed on the original input image, and 2) to perform partial reconstruction with ‘k’ = (2, 3 and 4) considering the skeletal components computed in (1).

**Figure 4**: Original ‘penn256’ and ‘bear’ image.

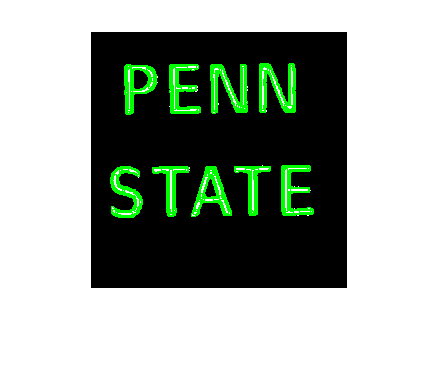
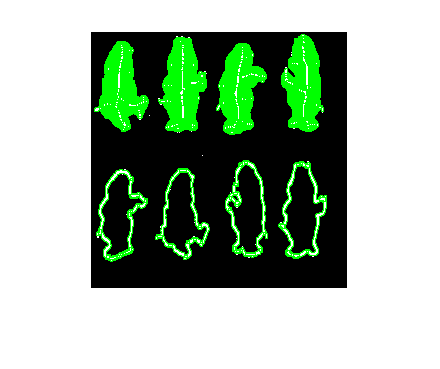
In this work, we have used three different structuring elements to perform the skeletonization and reconstruction operation as discussed in Section B. Methods i.e., 1) square, 2) rhombus, and 3) VEC045, as illustrated in Figure 1. We consider each of these structuring elements and explore the behavior of each of these in the case of skeletonization and reconstruction operation.

Firstly, we illustrate the functioning of square structuring element on the ‘penn256’ and the ‘bear’ image in the case of skeletonization operation. That is, we perform the skeletonization algorithm as illustrated in Section B. Methods with the break statement being erosion operation resulting in a null set. The result for both the images considering the square is illustrated in Figure 5, where it can be observed that the skeletons generate for both the images are comparable to the one illustrated in Lecture 6 (Page 12). However, the skeletons are not very smooth and are broken at places illustrating the deficiency of the selected structuring element.

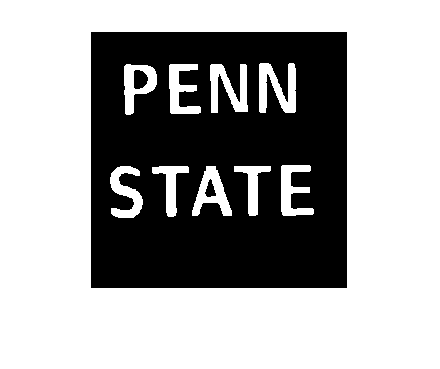
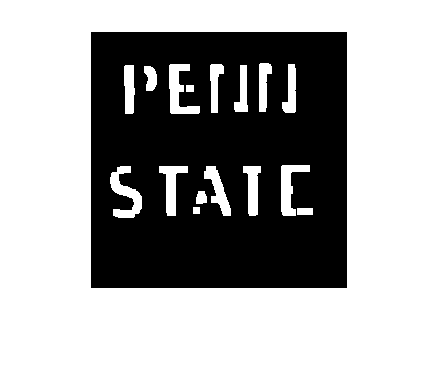
**Figure 5**: Skeleton of ‘penn256’ and ‘bear’ obtained using square structuring element.

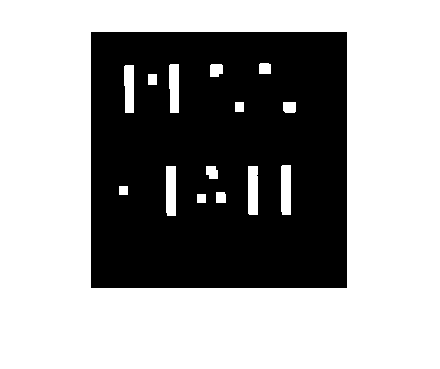
Further we superimpose the skeleton obtained by the skeletonization algorithm on to the original image using the ‘**imfuse.m**’ MATLAB function and the superimposed images for both ‘penn256’ and ‘bear’ image can be observed in Figure 6.

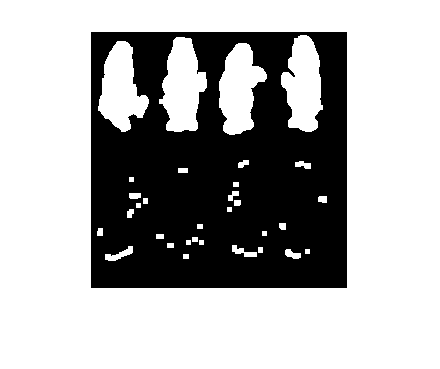
**Figure 6**: Super-imposed skeleton of ‘penn256’ and ‘bear’ obtained using square structuring element on the original image.

Further we also perform partial reconstruction using the skeletal components generated using the skeletonization algorithm illustrated in Section B. Methods and perform partial reconstruction considering different values of ‘k’ in ‘XkB’ i.e., k = (2, 3 & 4) for both ‘penn256’ and ‘bear’ image. The partial reconstruction for both ‘penn256’ and ‘bear’ image is illustrated in Figure 7 & 8, and from the figure it can be observed that as the value of ‘k’ increases, the white pixels with smaller area starts diminishing as can be observed for both the images. However, in the ‘bear’ image the complete white image starts to grow bigger, showing that the partial reconstruction has correlation to the opening operation.



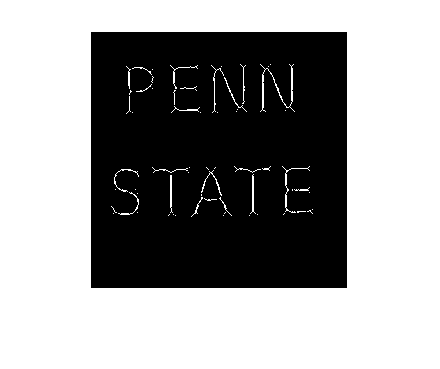
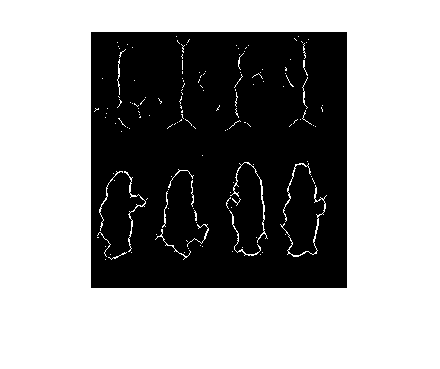
**Figure 7**: Partially reconstructed ‘**penn256**’ images with ‘k’ value in ‘XkB’ being 2, 3 & 4 considering square structuring element.



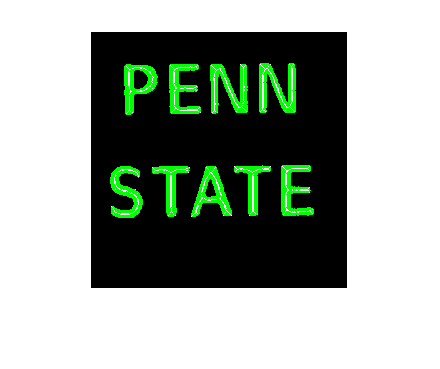
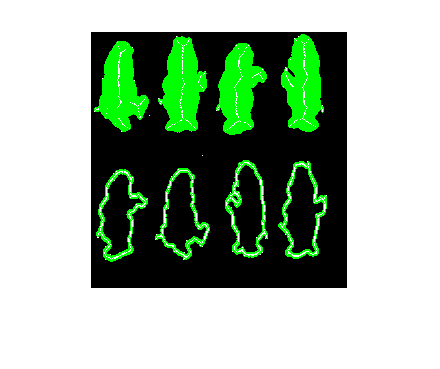
**Figure 8**: Partially reconstructed ‘**bear**’ images with ‘k’ value in ‘XkB’ being 2, 3 & 4 considering square structuring element.

Next, we consider the structuring element ‘rhombus’ on the ‘penn256’ and the ‘bear’ image in the case of skeletonization operation. That is, we perform the skeletonization algorithm as illustrated in Section B. Methods with the break statement being erosion operation resulting in a null set. The result for both the images considering the rhombus is illustrated in Figure 9, where it can be observed that the skeletons are relatively smoother than the skeleton generated by the square structuring element and it is closer to the centroid of the structures.

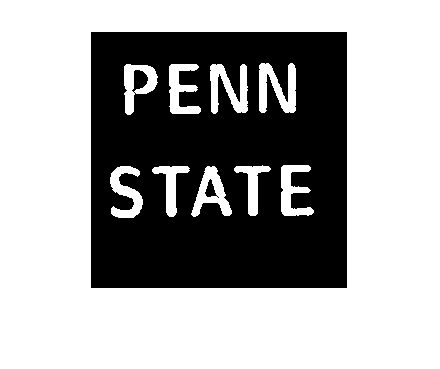
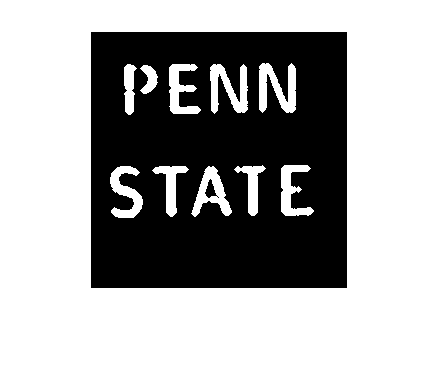
**Figure 9**: Skeleton of ‘penn256’ and ‘bear’ obtained using rhombus structuring element.

Further we superimpose the skeleton obtained by the skeletonization algorithm on to the original image using the ‘**imfuse.m**’ MATLAB function and the superimposed images for both ‘penn256’ and ‘bear’ image can be observed in Figure 6.

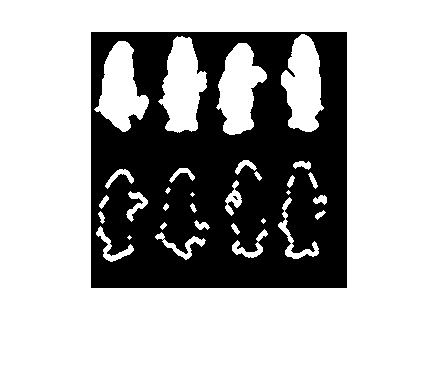
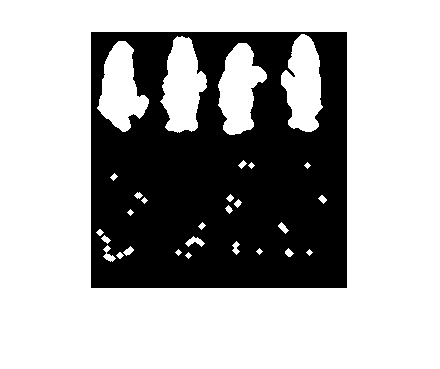
**Figure 10**: Super-imposed skeleton of ‘penn256’ and ‘bear’ obtained using rhombus square structuring element on the original image.

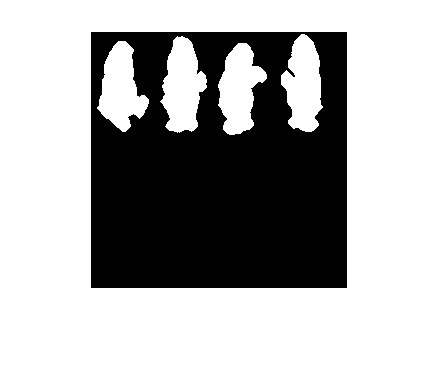
Further we also perform partial reconstruction using the skeletal components generated using the skeletonization algorithm illustrated in Section B. Methods and perform partial reconstruction considering different values of ‘k’ in ‘XkB’ i.e., k = (2, 3 & 4) for both ‘penn256’ and ‘bear’ image. The partial reconstruction for both the images are illustrated in Figure 11 & 12, and from the figure it can be observed that as the value of ‘k’ increases, the white pixels with smaller area starts diminishing as can be observed for both the images. If we compare the partial reconstruction results obtained with square and rhombus structuring element, it can be noticed that in the latter one, the white pixels diminish quickly with growing value of ‘k’ as compared to the rhombus.



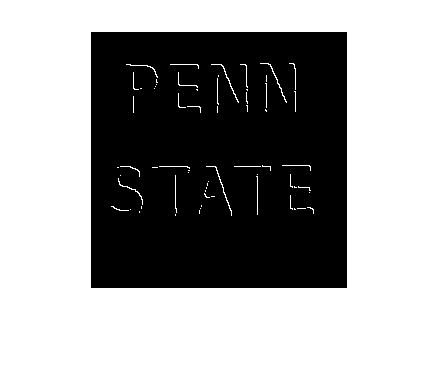
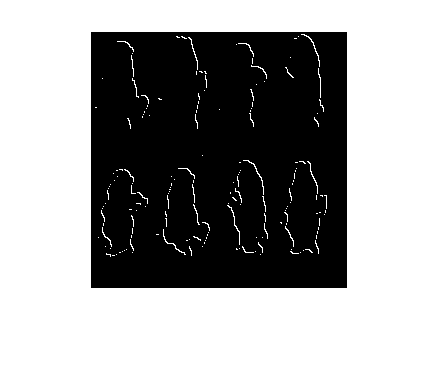
**Figure 11**: Partially reconstructed ‘**penn256**’ images with ‘k’ value in ‘XkB’ being 2, 3 & 4 considering rhombus structuring element.



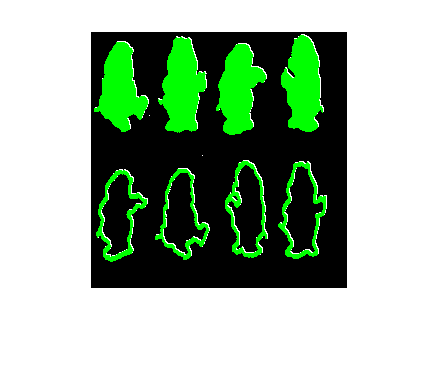
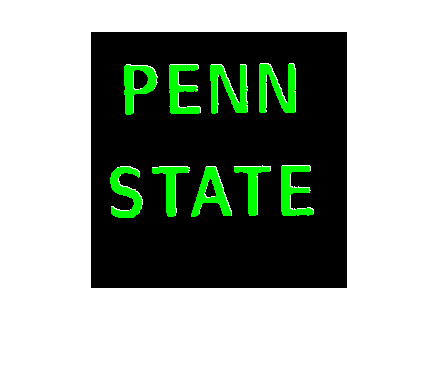
**Figure 12**: Partially reconstructed ‘**bear**’ images with ‘k’ value in ‘XkB’ being 2, 3 & 4 considering rhombus structuring element.

Lastly, we consider the structuring element ‘VEC045’ on the ‘penn256’ and the ‘bear’ image in the case of skeletonization operation and the result for both the images considering the ‘VEC045’ is illustrated in Figure 13, where it can be observed that the skeletons are appear at the edges of the body of the image structure, for example in the ‘penn256’ image the skeleton appears at the white pixels boundary and the same pattern can be observed in the case of ‘bear’ image as well. This skeleton representation is similar to the one observed in Lecture 6 (Page 12).

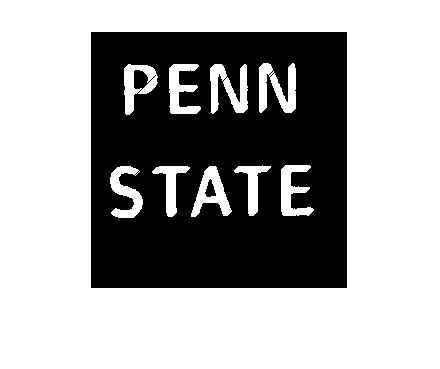
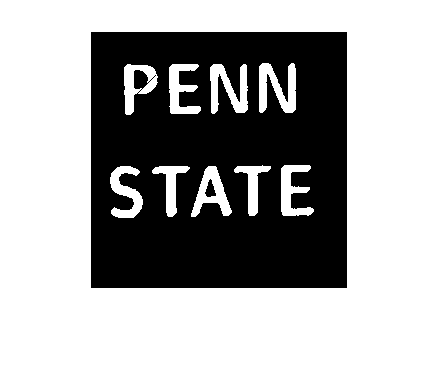
**Figure 13**: Skeleton of ‘penn256’ and ‘bear’ obtained using VEC045 structuring element.

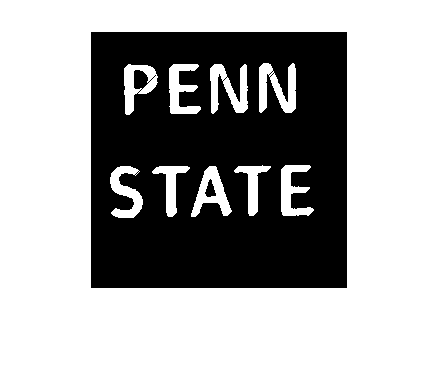
Further we superimpose the skeleton obtained by the skeletonization algorithm on to the original image using the ‘**imfuse.m**’ MATLAB function and the superimposed images for both ‘penn256’ and ‘bear’ image can be observed in Figure 6.



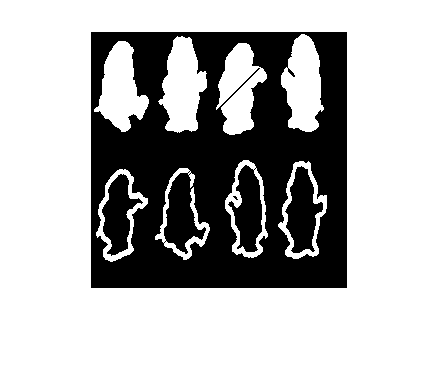
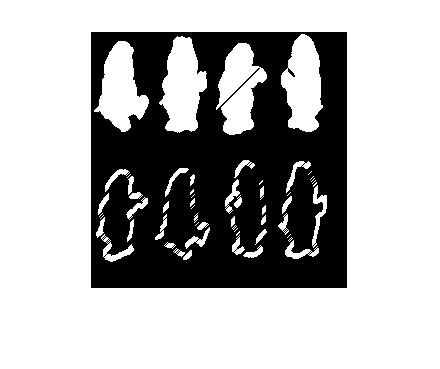
**Figure 14**: Super-imposed skeleton of ‘penn256’ and ‘bear’ obtained using VEC045 square structuring element on the original image.

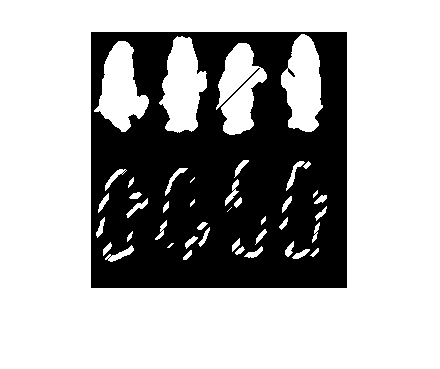
Further we also perform partial reconstruction using the skeletal components generated using the skeletonization algorithm illustrated in Section B. Methods and perform partial reconstruction considering different values of ‘k’ in ‘XkB’ i.e., k = (2, 3 & 4) for both ‘penn256’ and ‘bear’ image. The partial reconstruction for both the images are illustrated in Figure 15 & 16, and from the figure it can be observed that as the value of ‘k’ increases, the images appear to be completely reconstructed without any loss of white pixels as seen in the other two structuring elements. This could be because of the area covered by the structuring element i.e.; it has got lower number of pixels than the square and rhombus structuring element.



**Figure 15**: Partially reconstructed ‘**penn256**’ images with ‘k’ value in ‘XkB’ being 2, 3 & 4 considering VEC045 structuring element.



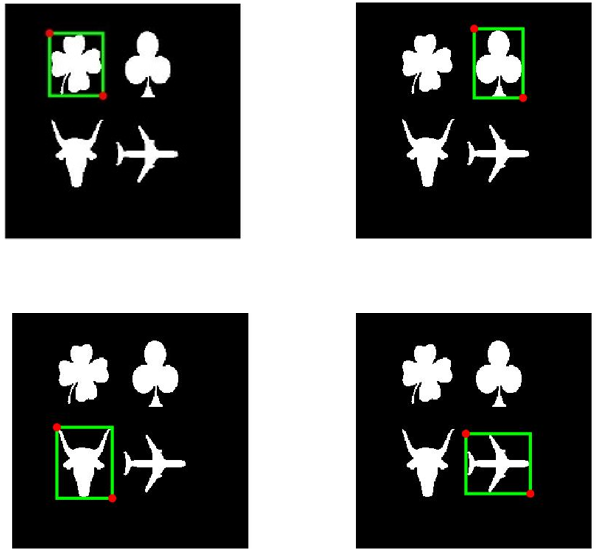
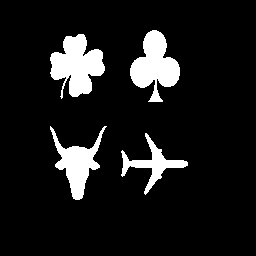
**Figure 16**: Partially reconstructed ‘**bear**’ images with ‘k’ value in ‘XkB’ being 2, 3 & 4 considering VEC045 structuring element.

**PART 2: Shape Analysis**

There are two parts in Part 2 of this project. In Part 2(a), we are required to (i) compute the size distribution U(n), pecstrum f(n) and complexity H(X | B) of each object in the image *match1.gif*. (ii) use Pecstral Analysis to match objects between the reference image *match1.gif* and the test image *match3.gif*. In Part 2(b), there are two images given to us: *shadow1.gif* and *shadow1rotated.gif* between which object matching has to be achieved. The only difference between the object matching performed in Part 2(a) (ii) is that these images also have non-solid objects. So, we use minimum bounding rectangle method to remove the non-solid objects and perform object matching similar to that of Part 2(a).

**PART 2(a)(i)**

We are given a binary image match1.gif and the objective is to find the size distribution, pecstrum and shape complexity of all the four objects present in it: (1) Clover (2) Steer (3) Airplane (4) Spade



**Figure 17**: Original Image *match1.gif* (left) and Images of *match1.gif* with MBR around each object (right).

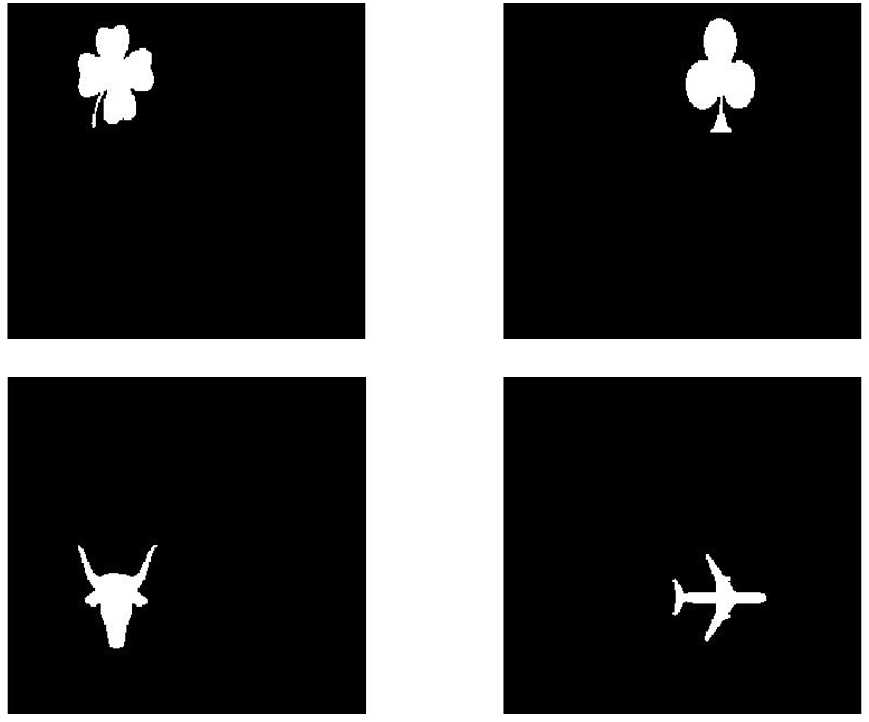
All the four objects were extracted by generating minimum bounding rectangles around each object. This way, the pixel coordinates of each object was obtained as shown in Figure 17. We formed four images of the original size, each with one object. To extract an object, we filled all the other objects with background pixels. The resulting images used for shape analysis is shown in Figure 18.

Airplane object

Steer object

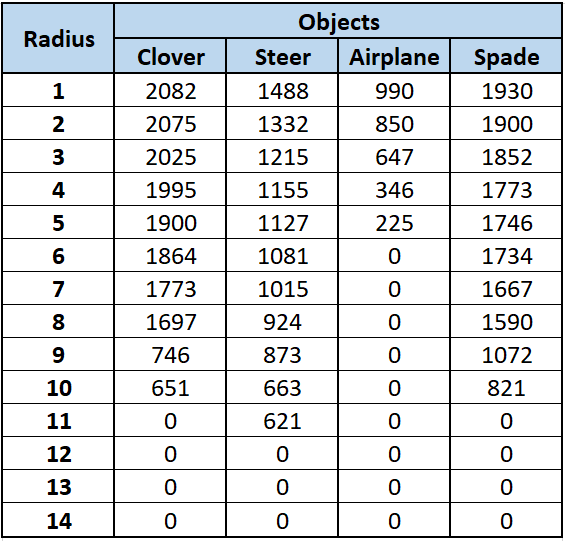
Spade object

Clover object



**Figure 18**: Images of objects extracted from *match1.gif.*

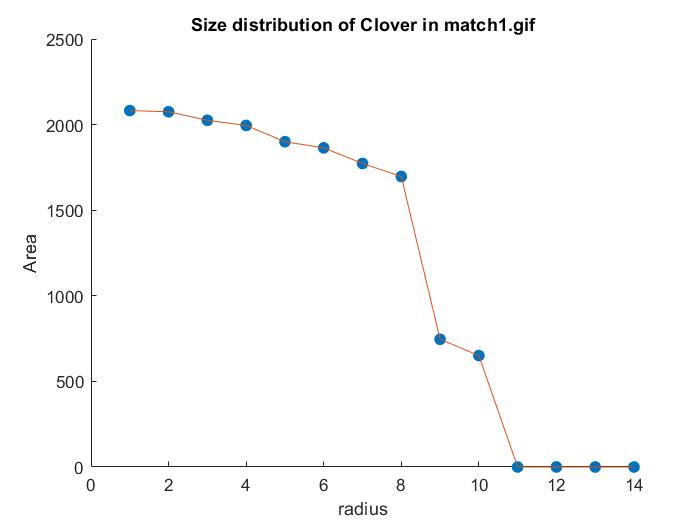
After isolating the objects, we iteratively perform opening operation on each object image using structuring elements of size 3, 5, 7, …, (2r +1). We chose r = {1:14} to observe the trend of the size distributions of each object as they monotonically decrease to zero. The area of the opened image was calculated in each iteration by summing the foreground pixels in the it. We obtained a size distribution matrix of all objects of dimensions [4 x 14]. Table 1 gives the size distributions of each object.



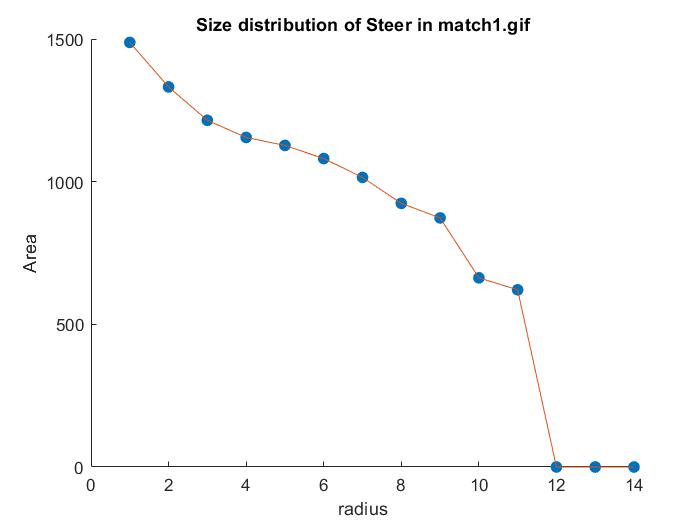
**Table 1**: Size Distribution of all four objects in *match1.gif* computed iteratively for incrementing radii of structuring elements.

It can be observed from Table 1 the area of the opened image decreases monotonically as radius of the structuring element used for opening the image increases. This experimentally proves the statement in P&V 6.11 that the size distribution U(r) is a strictly monotone function.

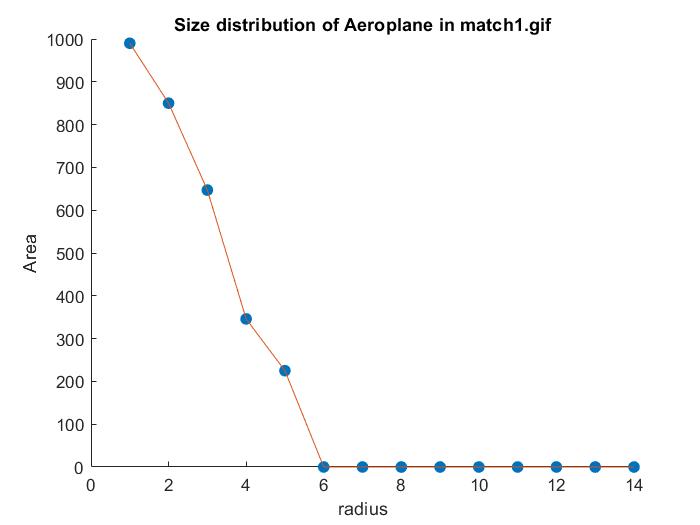
The plots of size distribution of all four objects are shown in Figures 19 to 22. The plots also clearly show that the area decreases with increase in radius of the structuring element.



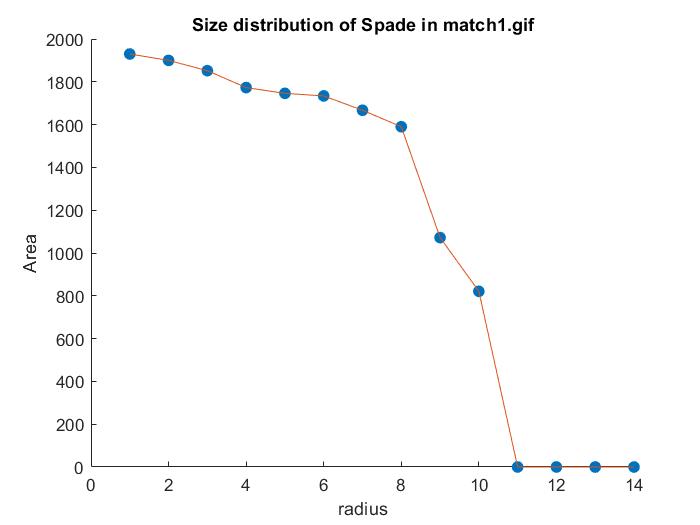
**Figure 19**: Size distribution of Clover in *match1.gif.*



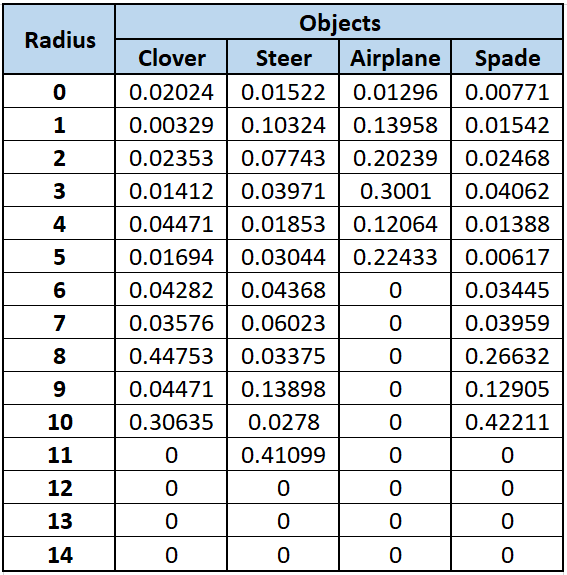
**Figure 20**: Size distribution of Steer in *match1.gif.*



**Figure 21**: Size distribution of Airplane in *match1.gif.*

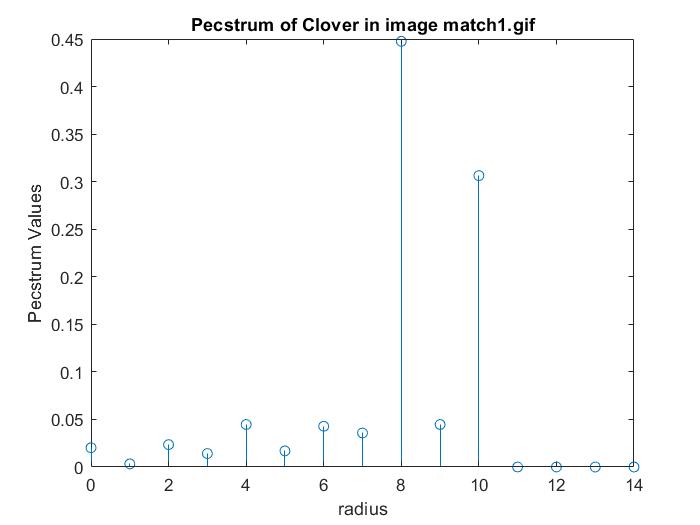
**Figure 22**: Size distribution of Spade in *match1.gif.*

In Figure 19, there is a steep drop in the area of after a radius of 8. This is due to the fact that the structuring element 8B cuts the leaves of the clover object, which contribute to a big portion of the area of the clover object. The trend in Figure 21 clearly depicts the shape features of the Airplane object. The Airplane object has sharp edges in the tail and wings, which makes its original area the least. After chopping these sharp edges with the increase in the radius, it can be observed that the area converges to zero in the least number of iterations (at 6B).

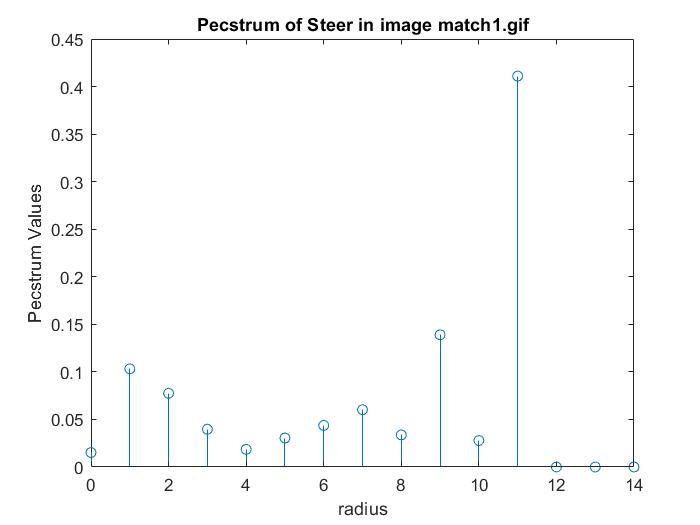


**Table 2**: Pecstrum values of all four objects in *match1.gif* computed iteratively for incrementing radii of structuring elements.

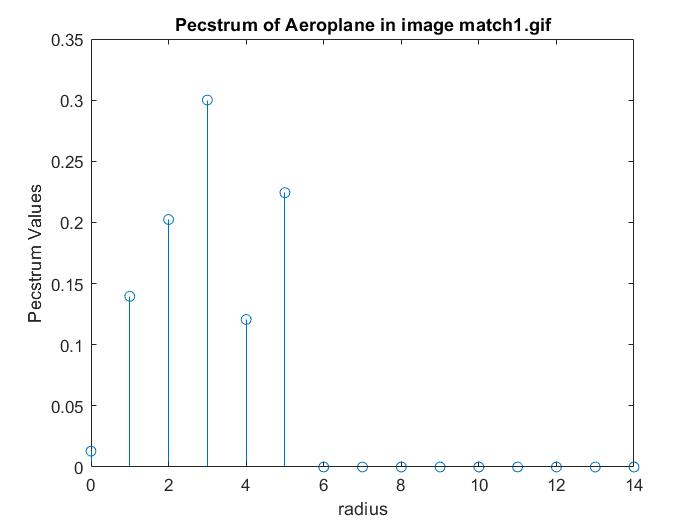
Table 2 provides pecstrum values of each object in *match1.gif* for different radii of the structuring element. The discrete plots of pecstrum values is shown in Figures 23 to 26.



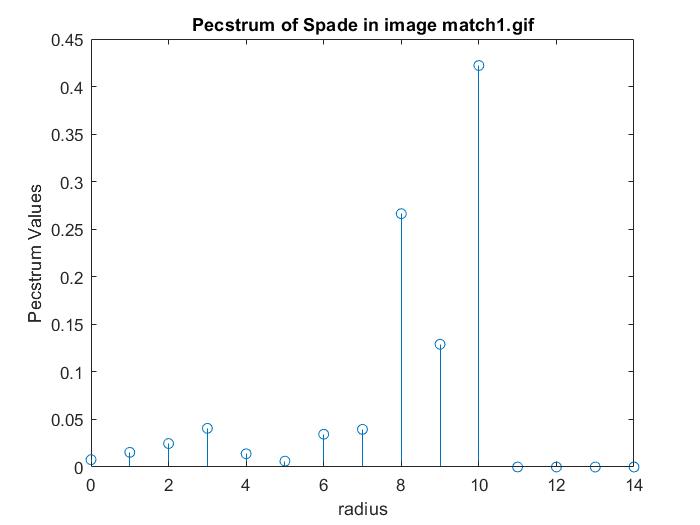
**Figure 23**: Pecstrum values plot of Clover in *match1.gif.*



**Figure 24**: Pecstrum values plot of Steer in *match1.gif.*

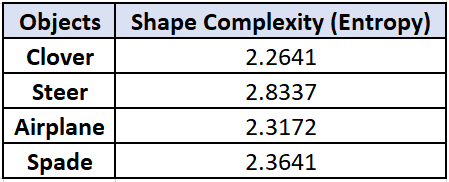


**Figure 25**: Pecstrum values plot of Airplane in *match1.gif.*



**Figure 26**: Pecstrum values plot of Spade in *match1.gif.*

A steep drop in the values of size distribution gets reflected as a spike in the values of pecstrum. These spikes can be observed in clover for a radius of 8 in Figure 23, steer for a radius of 11 in Figure 24 and spade for a radius of 10 in Figure 26. The drop in the size distribution due the chopped wings and tails in the Airplane object is reflected as the biggest spike for a radius of 3 in Figure 25.

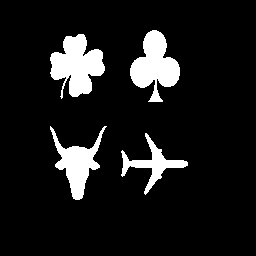
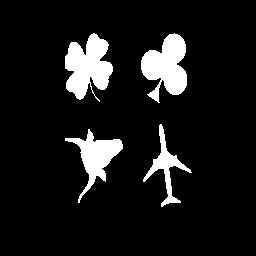


**Table 3**: Shape complexity (entropy) values of all four objects in *match1.gif.*

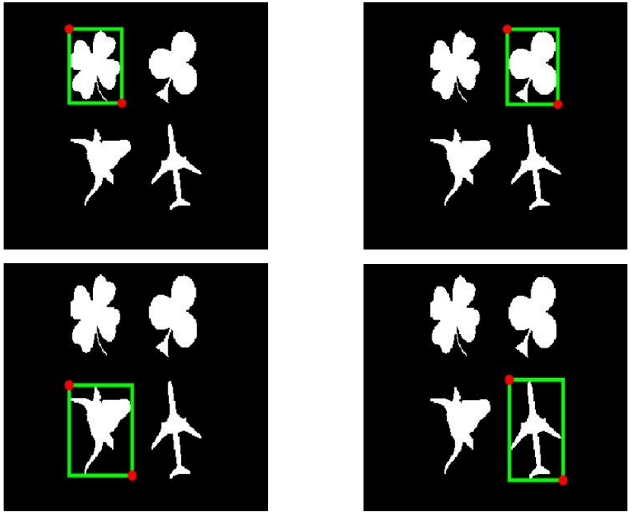
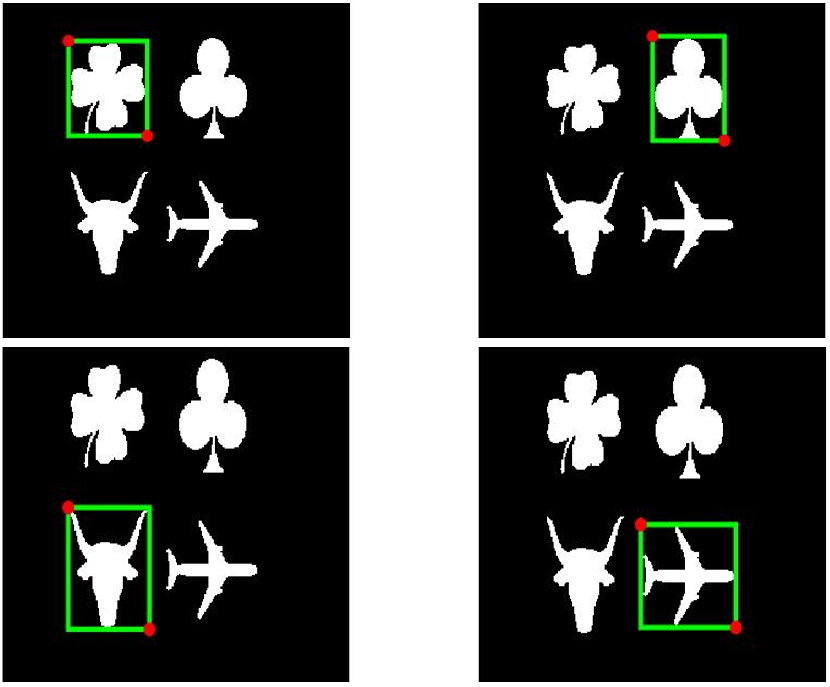
Table 3 shows the shape complexity of all four objects in *match1.gif*. Higher the values of entropy, more the complexity of the shape of the object. Shape complexity can be perceived as the boundary roughness averaged over all depths the structuring element can reach. It can be observed that Steer has the highest value of shape complexity, making it the most complex object. The least complex object is Clover.

**PART 2(a)(ii)**

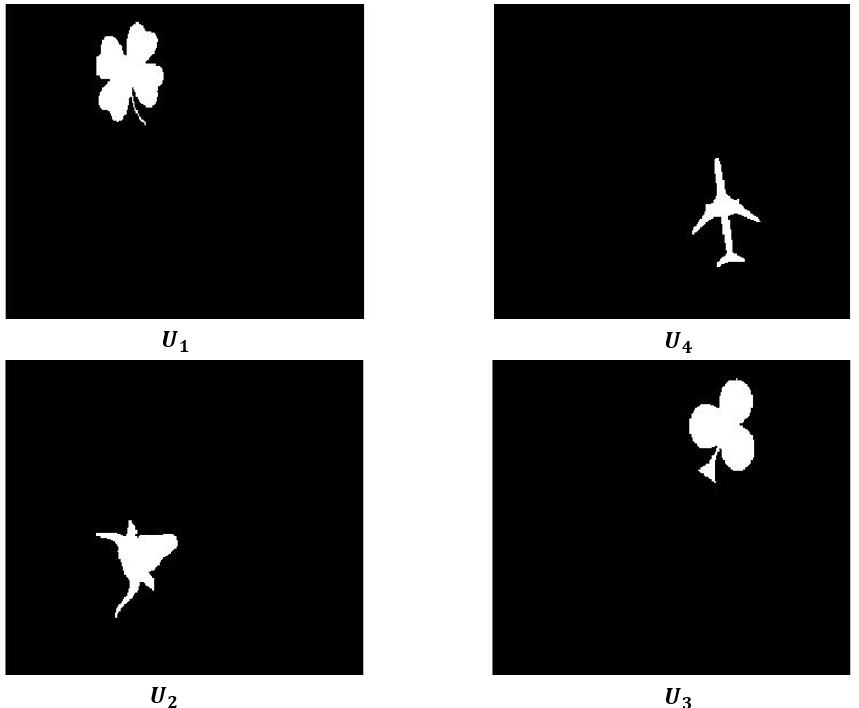
In this part, we are asked to use Pecstral analysis to determine which object in *match3.gif* best matches each object in *match1.gif*. Figure 27 shows the original images. It can be observed that the objects in *match3.gif* are all located in the same position as that of objects in *match1.gif* but are rotated versions of objects in *match1.gif*.



**Figure 27**: Original *match1.gif* (left) and *match3.gif* (right)



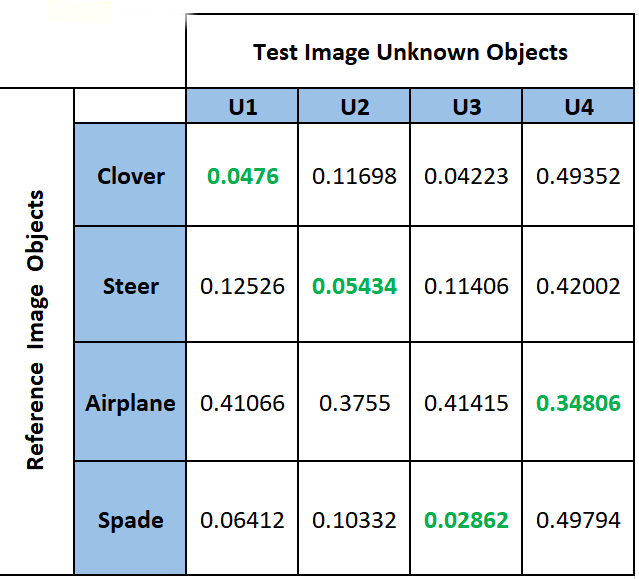
**Figure 28**: Images of *match1.gif*: reference image (left) and *match3.gif*: test image (right) with MBR around each object.



**Figure 29**: Extracted unknown objects of *match3.gif*: test image named and

The pecstrum values of objects in *match.gif*, also known as the reference image, were computed in the previous part. Figure 28 shows the MBR around each object in both reference and test images. In this part we first compute the pecstrum values of objects in *match3.gif*. It can be observed that the test objects are rotated versions of the reference objects. But we need to prove this quantitatively. Hence, we name the four objects Unknown and match the objects using the distance metric discussed in methods. The unknown objects are numbered in the same order they were identified by the MATLAB function bwlabel. The objects are named and The extracted objects are shown in Figure 29.

Table 4 shows a 4x4 matrix where the rows represent the objects of the reference image Clover, Steer, Airplane and Spade and the columns represent the unknown test image objects and



**Table 4**: Distance calculations between objects in *match1.gif* (reference image) and *match3.gif* (test image)

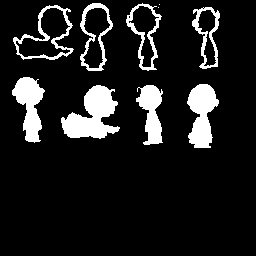
The distances highlighted in green are the minimum distances between the objects in reference image and objects in the test image. This represents the best match between the objects in the two images. The weight array ( chosen for arriving at these results is . In the methods section, we discussed that the weight array is to be chosen such that the weights decrease proportionally with respect to the decreasing radius of the structuring element. We have followed the same approach. Observing the trend that the size distributions of all the objects tend to decrease rapidly approximately after a radius of 6 , we chose the weight array to be all ones before this radius and all zeros after. Our results are highly sensitive to the change in weight array.

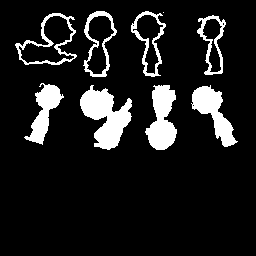
According to the results obtained, we observe that:

* **Clover** is matched with
* **Steer** is matched with
* **Airplane** is matched with
* **Spade** is matched with

Upon observation of the objects in the reference and test images in Figure 27, we notice that is the rotated version of **Clover**, is the rotated version of **Steer,** is the rotated version of **Spade** andis the rotated version of **Airplane**. This concludes that our Pecstral analysis for object matching is correct.

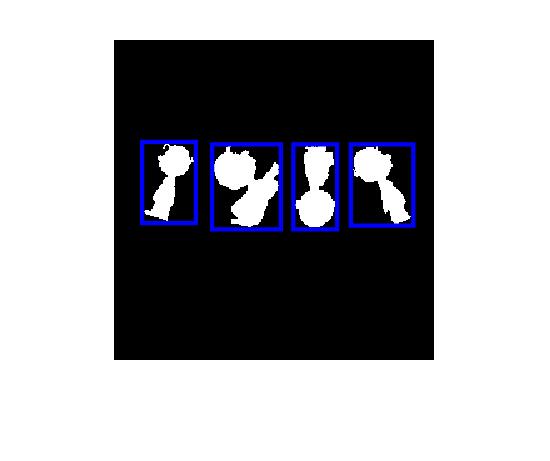
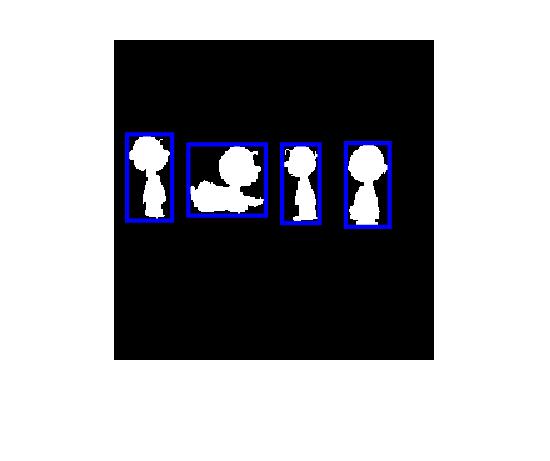
**PART 2(b)**

In this part, we are given two images *shadow1.gif* and *shadow1rotated.gif.* The images are shown in Figure 30. We have to match the solid objects in both the images.



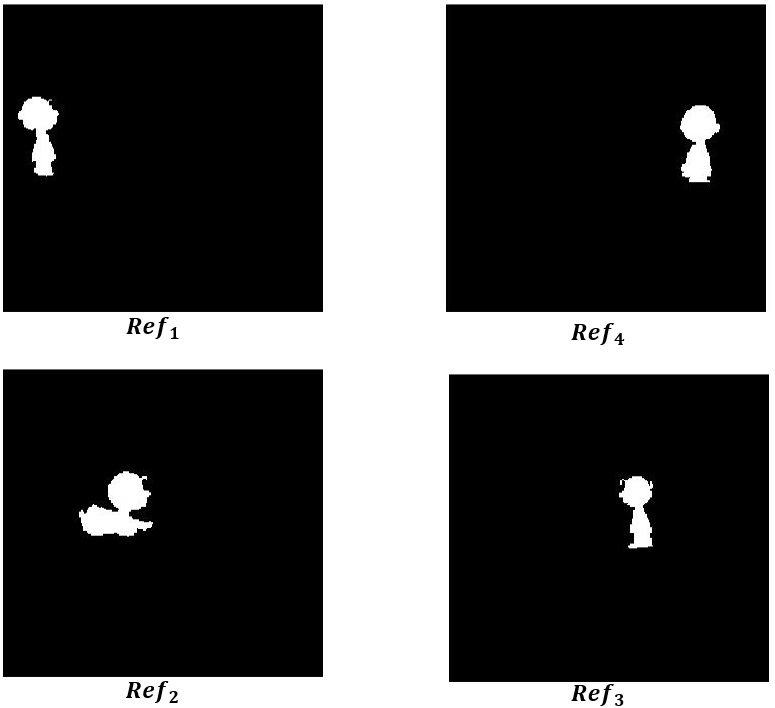
**Figure 30**: Original *shadow1.gif* (left) and *shadow1rotated.gif* (right)

This task cannot be achieved by directly applying the method used in Part 2(a)(ii). This is because, the images contain non-solid objects (hollow) that distort the results of pecstral analysis and hence object matching. Hence, we need an additional step of removing these non-solid objects from both the images. We achieved this by finding the pixel coordinates of the non-solid objects with the help of Minimum Bounding Rectangles by calling the bwlabel function of MATLAB on both the images. We observed which MBRs belong to the non-solid objects and filled the pixel coordinates of these objects with background pixels (zeros). After this step, we had corrected images of both *shadow1.gif* and *shadow1rotated.gif*. The steps used in Part 2(a)(ii) were used on these two corrected images to match the objects in them. The images after correction and finding MBRs of objects in them are shown in Figure 31. This task is more complicated than that of Part 2(a)(ii) because the objects in each image are differently located spatially.

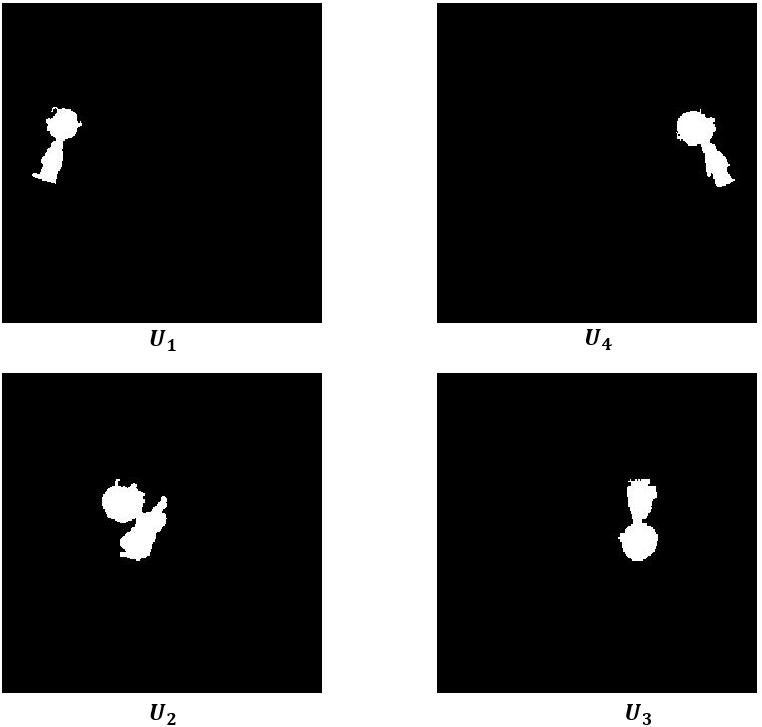


**Figure 31**: Corrected *shadow1.gif* (left) and *shadow1rotated.gif* (right) with MBR around each object.

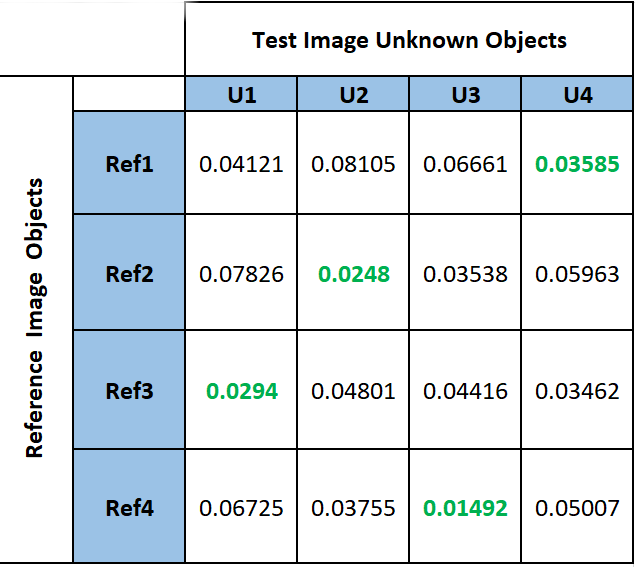
For this task, we consider the corrected *shadow1.gif* to be the reference image and the corrected *shadow1rotated.gif* to be the test image. We perform pecstral analysis similar to Part 2(a)(ii) to achieve object matching. Pecstrums of both the corrected images are calculated and the distance matrix is calculated using the equation discussed in Methods. Table 5 shows the calculated distance matrix. We chose the weights to be [1,1,1,0.8,0.7,0.1,0,0,0,0,0,0,0,0,0]. We observe that the distance calculations are very sensitive to change in weights.



**Figure 32**: Extracted objects of corrected version of *shadow1.gif.*



**Figure 33**: Extracted objects of corrected version of *shadow1rotated.gif.*



**Table 5**: Distance calculations between objects in corrected *shadow1.gif* (reference image) and corrected *shadow1rotated.gif* (test image)

In Table5, the minimum distances between objects in reference image and test image are highlighted in green. The minimum values imply best match between the corresponding objects. The matching is as follows:

* is matched to
* is matched to
* is matched to
* is matched to

Upon observation of the images in Figure 31, we note that is a rotated version of is a rotated version of is a rotated version of is a rotated version of This exactly matches the results of our algorithm, conclusively proving that our object matching algorithm and weights are correct.

Hence, the project is completed successfully.

**Conclusion**

In this project, we experimentally proved the theoretical concepts learnt in class. In Shape analysis, we implemented functions to calculate size distribution, pecstrum and shape complexity of multiple objects in a given binary image. This led to a better understanding and intuition of these concepts. We observed the plots of distributions and experimentally proved that the size distribution is inversely proportional to the radius of the structuring element. We noticed that the size distribution is strictly monotonic decreasing function with respect to the radius. We also quantitatively classified the complexity of objects in a given image with the help of entropy values. We also noted how important isolation/extraction of objects in an image are as a pre-processing step to make the shape analysis calculations easier and errorless. We also performed pecstral analysis for object matching, which is one of the most important concepts required for pattern recognition and computer vision. We could quantitatively match the objects in a reference image to objects in a test image using distance metric and pecstral analysis. We also noted how sensitive the weight array is to the results of object matching. With all these experiments, we now have a better understanding of the working of these algorithms and an intuition of their practical applications.