

Pooja Voladoddi
6571-9160-65
voladodd@usc.edu

EE569 – Homework #3 Report
November 3, 2013

PROBLEM 1 – SPATIAL WARPING

Abstract and Motivation

Warping is essentially a process to introduce distortion into a digital image by manipulation. This distortion could serve many purposes. At the fundamental level, it could be for creative purposes, and to cancel distortion from an already existing distorted images. At higher levels of implementation, warping can be used for morphing (which in itself has several important applications; for generating certain visual effects in cinema, analysis of facial expressions in video feeds, etc)

In short, there are many domains that make use of warping – may it be in security, film or fields that require constant correction of distorted images.

An image warp is defined by a mapping from the coordinate space of a source image (u,v) to the coordinate space of a destination image (x,y) .

We will look at three cases of Warping that require distorting the input image into 1) a diamond shaped image

2) a pentagon shaped image

3) a circle shaped image

1a- WARPING TO DIAMOND SHAPE

I. Procedure

There are two approaches taken to this problem – one is using a scale factor and one is using 1st order warping polynomial.

Because our output image does not contain any curves, it should be noted that higher order polynomials cannot be used.

■ Approach 1

- Divide input image into four quadrants
- Map each quadrant into a triangle by scaling the pixels in corresponding source image quadrant.
- The figure below illustrates it more clearly.

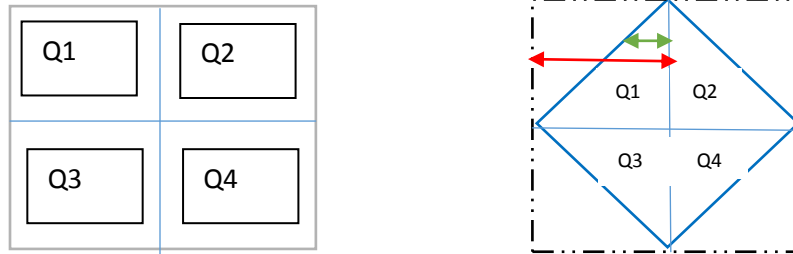


Figure 1. [L] Input Square Image(u,v) [R] Output Diamond warp image(x,y)

- The scaling factor for each quadrant is given by the ratio of length of each triangle quadrant to the length of square input image (green by red lines Figure 1. [R])
- In loops for four quadrants, the reverse address mapping is performed to get the required coordinates.

$$u = \text{scale_factor} * x;$$

$$v = \text{scale_factor} * y;$$

These addresses are then used to get the output image using bilinear interpolation (reference example is given below)

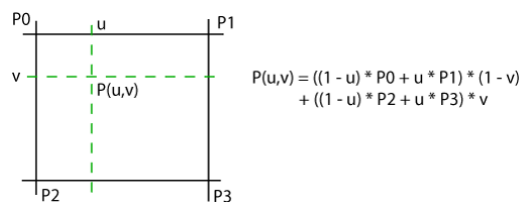
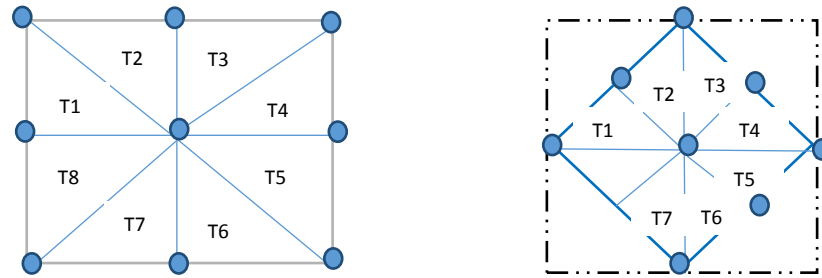


Figure 2. Bilinear Interpolation

■ Approach 2

- Divide the input source image into 8 triangles



- Select 3 control points for each triangle from square image [LEFT] and warp them into resultant triangles in warped image [RIGHT], as shown above. Control points are nothing but the coordinates of the images (blue dots as in above figure)

- This is done using 1st order polynomial equation of transformation

$$\begin{bmatrix} u_p \\ v_q \\ 1 \end{bmatrix} = \begin{bmatrix} a_0 & a_1 & a_2 \\ b_0 & b_1 & b_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_j \\ y_k \\ 1 \end{bmatrix}$$

Where 'u' and 'v' are input image coordinates and 'x' and 'y' are output image coordinates. Therefore, we now have $U=[u_1...u_3]$ and $V=[v_1...v_3]$ for each triangle, and the same is obtained for 8 such triangles created in the above figure.

The above equations represent the reverse address mapping equations. We solve for 'a' and 'b' using

$$[a]=[xy]^{-1}[u]$$

$$[b]=[xy]^{-1}[v]$$

, where $[a]$ is the matrix consisting of 'a' coefficients of all the 3 control points in one region (or triangle), and $[u],[v]$ are the $[u1..u3]^T$ and $[v1..v3]^T$ matrices for one triangular region respectively (source image address coordinates of control points) .

The values are then substituted in the 1st order equation shown above to get reverse mapped coordinates of input image.

Bilinear Interpolation is performed to get the destination pixel intensity value.

To recover the original image, we follow reverse warping by inverting the scale factor in Approach 1, and interchanging the $[xy]$ and $[u],[v]$ values in Approach 2

II. Results

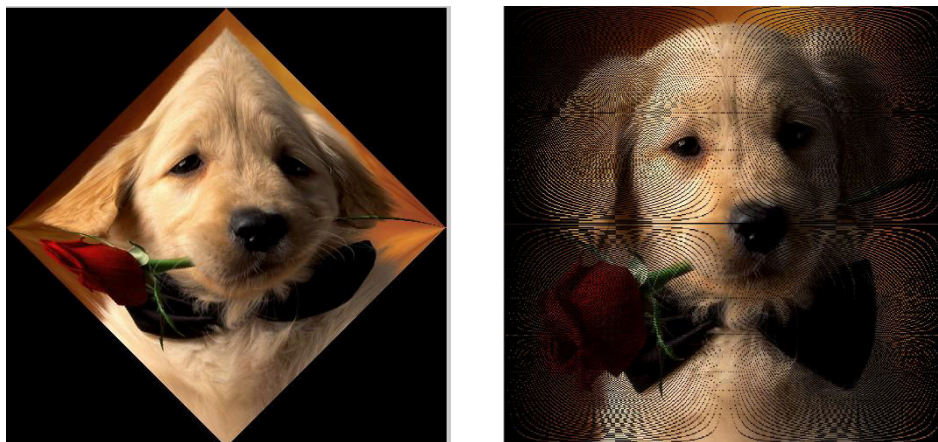


Figure 3 – Warped Image [L] , Recovered Image [R] using approach 1



Figure 4 – Diamond Warped Image [L] and Recovered Puppy image [R]



Figure 5 – Original puppy image

III. Discussion and Conclusion

It is observed from the figure 3 that scaling-approach recovered image has false contours. This is because some pixels while being mapped from source to destination are missed when scaling according to the method described. The second method has additional distortion along the diagonals as the region was mapped into 8 triangles , and the distortion is represented in the resulting image along the intersection edge of two consecutive triangular regions.

Note, however that the recovered image is absolutely close to the original image and there are no visible distortions present. This is an added advantage of the ‘control points’ method. Note that even though additional distortion is added while warping, the diamond warp is produced along with colour preservation, and is able to fully recover the image.

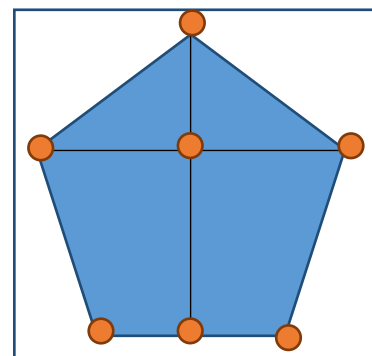
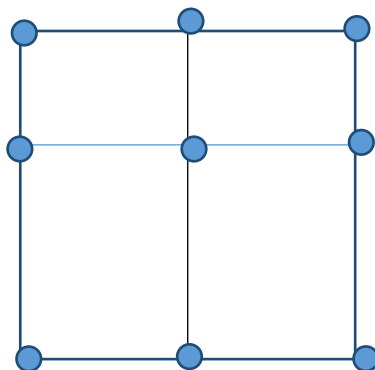
1(b) WARPING TO PENTAGON SHAPE

I. Procedure

The scaling factor is used in this problem too, as described in the problem 1a)

However, the pentagon to which the figure is scaled is assumed to be regular, and is scaled accordingly.

- The image is divided into 4 parts - two triangles and two quadrilaterals as shown in the figure below [RIGHT]



- In the above figure, the blue dots are the coordinates [Left] that are mapped into the orange ones [Right]. The orange dot coordinates are obtained by assuming that the top most vertex of the pentagon is situated at [4,246]. Therefore the further coordinates are obtained by using the equation

$$x = r \cos (\text{theta}) ; y = r \sin (\text{theta})$$

where $r = 246$, $\text{theta} = 72 \text{ deg.}$ (interior angle of pentagon)

- After the above step, we repeat the steps as in problem 1a.

In loops for four quadrants, the reverse address mapping is performed to get the required coordinates.

$$u = \text{scale_factor} * x;$$

$$v = \text{scale_factor} * y;$$

These addresses are then used to get the output image using bilinear interpolation (reference example is given below)

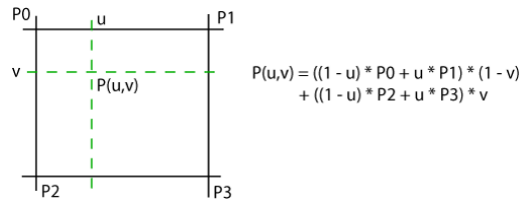


Figure 6. Bilinear Interpolation

Original image is again, recovered using the reverse procedure – where the scaling factors are interchanged for the loops, and the initialization of the loop variables is the reverse of the corresponding warping procedure.

I. Results



Figure 7. Pentagon warped Image



Figure 8. Original Image [left], recovered image [right]

II. Discussion and Conclusion

A nice, clean pentagon warp is obtained by using the ‘regular pentagon’ assumption, and it seems to have produced a fully recovered original image on reversal as well.

However, note that on close observation of [right] image of figure 8, the pixels across the edges and some on the boundary are lost, and there seems to be kind of an averaging blur on the recovered image. This is due to the fact that we are not retaining the entire size of the image while warping to the pentagon shape. In fact, we are making an assumption that the radius or the side length is 246, and thus the scaling factor responsible for scaling affects the recovered in the sense that not all the pixels are recovered, and the reduced number of pixels are now occupying the area equal to the original image, and hence the recovered image has some distortions.

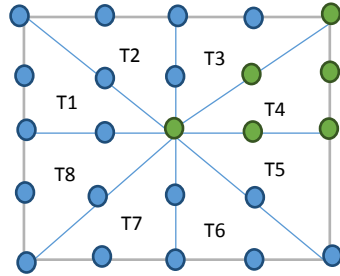
1(c) WARPING TO CIRCLE SHAPE

I. Procedure

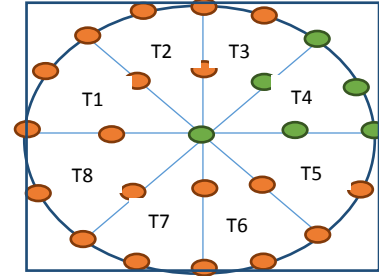
For transforming the square image into a circle , I make use of the second polynomial transformation matrix and polar coordinates. This is because the image produced in circle warp have non-linear curves , for which choosing three control points and 1st order polynomial will not suffice anymore.

Brief steps are explained as below :

- Divide the square image into four quadrants, and each quadrant into two triangles. Therefore, we totally have 8 regions T1...T8 as shown below.



Input Image (u,v)



Output Image (x,y)

- The blue, green and orange dots represent the control points of the input image and the output warped image respectively. The control points of the region T4 have been made green in both the images to show that each triangle has 6 control points. To solve for the coefficients of transformation matrix, we're hence going to be needing a higher order polynomial – a 2nd order polynomial will suffice.
- The radius is calculated using the formula

$$R1 = 250 * \text{sqrt}(2) - \text{sqrt}(74^2 + 74^2);$$

Where $250 * \text{sqrt}(2)$ is the diagonal length of the square (equivalent to circle size) and $\text{sqrt}(74^2 + 74^2)$ is the correction factor at corners.

- The coordinates of the dots represented are put in the second order transformation matrix that looks like below.

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} a_0 & a_1 & a_2 & a_3 & a_4 & a_5 \\ b_0 & b_1 & b_2 & b_3 & b_4 & b_5 \end{bmatrix} \begin{bmatrix} 1 \\ x \\ y \\ x^2 \\ xy \\ y^2 \end{bmatrix}$$

Which provides for reverse address mapping.

‘a’s and ‘b’s are determined by writing down $[u]=[u_1...u_6]^T$ for each triangle , similarly $[v]=[v_1...v_6]^T$. For all the control points, we get the unknown coefficients and frame a matrix like below

$$\mathbf{a}^T = [a_0, a_1, ..., a_5]$$

$$\mathbf{b}^T = [b_0, b_1, ..., b_5]$$

Post which we can represent the ‘A’ matrix for all the destination warped image coordinates like below.

$$\mathbf{A} = \begin{bmatrix} 1 & x_1 & y_1 & x_1^2 & x_1y_1 & y_1^2 \\ 1 & x_2 & y_2 & x_2^2 & x_2y_2 & y_2^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_M & y_M & x_M^2 & x_My_M & y_M^2 \end{bmatrix}$$

Using this, ‘a’s and ‘b’s are calculated using

$$\mathbf{a} = \mathbf{A}^{-1} \mathbf{u}$$

$$\mathbf{b} = \mathbf{A}^{-1} \mathbf{v}$$

In the given problem, there are six control points and 2nd order equation suffices this choice therefore instead of using the generalized inverse A^+ , we will use the actual inverse A^{-1} $x=r \cos(\theta)$ and $y=r \sin(\theta)$ where $\theta=\arctan(\text{slope})$ of each triangle [slope= 125/250].

Bilinear interpolation (as has already been shown in 1a, 1b) has been used to determine the pixel intensities using reverse address mapping.

Recovery of image is done by interchanging [u],[v] and [x],[y] coordinates.

I. Results



Figure 9 –Circular Warped Transformer image



Figure 10 – Original image [Left] and Recovered Image [Right]

II. Discussion and Conclusion

Figure 9 shows the circular warped image as implemented according to procedure described.

Figure 10 [right] is the recovered image . On comparing with the original image[Left] it is noticed that it is slightly blurred, due to rubber stretch effect (pixels are stretched inside a limited area while warping, and while reversing the procedure, this results in those restricted area pixels having to cover the original image area again, which results in lesser pixel density. Hence, the blurring effect.) It can also be noted that some of the pixels near the boundary are not recovered because of the fact that a circle boundary when reversed becomes a convex-like curve, and some of those pixels are left unrecovered from the warped image.

Problem 2: Perspective Transformation and Imaging Geometry

I. Abstract and Motivation

Consider a camera used to click pictures. Unlike the human visual system (which is capable of perceiving 3-D objects of the world), the camera has an image sensor plate onto which a 2-D image of the 3-D object it “sees” is projected.

In automation industry, where machines need to see things and perceive them, it is important to equip the machines with “vision” of the same capacity as human visual system, so that decisions can be taken based on the

images captured, similar to how the human brain perceives them, and take certain actions based on the decisions. Perspective Transformation is thus a precursor to machine vision or computer vision. Formally, it can be defined as a technique that is used to map a 3-D world object onto a 2-D camera/imaging device plane, and modification of the observed image to simulate an alternate viewpoint.

PROBLEM 2a – PRE-PROCESSING

II. Procedure

In this problem, we are going to map the length of a 2-unit length cube onto a camera/image plane that is 2-D. Therefore, our aim is to get the resultant coordinates and the intensity information of the images assumed to be placed on the cube.

For this, we follow the steps as shown below.

Step 1: Place the images on the faces of the cube. Bottom face is ignored

Step 2: Input the World Coordinates (cube coordinates- $[X,Y,Z]$), image height and width (pixel height/width) and the focal length of the camera considered.

Step 3: For the given camera placement, faces 1, 2 and 3 are visible. Check for visibility based on XYZ coordinates for face 1, face 2, face 3.

Step 4: Calculate the “x” and “y” image plane coordinates of the visible face using the formula

```
x=floor(100*X))+100;  
y=floor(100*Y))+100;  
z=floor(100*Z))+100; that maps X=-1,Y=-1,Z=-1 to x,y,z=1  
and X= 1, Y=1 , Z=1 to x,y,z=200
```

If only face 1 is visible ($Z=1$) , x,y are calculated.
 If only face 2 is visible ($X=1$) , z,y are calculated.
 If only face 3 is visible ($Y=1$) , x,z are calculated.

These are the **location** coordinates of the respective faces on the cube

The calculated **location coordinates** are then used to access the **intensity values** of the images placed on the cube faces, which are later displayed.

III. Figures

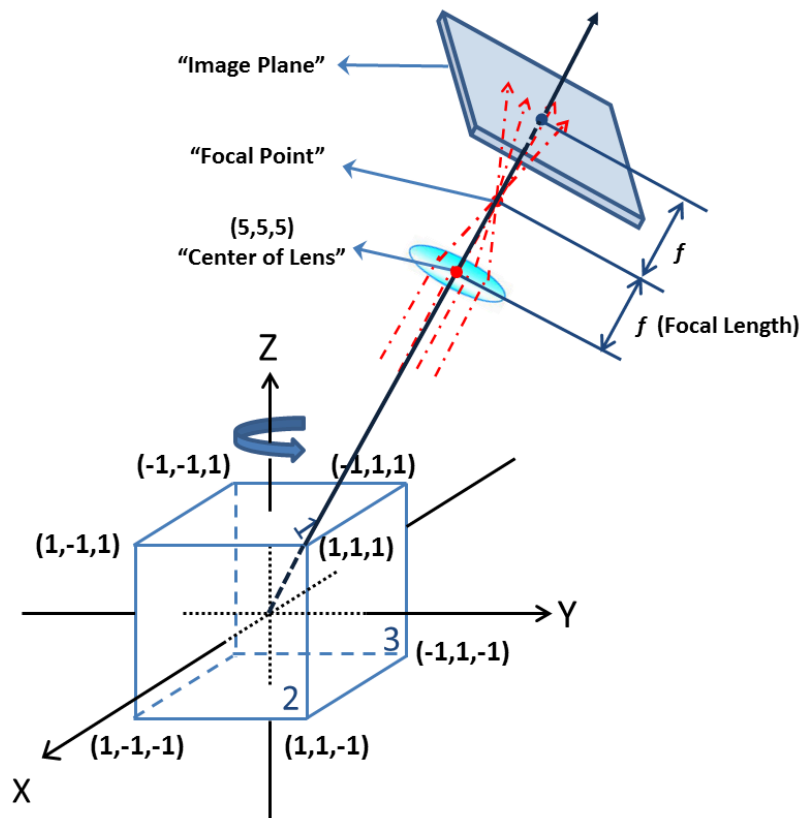


Figure 11 – Placement of camera and visible faces of cube

**FUNNY
EE 569**



Figure 12- [L-R] Images placed on face1, face2 and face3 respectively.

IV. Results , Discussion and Conclusion

The results can be observed on MATLAB command window , namely the location and the pixel intensities both. A function is written that uses all the steps in the procedure written above to return pixel intensities (RGB values) and the location (image plane coordinates).

The function is called from the Matlab command Window, and there exist three cases –

- 1) When only face 1, or face 2 or face 3 is visible – display the corresponding image plane coordinates and RGB values
- 2) Intersection of two faces (or three, in case of face1,face2,face3 corner (1,1,1)) is visible – display the images that are visible at intersection and pixel intensities of any one image.
- 3) When the pixel lies on faces 4,5,6 – or in the interior of the cube – pixel is deemed to be inaccessible and “not visible” is returned [This takes care of the many-to-one-mapping scenario, where only the nearest visible pixels are accessed. This is executed in step-3 of Procedure)
- 4) Some test results are published below for understanding this mechanism in a better way.

Input - cube coordinates/World Coordinates [X,Y,Z]	output Visible Face	output Intensity values [R,G,B]	output Image plane location (x,y)
[-0.2 , 0.2 , 1]	Face 1	[0 , 0 , 0]	(80, 120)
[1 , 0.2 , 1]	None (lies on face opposite to 3)	0	Not Visible
[1, 1 , 1]	Faces 1,2,3	[24, 12, 4] – face 3	(200,200)

The test results are verified using Adobe Photoshop CS6 software.

Further tests can be conducted to get the pixel values.

PROBLEM 2b – Capturing 3D scene

I. Procedure

Procedure for obtaining a 3-D scene remains straightforward and as explained in the HW problem sheet.

We take the input world coordinates [X Y Z] and produce corresponding ‘x’ , ‘y’ and ‘z’ values.

Do note that for a given 3D world coordinate, only two corresponding image plane coordinates exist at a time.

The process is summed up in the figure below -

$$w \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = K[R|t] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Where XYZ are world coordinates,

$$K = \text{intrinsic camera matrix} \quad \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

‘f’ – focal length , cx and cy are half-values of image height and width for translation of origin of image plane.

[R|t] = extrinsic camera matrix and is defined as below

$$[R|t] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} = \begin{bmatrix} X_c^X & X_c^Y & X_c^Z & -\mathbf{r} \cdot \mathbf{X}_c \\ Y_c^X & Y_c^Y & Y_c^Z & -\mathbf{r} \cdot \mathbf{Y}_c \\ Z_c^X & Z_c^Y & Z_c^Z & -\mathbf{r} \cdot \mathbf{Z}_c \end{bmatrix},$$

‘r’ is the alignment factor between camera origin and world origin $\mathbf{r}=(5,5,5)$

X_c , Y_c and Z_c are already defined and given as below.

$$\mathbf{Z}_c = \left(-\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}}\right)^T,$$

$$\mathbf{X}_c = \left(-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0\right)^T$$

$$\mathbf{Y}_c = \left(\frac{1}{\sqrt{6}}, \frac{1}{\sqrt{6}}, -\frac{2}{\sqrt{6}}\right)^T.$$

‘f’=sqrt(3).

So we plug in all these values, and calculate x,y ; y,z and x,z values in loop for faces 1,2,3 of the cube respectively. The cube is as displayed in Problem 2a. We then map the pixel values from the world coordinates to the image plane using “pixel density” for each face which gets captured as a 3-D object.

II. Results



III. Discussion and conclusion

The result shown above is the captured image for given details. Do note that the size of the captured image depends on 'pixel density'. Which is given by the ratio of cube side length divided by $\sqrt{\text{resolution of image}}$.

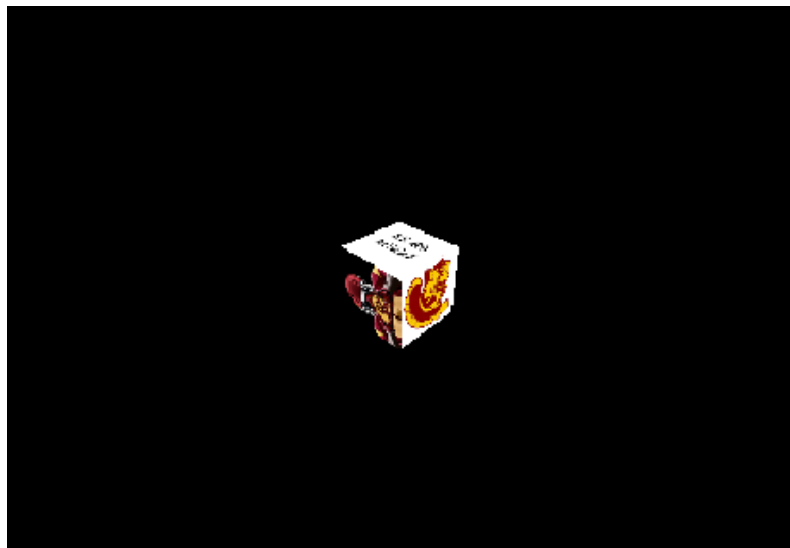
Very high pixel density will result in a very small image, and hence not all the details of the image will be visible.

Very small pixel density will result in a big image hence causing blurred artifacts to result on image captured on image plane.

To improve the result, one could choose an appropriate pixel density, or in the case of blurred artifacts, perform median filtering to get a sharper image.

The image shown in result uses a pixel density of “2/500” or 0.004 . As it can be observed, there are no distortions present and hence no processing has been done to it.

Choosing a pixel density that is high (2/200 will result in a very poor, small image as below)



Problem 3: TEXTURE ANALYSIS AND SEGMENTATION USING LAWS' FILTERS

I. Abstract and Motivation

Although there exist no particular definitions of the word “texture”, one of the ways to define it can - a repeated pattern of statistical appearances. There is a

characteristic “order” within a textural image. A texture can be either artificial or natural – and analysis of texture is a huge domain of Image Processing.

There are many areas that require the analysis of texture – including medical imaging (analyzing the texture of cancerous cells) , analyzing the texture of plants and soils in agriculture, texture of other terrestrial sources, in industrial manufacture of products, as well as analysis of extra-terrestrial land patterns or materials.

Textural analysis, because of the spatial relations a texture exhibits, is a precursor to image segmentation – another important field in Image Processing.

Textural Analysis has two main approaches – “Structured Approach” and “Statistical Approach”. While the former is good to analyze artificial textures, the latter is good for analysis and segmentation of naturally occurring texture features.

“Laws’ filter” approach is a Statistical approach, as described in class, a multichannel feature extraction method.

In order to generate texture features, Laws proposed a “texture-energy” approach that measures the amount of variation within a fixed-size window.

3a – Texture Image Clustering

I. Procedure

Normally, a set of nine 5x5 convolution kernels or masks are used to compute texture energy for a given textural image, which is then represented by a vector of nine numbers for each pixel of the current image under analysis. The masks are L5,E5,W5,S5,R5 .

The masks we are to use in this HW problem are E5,S5,W5.

$$E5 = \frac{1}{6} \begin{bmatrix} -1 & -2 & 0 & 2 & 1 \end{bmatrix}, \quad S5 = \frac{1}{4} \begin{bmatrix} -1 & 0 & 2 & 0 & -1 \end{bmatrix}, \quad W5 = \frac{1}{6} \begin{bmatrix} -1 & 2 & 0 & -2 & 1 \end{bmatrix},$$

Steps to follow are as described as below;

- Generate the nine 5x5 Laws convolution masks using the given E5, S5, W5 filters (an example is given below to generate a E5L5 mask. It is generated by $E5^T * L5$)

$$\begin{bmatrix} -1 \\ -2 \\ 0 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix} = \begin{bmatrix} -1 & -4 & -6 & -4 & -1 \\ -2 & -8 & -12 & -8 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 8 & 12 & 8 & 2 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

- Extend the boundaries of the 12 texture images, and apply all the 9 masks to each image. Therefore, each image yields a 9-dimensional feature vector images .
- The average value of the 9-D feature vector is extracted and this is used in the built-in K-means clustering code of MATLAB “kmeans” to segment into 4 groups of texture images, i.e, to get classified into four groups (which is decipherable visually).

II. Results

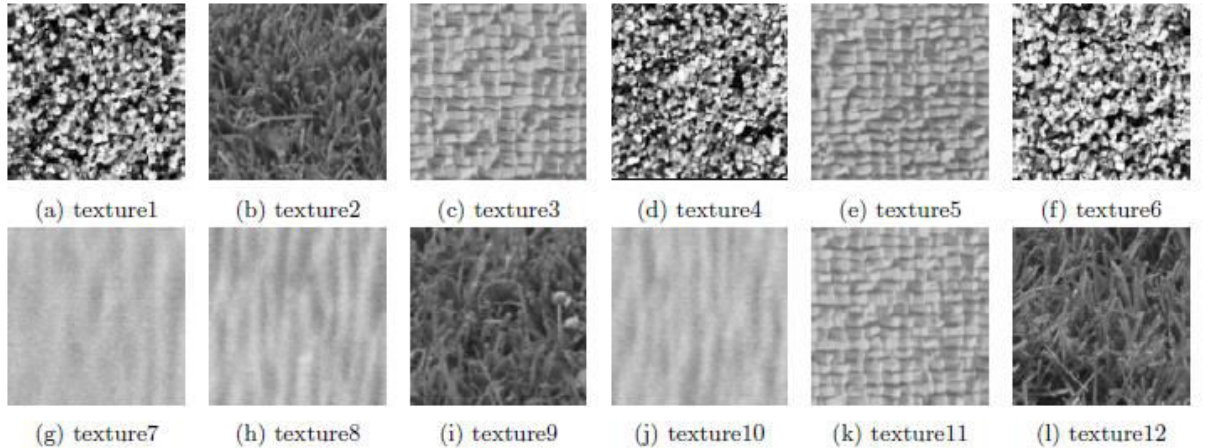
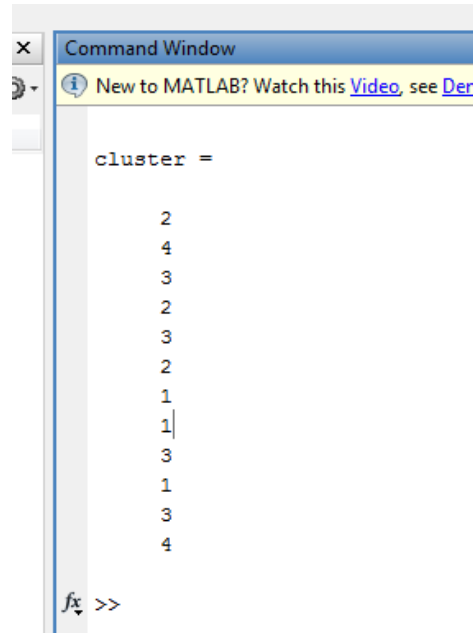


Figure 13 – The input textural images given to be classified.

On running the coded program, the following group numbers are the results obtained on command window

A screenshot of the MATLAB Command Window. The window title is "Command Window". Below the title bar, there is a message: "New to MATLAB? Watch this Video, see Den". The main area of the window displays the output of a clustering operation. It starts with "cluster =" followed by a vertical list of numbers: 2, 4, 3, 2, 3, 2, 1, 1, 3, 1, 3, 4. At the bottom left of the window, there is a prompt "fx >>".

```
cluster =  
  
2  
4  
3  
2  
3  
2  
1  
1  
3  
1  
3  
4
```

Figure 14- group numbers of the texture images after clustering.

group 1 – texture images 7,8,10.

group 2- texture images 1,4,6.

group 3 – texture images 3,5,9,11.

group 4- texture images 2, 12.

III. Discussion and conclusion

As briefly mentioned in the procedure, “kmeans” clustering algorithm is used to classify the texture images into 4 groups (group number verified physically by observing the inputs).

A brief explanation of the “kmeans” clustering algorithm is given in section III. of problem 3b.

As shown in figure 14, the images are classified as

group 1 – texture images 7,8,10.

group 2- texture images 1,4,6.

group 3 – texture images 3,5,9,11.

group 4- texture images 2, 12.

We find that all the texture pictures are correctly classified except the texture image 9. While it actually belongs to group 4, it is classified into group 3.

On observing the image, we find out why – there is a small part of the texture 9 image which has a couple of flowers and its intensity seems to be matching with that of group 3 images but there is more to it than that. The average values of the 9-D feature vectors gotten for picture 9 is the closest to that of pictures 3,5 and 11. Evidence of the same is below.

Texture image 3 avg. feature vector (group 3)	Texture image 2 avg. feature vector (group 4)
5.2234	3.7359
4.2403	2.3426
2.3610	1.1851
1.8478	1.5238
1.0355	0.5398
1.6491	1.0091

3.6684	2.7190
3.1288	1.7581
1.8410	0.9225

Texture image 9

avg. feature vector

4.0424

2.8804

1.4045

1.4437

0.6225

1.1510

2.7717

2.1006

1.0861

Hence it is easy to see that kmeans algorithm ends up clustering the texture image to group 3 instead of group 4.

3b – Texture Segmentation

Brief overview of Texture Analysis and Image Segmentation, significance and motivation have been expressed under the problem heading already.

The goal is to be able to classify and segment textures within a composite image. This is more under the domain of Image segmentation and holds significance for all the areas mentioned in the motivation.

I. Procedure

- Generate 9 Laws Filter masks using 3x3 Kernels.

The masks are as follows

$$\mathbf{L3} = (1/6) * \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

$$\mathbf{E3} = (1/2) * \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

$$\mathbf{S3} = (1/2) * \begin{bmatrix} -1 & 2 & 1 \end{bmatrix}$$

The generated masks are L3L3, E3E3, S3S3, L3E3, L3S3, E3L3, E3S3, S3L3 and S3E3.

They are generated by multiplying the transpose of the first kernel with the second kernel of the masks named above.

- These masks are then applied to the extended composite texture image, to get a 9-D gray level image (texture feature image) for every mask applied.
- To each of the 9-D feature vector of the image size obtained, a windowed approach is used to compute the “energy” of each pixel by using a moving window standard deviation (absolute squared) value. The equation is as follows.

$$E_k[r, c] = \sum_{j=c-7}^{c+7} \sum_{i=r-7}^{r+7} | F_k[i, j] |$$

- To the 9-feature images we get owing to the 9 gray scale images generated per texture and the energy computation, we normalize each resultant image by L3L3 applied feature image (it has zero mean).
- These 9-feature images are then passed to a K-means clustering function to segment into different groups, with number of textures to be set $(K) = 6$.
- Each resultant classified texture is assigned a gray level to denote the results of the texture segmentation.

II. Results

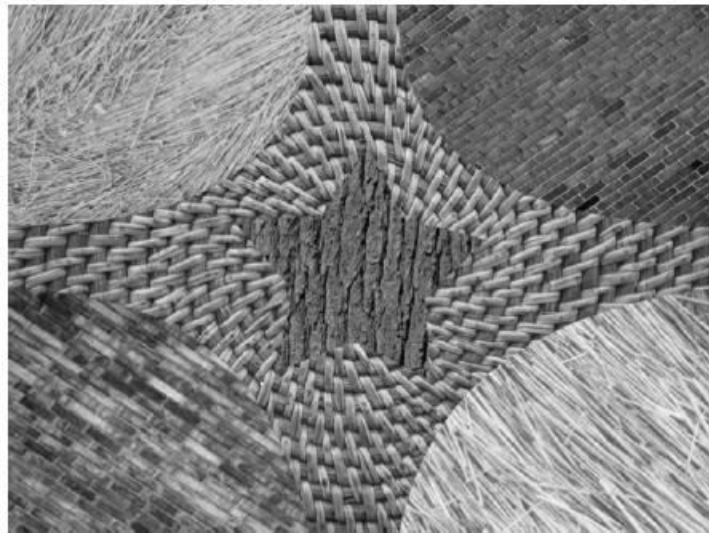


Figure 15- Original composite texture image

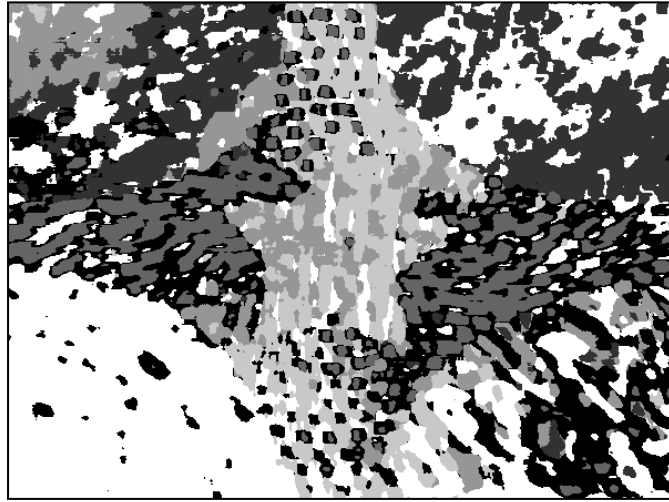


Figure 16- Segmentation with $K=7$, window size= 11×11

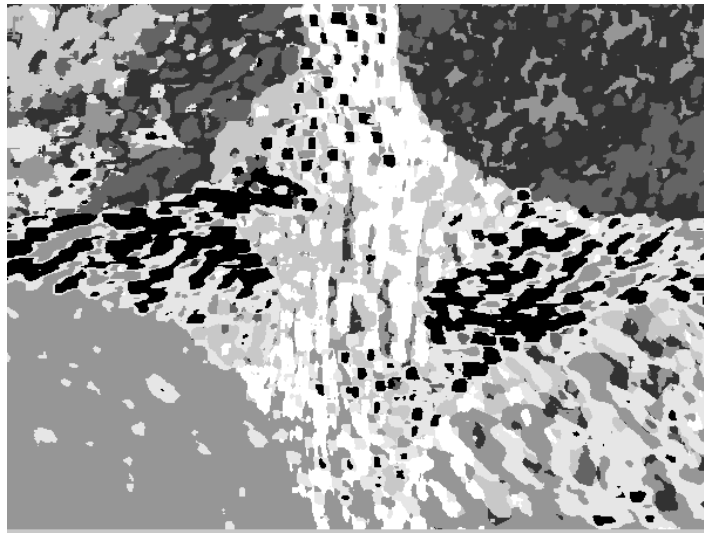


Figure 17- Segmentation with $K=7$, window size= 11×11 , reiterations=5



Figure 18- Segmentation with $K=7$, window size= 21×21 , reiterations= 20

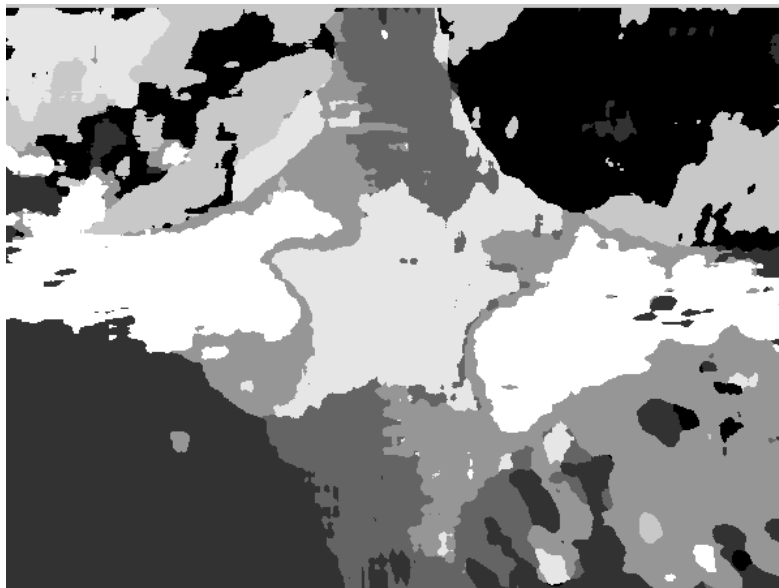


Figure 19- Segmentation with $K=7$, window size= 23×23 , reiterations= 20

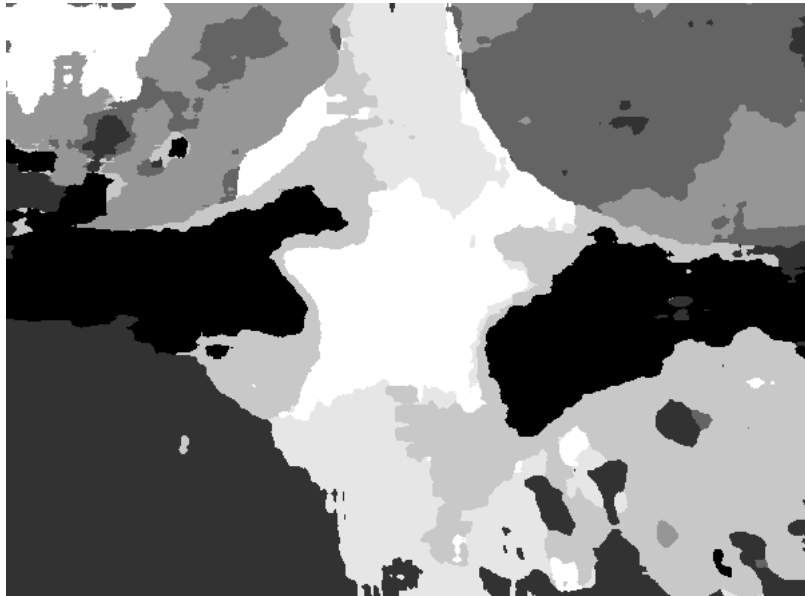


Figure 20- Segmentation with $K=7$, window size= 27×27 , reiterations=20

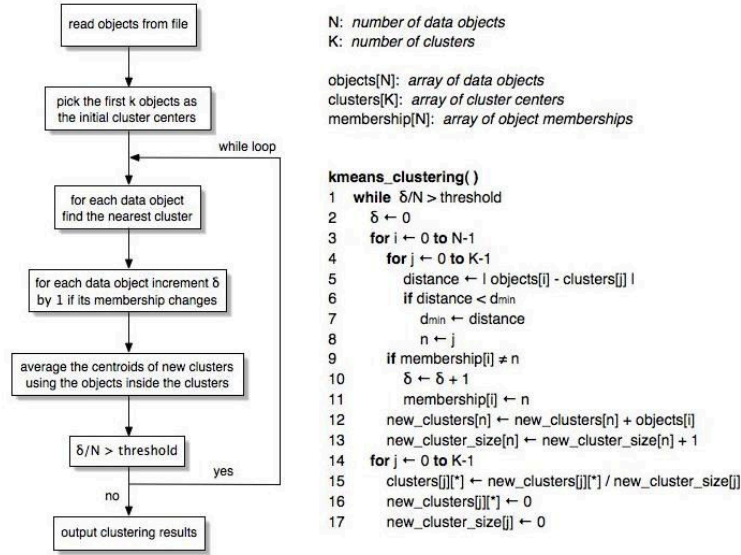


Figure 21- Segmentation with $K=7$, window size= 35×35 , reiterations=20

III.

Discussion and Conclusion

The working of K-Means clustering algorithm can be explained in this algorithm [4] as shown below:



MATLAB provides an inbuilt K-Means function that accepts the data matrix, computes random centroids initially and then provides for reiterations with the newly derived centroids after 1. Centroids serve the purpose of grouping the elements that have the lowest deviation when compared to the centroid value selected. The distance from each element to the centroid is measured, and if the elements belong to that group according to the internally computed threshold, it is clustered into one group.

The results of the procedure followed are shown above. Figure 15 shows the original composite texture picture. Clearly, there are 6 textures present within this picture. However the texture on the top left has two different orientations within itself so 7 groups could be considered in order to give a better segmentation result.

Figure 16 and 17 are the worst segmented results with window size 11x11 and no reiteration of clustering, and window size 11x11 with 20 reiterations. This shows that the window size isn't efficient.

Figure 18, 19 and 20 show the results with better segmentation results compared to the previous results, with window sizes 21x21, 23x23 and 27x27 respectively, and clustering reiterations 20 for all ($k=7$). However, there are still discrepancies. WS 21x21 seems to be better segmenting the star, the diamond region around it well compared to the increased window sizes – 23x23 and 27x27 whereas 27x27 seems to be the best at segmenting the 4 corner regions (including the differently oriented top left corner).

Figure 21 shows the result with window size 35x35, $k=7$ and 20 clustering reiterations – which again is giving very disappointing results as segmented regions are merging together in large area (see star and its surrounding region)

To conclude, window sizes that are in 20s seem to be giving best result with reiterations =20 and $k=7$.

3c- Principal Component Analysis.

Principal Component Analysis (or PCA) is a useful technique that has its foundation set in Linear Algebra. It is used most commonly in tandem with feature vectors for facial recognition, or data compression and pre-processing.

Its properties help in preserving the most important data in a collection of data having a high dimensionality, due to the fact that PCA function exploits the concepts of Eigen values (or vectors) and Covariance matrices.

How PCA works in order to preserve data and find patterns in data (here, with reference to feature vectors got from texture images) while simultaneously reducing dimensionality can be summarized in the following steps [6].

- From the data collected, each element of each data set is subtracted by the mean of that particular data set.

- Calculate covariance matrix , eigen values and hence eigen vectors from the covariance matrix.
- Choose components and hence form a reduced feature vector.

In this problem, we use PCA to reduce the dimensionality after having applied all the 25 Laws filter masks on Figure 16 (composite texture image)

I. Procedure

Procedure remains exactly the same as in 3b , except that we now have 25-D feature vector for the each pixel of the composite texture instead of 9-D vector.

Therefore the data generated is of size 27000x25 matrix that is passed to the inbuilt PCA vector.

The masks are generated as described in section I of 3b. but we use 5x5 kernels rather than 3x3 and all the 5 kernels.

Table 1: 1D Kernels for 5x5 Laws Filters

NAME	KERNEL
L5 (Level)	[1 4 6 4 1]
E5 (Edge)	[-1 -2 0 2 1]
S5 (Spot)	[-1 0 2 0 -1]
W5 (Wave)	[-1 2 0 -2 1]
R5 (Ripple)	[1 -4 6 -4 1]

They are generated by multiplying the transpose of the first kernel with the second kernel of the masks named above.

- These masks are then applied to the extended composite texture image, to get a 25-D gray level image (texture feature image) for every mask applied.

- To each of the 25-D feature vector of the image size obtained, a windowed approach is used to compute the “energy” of each pixel by using a moving window standard deviation (absolute squared) value. The equation is as follows.

$$E_k[r, c] = \sum_{j=c-7}^{c+7} \sum_{i=r-7}^{r+7} | F_k[i, j] |$$

- To the 25-feature images we get owing to the 25 gray scale images generated per texture and the energy computation, we normalize each resultant image by L5L5 applied feature image (it has zero mean).
- These 25-feature images are then passed to a K-means PCA function to reduce dimensionality of feature vectors.
- The reduced feature vectors due to PCA are then passed to Kmeans with K set to 6 or 7 respectively.
- Each resultant classified texture is assigned a gray level to denote the results of the texture segmentation.

II. Results

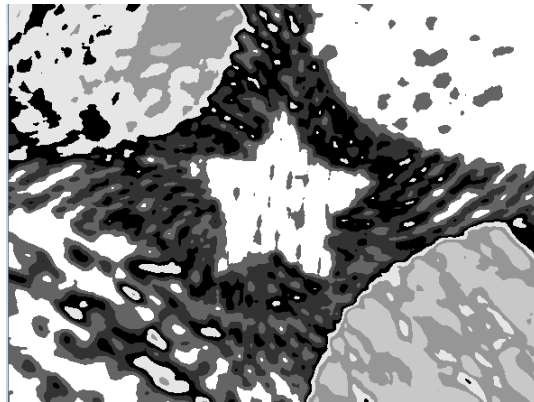


Figure 22- Segmentation with K=7, window size=15x15, reiterations=3

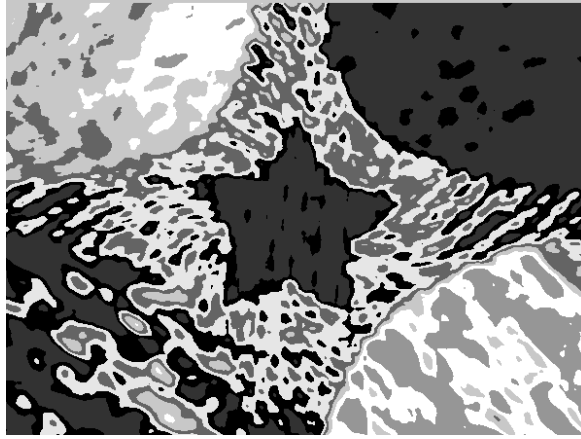


Figure 23- Segmentation with $K=7$, window size= 15×15 , reiterations=20

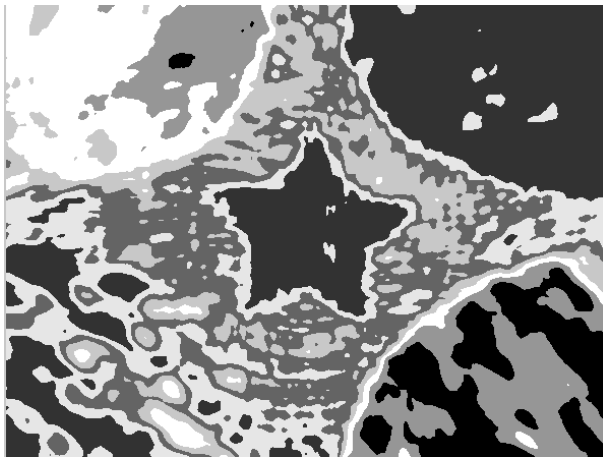


Figure 24- Segmentation with $K=7$, window size= 21×21 , reiterations=10

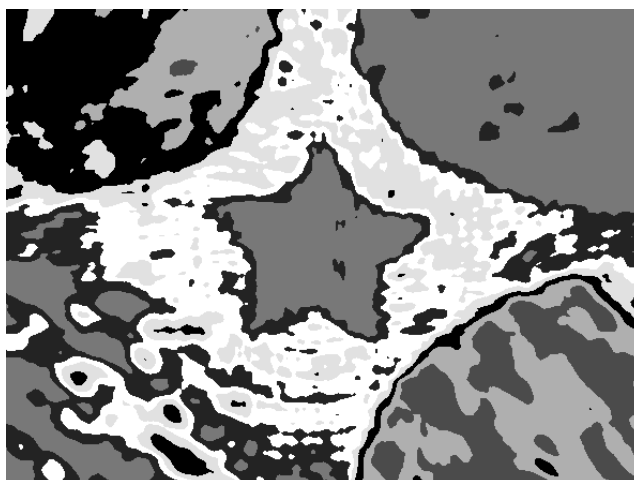


Figure 25 - Segmentation with $K=7$, window size= 21×21 , reiterations=20

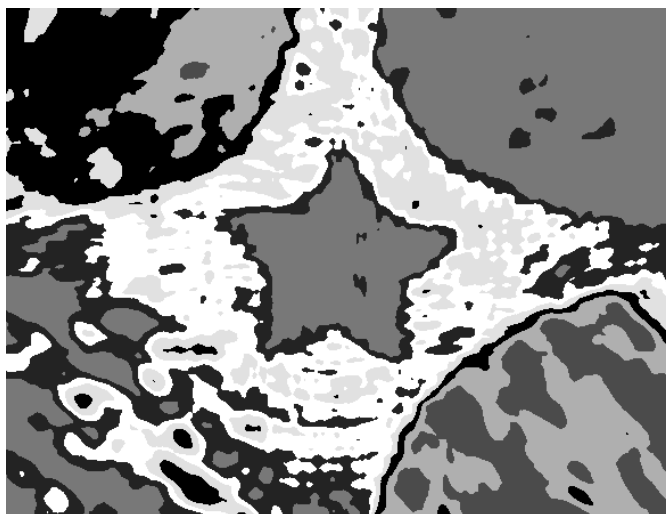


Figure 26- Segmentation with $K=7$, window size= 35×35 , reiterations=20

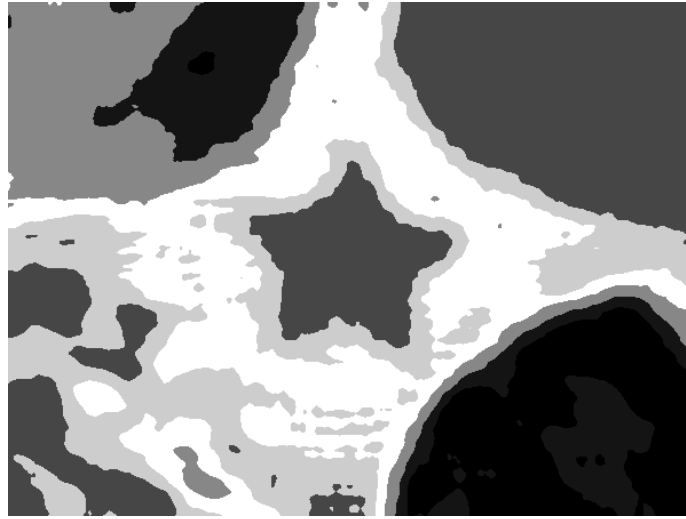


Figure 27 - Segmentation with $K=6$, window size=35x35, reiterations=25

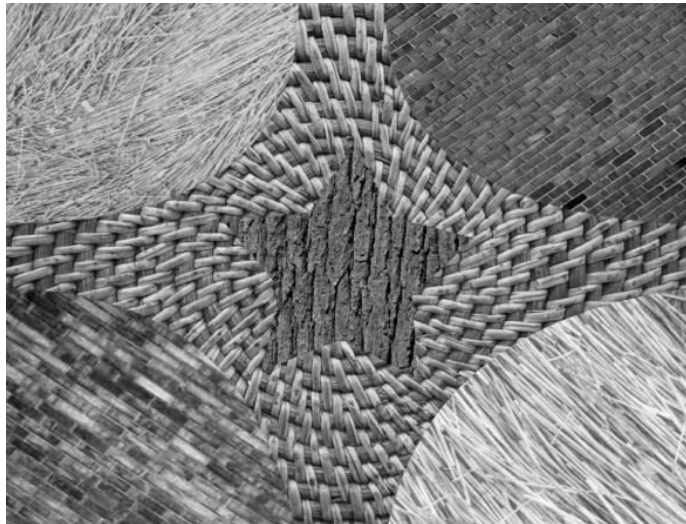


Figure 28 – ‘Comb.raw’ Original composite texture image

III. Discussion and conclusion

Within reason, using PCA itself could be considered an improvement to segmentation results obtained in 3b.

This is evident on comparing **3b** results section and the results obtained for **3c** above. On comparing figures 17,18 and 22,23 – we observe that PCA gives much better results for approximately same window sizes and reiteration values chosen for kmeans in 2b.

We note that PCA requires higher dimension window sizes to give optimum results, this could be related to the fact that more data is required for PCA to calculate more efficient principal components.

Clearly , window size 15x15 hasn't given a good segmentation (Fig 22,23).

Keeping the window size constant (21x21) with k=7 and changing reiteration values 20 to 25 for k-means (figure 24,25) hasn't produced much improvement but the overall segmentation is better than window size 15x15 .

Therefore, the window size is again increased – to 35x35 , but we observe the effect of choosing K=7 vs K=6 and different reiteration values (20 and 25 respectively).

On observing figures 26, and 27 it is clear that compared to K=7, window size = 35x25, reiterations= 20,

K=6, reiterations=25 is giving clear demarcation between different regions. Therefore, for the observed values, even though the bottom left corner region is not properly segmented, Figure 27 is the best observed result.

Note on improvement for segmentation results in 3b – there are two other possible approaches.

- 1) Use dilation for hole filling for similar gray level values assigned in the texture.
- 2) Use 8-connected component analysis for each gray level value in the output segmentation image.

Of course this does not mean segmentation results are improved per se, but they serve the purpose of improving the look of already derived segmentation regardless of whether it has given efficient segmentation or not.

-----END OF REPORT-----

REFERENCES –

- [1] <http://www.gson.org/thesis/warping-thesis.pdf>
- [2] <http://www.codermind.com/articles/Raytracer-in-C++-Part-III-Textures.html>
- [3] <http://courses.cs.washington.edu/courses/cse576/book/ch7.pdf>
- [4] <http://www.ece.northwestern.edu/~wkliao/Kmeans/>
- [5] http://www.cs.tufts.edu/~sarasu/courses/comp175-2009fa/project/project-billingham/NLA_CG_termPaper2009.pdf
- [6] http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf

