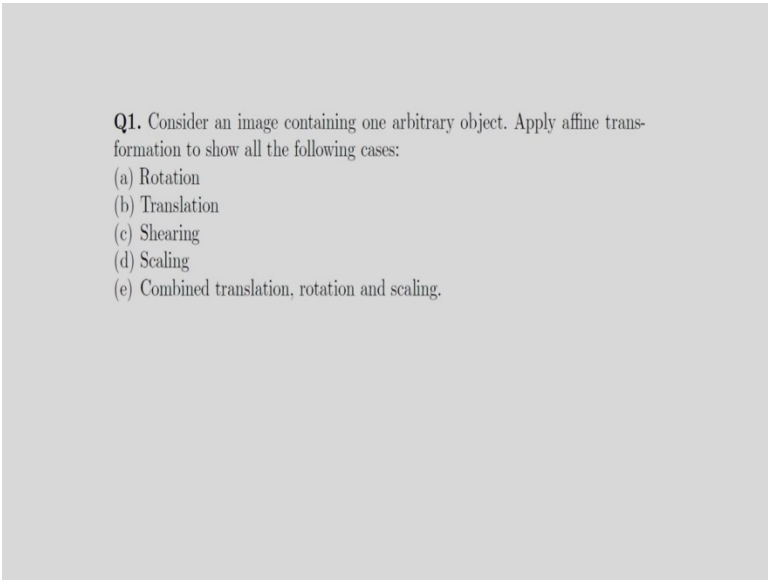


**Computer Vision and Image Processing – Fundamentals and Application**  
**Professor Doctor M. K. Bhuyan**  
**Department of Electronics and Electrical Engineering**  
**Indian Institute of Technology Guwahati India**  
**Lecture 41**  
**Programming Examples**

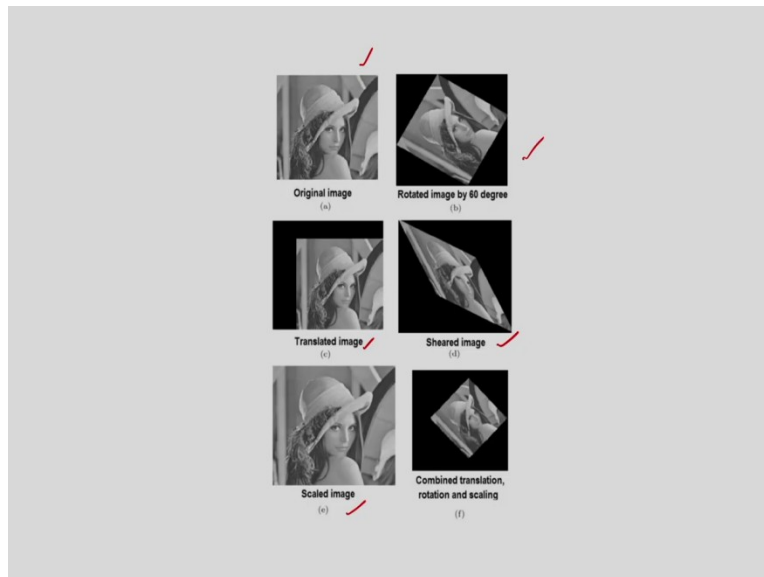
Welcome to NPTEL MOOCs course on Computer Vision and Image Processing Fundamentals and Applications. This is my last class of this course. In this class I will give some programming assignments. So, mainly I will discuss twenty programming assignments, the twenty programming problems. These programming assignments you can do either in MATLAB or maybe open CV Python or maybe any other programming languages. So, one by one all these programming assignments I will discuss now. So, let us start now.

(Refer Slide Time: 01:01)

- 
- Q1. Consider an image containing one arbitrary object. Apply affine transformation to show all the following cases:
- (a) Rotation
  - (b) Translation
  - (c) Shearing
  - (d) Scaling
  - (e) Combined translation, rotation and scaling.

So, first programming assignment is, consider an image containing one arbitrary object and after this apply affine transformation to show all the following cases. So, first we have to show the rotation, after this translation, shearing, scaling and also the combined translation, rotation and scaling. So, this concept already I have explained in my class, the concept of the affine transformation.

(Refer Slide Time: 01:26)



So, your output will be something like this, you can see I have the original image. This image can be rotated by 60 degree. And also this image can be translated. So, this is the translated image. This image can be sheared, that shearing can be done. That means, the scaling is not any form throughout the image, that is called shearing. Also we can do the scaling, scale image you can see.

And also we can do the combined translation, rotation and scaling. So, this is the concept of the affine transformation. So, for this you have to write a program, the program in MATLAB or maybe you can use Python, the open CV Python also you can use.

(Refer Slide Time: 02:12)

```
1 clc
2 clear all
3 close all
4
5 im = imread('lena.jpg'); %reading Lena image ✓
6 im = imresize(im,0.5); %resizing to make the computation fast
7 imshow(im);
8 title('Original Image'); %showing Original image
9
10 %% (a) Rotation
11 degree=60; ✓
12 theta = degree*pi/180; ✓
13 R = [cos(theta) sin(theta); -sin(theta) cos(theta)]; %Matrix ...
      for anticlockwise Rotation
14
15 %% Finding min and max of the Coordinates for avoiding ...
      negative indexes
16 minx=0;
17 miny=0;
18 for i=1:size(im,1)
19     for j=1:size(im,2)
20         A = [i j]*R;
21         if (A(1)<minx)
22             minx=round(A(1));
23         end
24         if (A(2)<miny)
25             miny=round(A(2));
26         end
27     end
28 end
29
```

So, in this case, I am showing one MATLAB program. So, here you can see first I am reading the image, the image is the Lena image. And after this what I am doing, the resizing the image. So, that resizing to make the computation first, after this I am showing the original image. After this I am considering the rotation of the image. So, 60 degree I am considering. So, first I am doing the rotation and this is the transformation matrix for the rotation.

So, you know the transformation matrix for the rotation is  $\cos \theta$   $\sin \theta$  minus  $\sin \theta$   $\cos \theta$ . And also we have to see the minimum and the maximum coordinates for avoiding negative coordinates. So, that we have checked this, in this part.

(Refer Slide Time: 02:56)

```
30 % finding rotated image coordinates and replacing pixel y ...
31 original image
32 for i=1:size(im,1)
33     for j=1:size(im,2)
34         A=[j]*R - (size(im,1)-1);
35         result(round(A(1)),round(A(2))) = im(i,j);
36     end
37 end
38 result = medfilt2(result);
39 figure();
40 imshow(result);
41 p=round(pi/degree);
42 awstrcat('Rotated Image by ', p, ' degrees');
43 title(a);
44
45 % (b) Translation
46 delx=50;
47 dely=70;
48
49 T = [1 0; 0 1]; %Translation matrix
50 B=[delx;dely];
51
52 % finding Translated image coordinates and replacing pixel by ...
53 original image
54 for i =1:size(im,1)
55     for j=1:size(im,2)
56         A = T*(i,j) + B;
57         result2(A(1),A(2)) = im(i,j);
58     end
59 end
60
61 figure();
62 imshow(result2);
63 title('Translated Image');
64
65 % (c) Shearing
66 shx=0.5;
67 shy=0.5;
68 Sh = [1 shxshy 1]; %Shearing Matrix
69
70 % finding Sheared image coordinates and replacing pixel by ...
71 original image
72 for i =1:size(im,1)
73     for j=1:size(im,2)
74         A = Sh*(i,j);
75         result3(round(A(1)),round(A(2))) = im(i,j);
76     end
77 end
78
79 result=medfilt2(result3);
80 figure();
```

After this, I am doing the rotation. So, I am finding the rotated image based on the transformation matrix. So, this rotated image can be displayed. After this I am considering the translation. So, for the translation I am considering the translation vector, the translation vector  $e$  is equal to 10, 01 and based on this I am translating the image, the input image. So, I can do the translation and I can show the result of the translation. So, this is the imshow result 2.

After this I am considering the shearing, shearing means the scaling is not any form throughout the object. So, for shearing, I am considering this. So, scaling along the x direction, scaling along the y direction I am considering. And from this I am considering the shearing matrix. After this, I am doing this one, these computations.

(Refer Slide Time: 03:54)

```
81 imshow(result3);
82 title('Shearing Image');
83
84 %% (d) Scaling
85
86 sx=2;
87 sy=2;
88
89 S = [sx 0 0; 0 sy 0]; %Scaling matrix
90
91 for i = 1:size(im,1)
92     for j = 1:size(im,2)
93         A = S*(i,j);
94         result4(round(A(1)),round(A(2))) = im(i,j);
95     end
96 end
97 if (sx<1 || sy>1)
98     result4=ordfilt2(result4,9,ones(3,3));
99 else
100     result4=medfilt2(result4);
101 end
102 figure();
103 imshow(result4);
104 title('Scaled Image');
105
106 %% Combined Translation, rotation and Scaling.
107
108 sx=0.5;
109 sy=0.5;
110 delayx=0;
111 delayy=70;
112 theta = 45*pi/180;
113 T = [1 0 0; 0 1 0; 0 0 1];
114 B=[delayx;delayy];
115 S = [sx 0 0; 0 sy 0];
116 R = [cos(theta) sin(theta); -sin(theta) cos(theta)];
117
118 for i = 1:size(im,1)
119     for j = 1:size(im,2)
120         A = T*(i,j)+B;
121         result5(round(A(1)),round(A(2))) = im(i,j);
122     end
123 end
124
125 minx=0;
126 miny=0;
127 for i = 1:size(result5,1)
128     for j = 1:size(result5,2)
129         A = [i j]+B;
130         if (A(1)<minx)
131             minx=round(A(1));
132         end
133         if (A(2)<miny)
```

And finally, you can see I am showing the result of shearing, imshow result 3, that is the result of shearing operation. And after this, what am I considering? The scaling of the image. So, so, scaling along the x direction, scaling along the y direction I am considering. And after this I am doing the scaling and after scaling, I can show the result of the scaling operation. So, you can see result 4, that is nothing but the result of the scaling operation.

After this, what am I considering? I am considering the affine transformation, considering translation, rotation and the scaling. So, for this I am defining the parameters, all the parameters I am defining, like scaling, translation parameter and also the rotation parameters I am defining. And based on this I am doing the combined operation.

(Refer Slide Time: 04:46)

```
135     miny=round(A(2));
136     end
137 end
138 end
139
140 for i=1:size(result5,1)
141     for j=1:size(result5,2)
142         A=[i j]*R - [minx-1 miny-1];
143         result6(round(A(1)),round(A(2))) = result5(i,j);
144     end
145 end
146 result6 = medfilt2(result6);
147
148 for i =1:size(result6,1)
149     for j=1:size(result6,2)
150         A = S*[i;j];
151         result7(round(A(1)),round(A(2))) = result6(i,j);
152     end
153 end
154 if (sx>1 || sy>1)
155     result7=ordfilt2(result7,9,ones(3,3));
156 else
157     result7=medfilt2(result7);
158 end
159 figure();
160 imshow(result7); ✓
161 title('Combined translation, rotation and Scaling.');
```

And after this finally you can see, I am showing the result. The result is, result 7. That is the result corresponding to combined translation, rotation and scaling. So, this is about the first assignment. So, in the first assignment, the problem is I have to do the affine transformation of the image.

(Refer Slide Time: 05:06)

Q2. Write a MATLAB program to show perspective, weak perspective and orthographic projections of an object in an image. Make suitable assumptions if required.

(a) Original 3D cube ✓

(b) Perspective projection ✓

(c) Orthographic projection ✓

(d) Weak perspective projection ✓

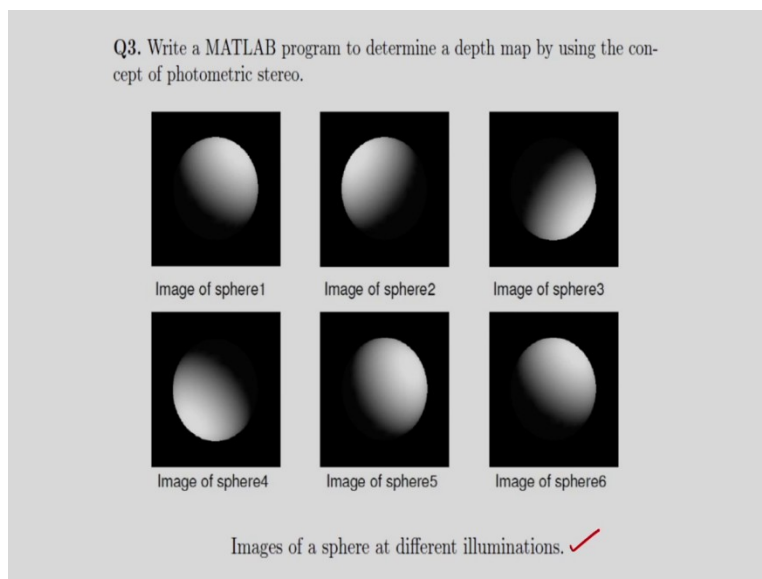
$(x, y, z) \rightarrow (x, y)$

The second problem is writing MATLAB program to show perspective, weak perspective and orthographic projections of an object in an image. And for this make suitable assumptions if required. So, I am showing the original 3D cube and you can see the output corresponding to perspective projection. So, first one is the perspective projection. Next one is the orthographic projection and the last one is the weak perspective projection.

So, in the case of the perspective projection, I have already explained that distant objects appear smaller. In case of the orthographic projection, it is nothing but  $x, y, z$  is projected onto  $x, y$  coordinate. That is the orthographic projection. And in case of the weak perspective projection, it is nothing but the scaling of the  $x$  coordinate, scaling of the  $y$  coordinate like this and the scaling of the  $z$  coordinate.

So, that is the weak perspective projection. So, based on some assumptions, you can show the perspective projection, orthographic projection and weak perspective projection. There is a mistake here in the spelling, it should be perspective, it should be perspective. So, this is the second assignment. So, you can do it in the MATLAB.

(Refer Slide Time: 06:32)



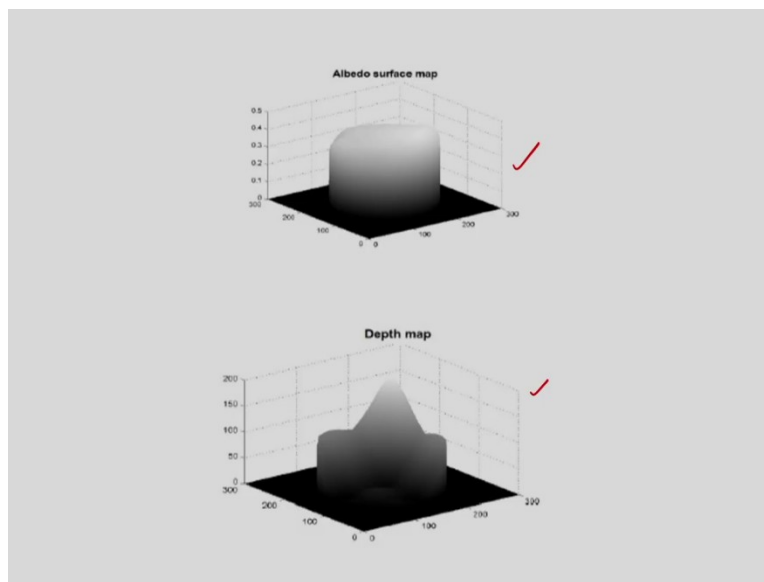
Third problem is, write a MATLAB program to determine a depth map by using the concept of photometric stereo. So, this problem is already I have explained that is, the concept of the shape from shading. The shape from shading problem can be solved by the concept of photometric

stereo. So, for this I am considering different light sources and shading information is used to determine the depth information.

So, different illumination conditions or different lighting conditions I am considering and you can see, I am getting the image of sphere one, sphere two like this. Because I am considering different lighting conditions. That means, I am getting the shading conditions. So, from all these images, I have to get the depth information, that is the concept of the photometric stereo. So, images of a sphere at different illuminations.

So, different illuminations I am considering and based on this I am getting the shading. So, from the shading information, I have to get the depth information.

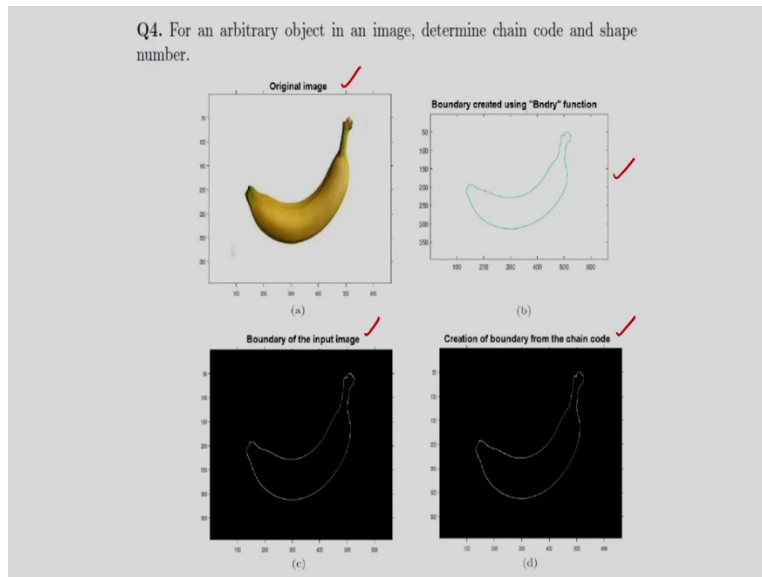
(Refer Slide Time: 07:34)



So, here first based on the principle of the photometric stereo, you can determine the albedo surface map. The albedo also you can determine, and from the albedo you can determine the depth map of the object. So, the principle of the photometric stereo has already I have explained in my classes. So, this is the output of this assignment, that is you can determine the albedo of the surface map, albedo you can determine. And also from this you can determine the depth map. This is about the third assignment.



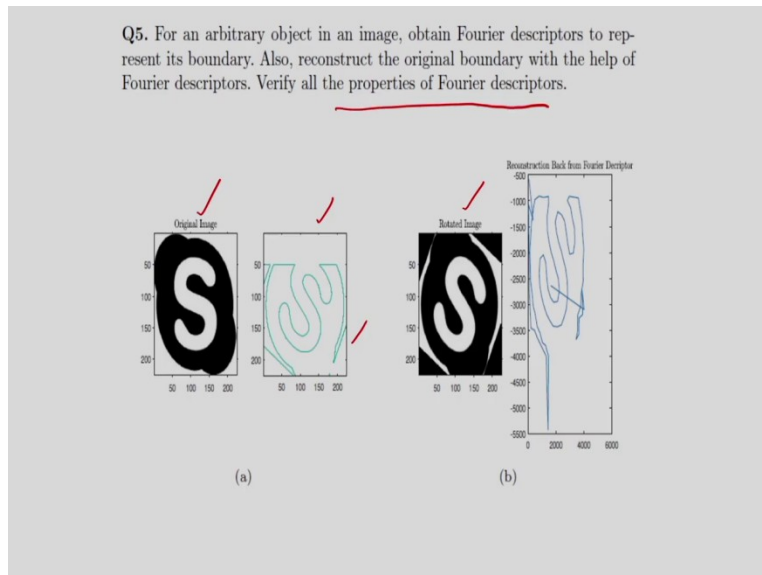
(Refer Slide Time: 08:10)



The next assignment is for an arbitrary object in an image, determine chain code and shape number. So, here I am showing the original image and this image or the this boundary of the object, I have to represent by using the chain code. So, for this you can see I am showing the boundary of the object, that is the boundary of the input image. That can be represented by the chain code.

The concept of the chain code has already I have explained. And after this I can reconstruct the boundary from the chain code. So, in the fourth result, that is the fourth image, that is the reconstruction of the boundary from that chain code. So, this problem you can try, that is first from the boundary information, you can determine the chain code and again from the chain code you can reconstruct the original boundary.

(Refer Slide Time: 09:10)

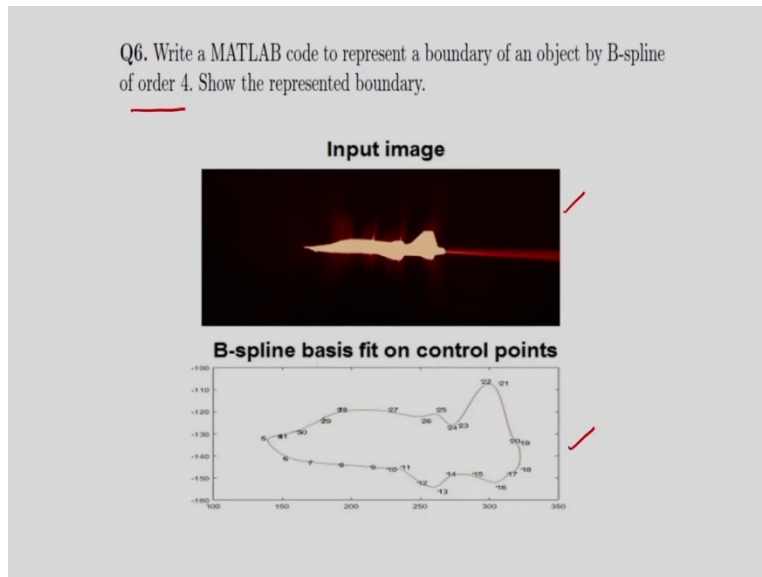


Question number five is for an arbitrary object in an image, obtain fourier descriptors to represents its boundary. So, that means the fourier descriptor we have to use to represent a particular boundary. So, you can see, this is the original image. And you can see the boundary of the original image, that can be reconstructed by the fourier descriptors. So, corresponding to this original image, the boundary can be represented by the fourier descriptors.

After this, this boundary can be reconstructed by considering inverse fourier transform, that is by considering the fourier coefficients, I can reconstruct the original boundary. That is the reconstructed boundary. Similarly, I am considering the rotated image and corresponding to this I am reconstructing the boundary with the help of the fourier descriptors. So, for the rotated image also we can determine the fourier descriptors.

After this, by using these fourier descriptors, we can reconstruct the boundary. So, all the properties of the fourier descriptors can be verified, because there are many properties of the fourier descriptors like scaling, rotation, translation. So, all the properties I have discussed in my class. So, all these properties you can verify with the help of these images. So, you take one image and you try to verify all the properties of the fourier descriptors. So, this is question number five.

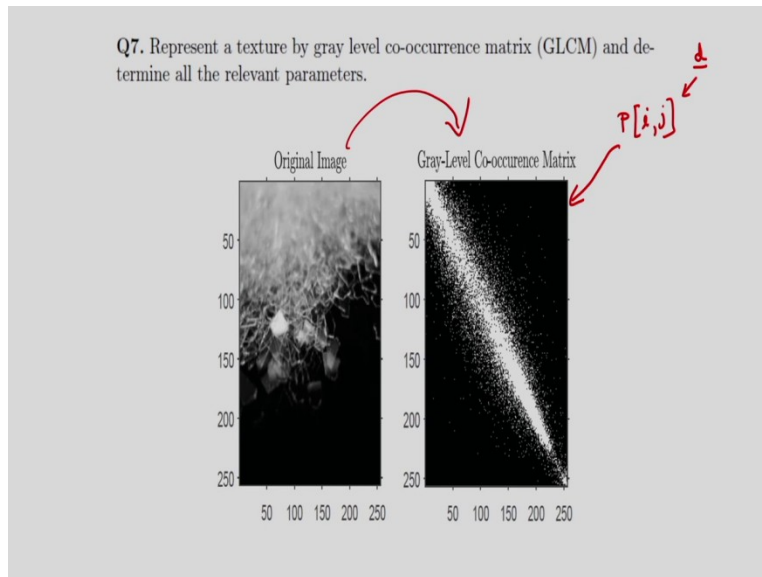
(Refer Slide Time: 10:40)



After this the next question is, writing MATLAB code to represent a boundary of an object by B-spline of order four, show the represented boundary. So, here I am considering the input image and you know that in the case of the B-spline, I have to consider some control points. And also the important point is the order of the B-spline. So, here I am considering the order is four. So, with these B-spline curves, I can represent the boundary with the help of the control points.

So, in the picture you can see here, I am showing the control points. And after this between these control points, I am considering the B spline curve of order four. And like this you can represent the boundary of the input image. So, this problem also you can see, that is, the representation of the object boundary by B-spline curve. So, that is one important problem, the B-spline curve.

(Refer Slide Time: 11:42)



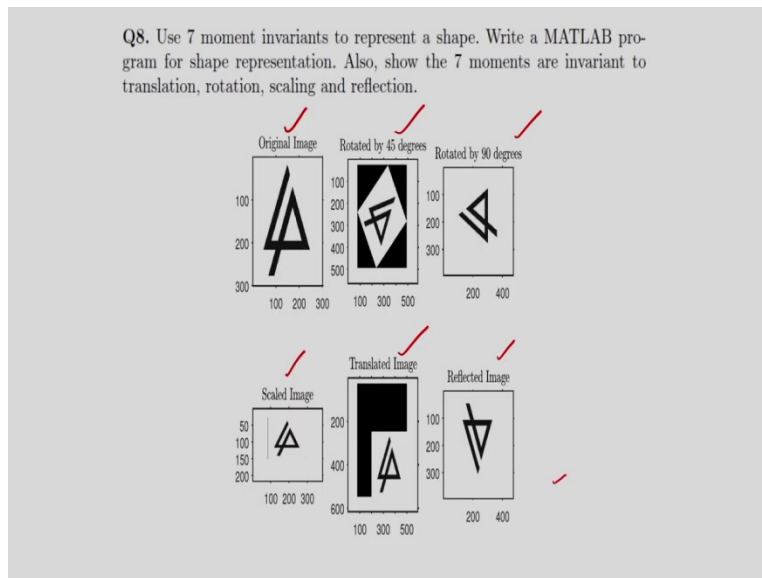
The problem number seven is, represent a texture by gray level co-occurrence matrix. So, that is GLCM and determines all the relevant parameters. So, in my class I have discussed how to determine the GLCM that is the gray level co-occurrence matrix. So, for a particular displacement vector, you can determine the gray level co-occurrence matrix that is the array, the  $P[i, j]$  you can determine. And that is displayed as an image.

So, for a particular displacement  $d$ , I can determine the co-occurrence matrix, the  $P[i, j]$ . That array I can determine and that is displayed as an image, that is the gray level co-occurrence matrix. And from this I can determine the parameters, the parameters maybe the maximum probability I can determine, moments I can determine, the contrast I can determine, entropy I can determine, uniformity I can determine, homogeneity I can determine.

So, all these parameters I can determine. And this GLCM is computed from several values of  $d$ . So, the GLCM can be computed from several values of  $d$ , and the, the one who is maximizes a statistical measure computed from  $P[i, j]$  is ultimately used. So, that means I am repeating this: The GLCM is computed from several values of  $d$ , that is the displacement vector. And the one which maximizes a statistical measure computed from  $P[i, j]$  that is the GLCM is finally employed or finally used.

So, that is the concept of the GLCM. So, we can determine all the relevant parameters, the parameters like the maximum permeability, moments entropy, all these parameters I can determine from the GLCM. So, corresponding to this original image, I can determine the GLCM matrix. So, this is problem seven.

(Refer Slide Time: 13:52)

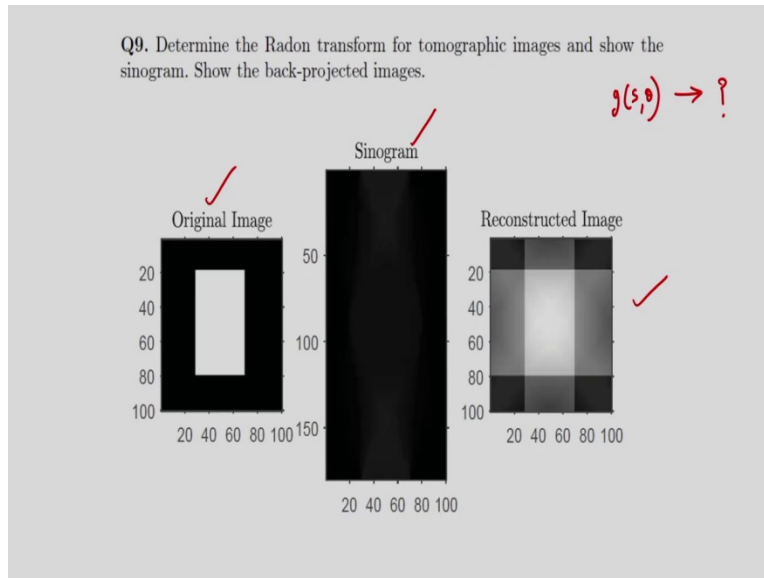


Next the problem is, use seven invariance that is the moment invariance to represent a shape. And for this write a MATLAB program for shape representation. Also show the seven moments are invariant to translation, rotation, scaling and reflection. So, corresponding to this original image, you can determine seven moments, that is seven moment invariants you can determine to represent a particular shape. So, corresponding to this original image, you can determine seven moment invariants.

After this, we have to prove them, the seven moments are invariant to translation, rotation, scaling and reflection. So, this image can be rotated. In this case I am rotating the image by 45 degree, rotated by 90 degree I can do the scaling, I can do the translation and also I can do the reflection of the original image. So, for all these cases I can determine seven moment invariance. And from this I can prove the seven moments are invariant to this transformation, the transformation are translation, rotation, scaling and reflection.

So, the problem is corresponding to these images I can determine seven moment invariance. And with the help of the seven moment invariance, we have to prove that these moments are invariant to translation, rotations, scaling and reflection. So, this is one important problem. So you can write a MATLAB program or maybe the Python program for this assignment. This is about question number eight.

(Refer Slide Time: 15:40)



Next one is question number nine. Determine the radon test from for tomographic images and show the sonogram, show the back projected images. So, suppose if I consider the original image, this is the original image, I can determine  $g(s, \theta)$  that is nothing but a radon test from the input image. So that I have to determine,  $g(s, \theta)$ . And the  $g(s, \theta)$  can be represented as an image, and that is called the sinogram. So, how to determine the radon transform?

We have to compute  $g(s, \theta)$ , there are two parameters one is  $s$  another one is  $\theta$ . After determining  $g(s, \theta)$ , we can display the  $g(s, \theta)$  as an image and that is called the sonogram. After this I am considering the reconstruction of the image. And in this case, I am considering the back projection technique. So, for different angles  $\theta$ , I am doing the back projections. So, I will be getting the back projected images.

So, from the number of back projected images, I can reconstruct the original image. So, for this I have to determine that ray sum, so ray some soul also I have to determine. And this ray some or

the, this GST data can be displayed as an image and that is called a sonogram. And reconstruction is nothing but four different theta, I have to determine the back projected images and from the back projected images, if I combined all the back projected images, I can reconstruct the original image.

So, this is the reconstructed image by considering the back projection technique. There are two techniques, in my class I have explained, one is the back projection technique, another one is the fourier transform technique. But in this case, I am considering the back projection technique, you can also apply the fourier transform technique.

(Refer Slide Time: 17:30)

```
1 clear
2 close all;
3 close all;
4 %A rectangular white patch is created
5 imp=ones(100,100);
6 imp(20:60,30:70)=255;
7 subplot(1,2,1);imshow(imp);title('Original Image'); ...
8 %Visualization of the rectangular image created
9
10 %Ray sum is found out for all the angles from 0 to 180 degree
11 for theta=0:180
12     %theta=0,180 ...
13     (sum(sum(rotate(imp,theta,'bilinear','crop'))/540));
14 end
15 subplot(1,2,2);imshow(imp);title('Sinogram'); %Plotting ...
16 %of the sinogram
17
18 ninety_degree = sum(imp); %Finding out the ray sum along ...
19 %ninety degree
20 zero_deg = sum(imp); %Finding out the ray sum along ...
21 %zero/180 degree
22 no_of_row = size(imp,1); %Finding the number of rows of the ...
23 %input image
24 no_of_col = size(imp,2); %Finding the number of columns of ...
25 %the input image
26
27 no_of_elements = no_of_row * no_of_col-1; %Number of ...
28 %elements when the ray sum is calculated along 45/135 degree
29
30 for i=1:no_of_row %Creating the index vector with each ...
31 %element corresponding to the index of rows of the matrix
32 x(i)=i;
33 end
34 for j=1:no_of_col %Creating the index vector with each ...
35 %element corresponding to the index of columns of the matrix
36 y(j)=j;
37 end
38
39 %Calculating the ray sum along 45 degree
40 sum=0;
41 for i=1:no_of_elements+1
42     sum=0;
43     for j=1:no_of_row
44         [m,n]=ind2sub(
45             no_of_row,
46             no_of_col,
47             i);
48         sum=sum+imp(m,n);
49     end
50 end
```

So, corresponding to this I am showing the program. So, first I have to read the original image. So, this is the original image I am showing. After this, what am I considering? I am finding the ray sum in different directions, in 90 degree directions, in a lambda zero divided by 180 degree like this, all the directions I am determining the ray sums. So that means I am calculating the ray sum.

(Refer Slide Time: 17:57)

```
end
end
fortyfive.degree(i)=sum1;
end
end
% Calculating the ray sum along 135 degree
sum2=0;
for i=1:nrof.row-1:1:-1:nrof.row-1
    sum2=0;
    for j=1:nrof.col
        for b=1:nrof.col
            if(i-b==1)
                sum2=sum2+img(x(i),y(b));
            end
        end
    end
end
sumthirtyfive.degree(i)=sum2;
end
end
% Creation of the back-projected matrix from ray sums along ...
% directions of 0 degree, 45 degree and 135 degree
reconstruct = repmat(reco.degree*1, nrof.col);
repmat(sum2y.degree, nrof.col, 1) * ...
repmat(reco.degree*1, nrof.col);
% Adding the rayon along 0 degree to the back-projected ...
% matrix
for j=1:nrof.col
    for i=1:nrof.col
        reconstruct(x(i),y(i))=reconstruct(x(i),y(i)) + ...
        fortyfive.degree(i);
    end
end
end
end
% Adding the rayon along 135 degree to the back-projected ...
% matrix
for i=1:nrof.col-1:1:-1:nrof.col-1
    for j=1:nrof.col
        for b=1:nrof.col
            reconstruct(x(j),y(i))=reconstruct(x(j),y(i)) + ...
            sumthirtyfive.degree(i);
        end
    end
end
end
end
reconstructed_image = (255/nof(nof(reconstruct)))*reconstruct; ...
% Mapping the reconstructed matrix values from 1 to 255
subplot(2,2,3)
imshow(uint8(reconstructed_image));
title('Reconstructed Image'); % Writing the reconstructed image
```

After calculating the ray sums, what I am considering? So, ray sums lambda 135 degree, adding the ray sums 45 degree like this. So, I am showing the reconstructed image, because I am doing the back projection at different angles. And from this back projected image, I can reconstruct the original image. So, that means I am considering the ray sum, ray sum in different angles, different directions.

And after this, I am reconstructing the original image from this back projected images, that is about the sonogram and the reconstructed image. So, for the sonogram I have to determine the  $\theta$  theta.



(Refer Slide Time: 18:38)

Q10. Write a program to implement K-L transform of an image, and show the transformed image. Represent the image in terms of the eigen images and show the results.

(a) Input Image

(b) Transformed Image

(c) Inverse Transformed Image

Handwritten notes:

$$y = A(x - \mu_x)$$

$$x = A^T y + \mu_x$$

Diagram:  $x \rightarrow A_k \rightarrow C_k \rightarrow e_i \rightarrow \lambda_i \rightarrow A$

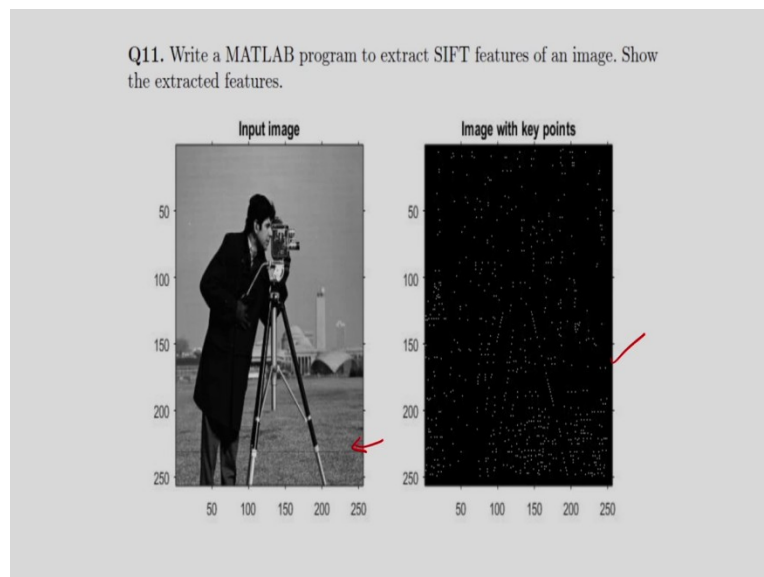
The next program is, write a program to implement K L transform of an image. And show the, transform image, represent the image in terms of Eigen images and show the results. So, in the K L transformation, so what is it the transformation? Y is equal to a  $x - \mu_x$ , that already I have explained. So, a is the transformation matrix, this transformation matrix A, that is determined from the Eigen vectors of the covariance matrix.

So, x is the input vector. So, from the input vector, I can determine the mean and also I can determine the covariance. From the covariance matrix I can determine the Eigen values and Eigen vectors. So, e i is the Eigen vectors and lambda i is the Eigen values and from this I can determine the transformation matrix. So, a is the transformation matrix. So, if I apply thios transformation, y is equal to a  $x - \mu_x$ , then I will be getting the transformed image something like this.

So, this is the transform image. After this I can reconstruct the original image. If I consider all the Eigen vectors and all the Eigen values, then the perfect reconstruction is possible. So, what is the reconstruction formula? The reconstruction formula is a transpose  $y + \mu_x$ . So, by using this I can reconstruct the original image. So, already I have explained that if I consider all the Eigenvalues, Eigen vectors, then I will be getting the original image.

But if I consider the truncated transformation matrix, then in this case I will, I will not be getting the original image, I will be getting the approximated image. So, if I consider all the Eigen values and the Eigen vectors in the transformation matrix, then in this case, the perfect reconstruction is possible. But if I consider only some Eigen vectors for the transformation matrix, then the perfect reconstruction may not be possible. So, this is about the image reconstruction, in case of K L transformation.

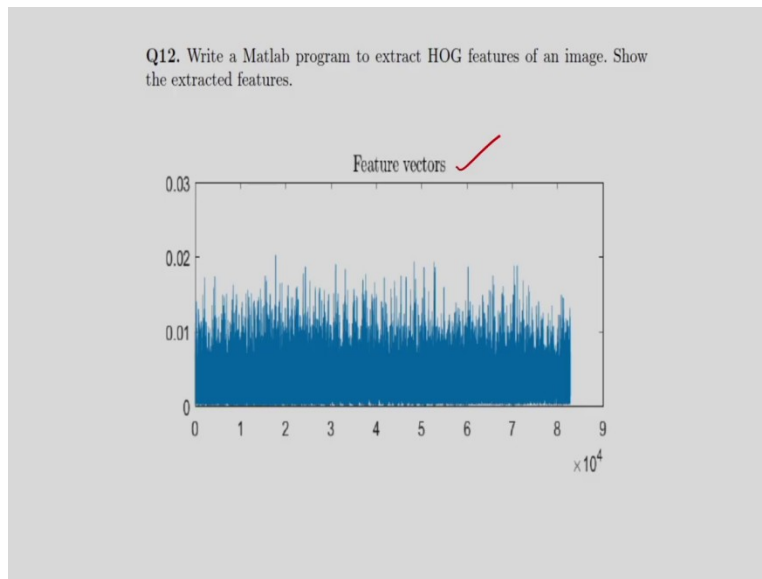
(Refer Slide Time: 20:49)



The next problem is write a MATLAB program to extract SIFT features of an image. So, the extracted features. So, first what you have to consider? First I have to consider the skill spedge representation, after this I have to consider the difference of Gaussian and from the difference of Gaussian, roughly I can look at the key points. Because I have to, I have to find the extrema, I have to find a maxima or the minima of the difference of Gaussian. And from this, I can determine the roughly, the key points.

After this I have to neglect the low contrast key points and also the edge pixels corresponding to the key points. So, by considering the hessian matrix, I can remove the edge pixels which are detected as key points. After this, I have to do the orientation assignments and finally, the SIFT descriptors I will be getting. So, here you can see I am showing the key points corresponding to this input image. These key points are the SIFT features. So, I will be getting the SIFT features and I will be getting the descriptor and the size of the descriptor is generally 128.

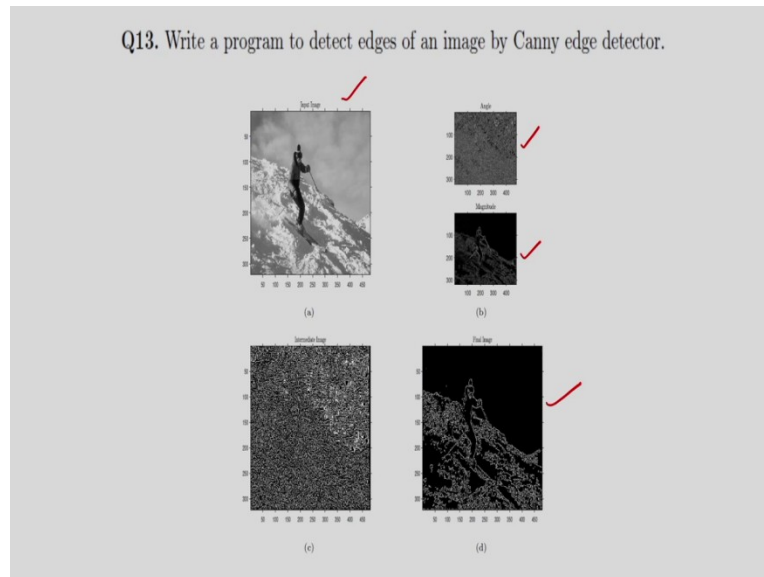
(Refer Slide Time: 22:06)



Question number twelve, write a MATLAB program to extract HOG features of an image. So, the extracted features. So, hog principle already I have explained in my class, the histogram of oriented gradient. So, you can get the feature vectors based on the gradients. So, gradient of an image you can determine and from the orientation of this gradient, you can determine the feature vector, corresponding to the input image.

So, you take one input image and corresponding to this, you can determine the HOG features. And with the HOG features, you can detect objects present in an image, that is nothing but object detection. That is question number twelve.

(Refer Slide Time: 22:48)



Question number thirteen, write a program to detect edges of an image by canny edge detector. So, this problem is the detection of the edges by canny edge detector. So, here I am showing the input image, for canny edge detection, first I have to do the Gaussian blurring, that is the image is convolve with the Gaussian, that means the noises are removed. And after this, I have to determine the gradient magnitude and also the orientation, that is the direction of the edge normal.

Here you can see, I am determining the gradient magnitude and also I am determining the orientations, that is the direction of the edge normal. So, first step is to convolve the image with Gaussian. The second step is, I have to determine the gradient magnitude and the direction. So, you can see I am determining the gradient magnitude and I can determine the angle. After this, the next important step is the non maximum suppression.

So, that concept already I have explained, the non maximum suppression. And after non maximum suppression, you have to apply the concept of the thresholding with hysteresis. So, for this you can consider the low threshold and the high threshold, and based on this, you can determine the edge pixels. And you can see the finally, the edges are determined, that is the edge pixels are determined with the help of the canny edge detector. So, this is the concept of canny edge detector.

(Refer Slide Time: 24:21)

**Theoretical Understanding**

The basic steps involved in this algorithm are:

- Noise reduction using Gaussian filter ✓
- Gradient calculation along the horizontal and vertical axis
- Non-Maximum suppression of false edges
- Double thresholding for segregating strong and weak edges
- Edge tracking by hysteresis ✓

So, the main steps I have already explained. So, first I have to do the convolution of the image with the Gaussian, that is the noise reduction using a Gaussian filter. After this, I have to determine the gradient magnitude. So, first I have to determine the gradient along the x direction and the gradient along the y direction, the horizontal and the vertical directions. After this I can determine the gradient magnitude.

And also I can determine the, direction of the edge is normal. After this the non maximum suppression, that principle I have to consider. Because in this case, I have to see the neighborhood. So, based on the neighborhood, I can apply the principle of non maximum suppression. The next one is the thresholding with the hysteresis.

So, that means I can consider two thresholds, one is the low threshold another one is the high threshold. And I can determine the strong and the weak edge pixels. So, that is the thresholding with hysteresis. And finally, we can do the edge linking. So, these are the steps of a canny edge detector.

(Refer Slide Time: 25:30)

```
1 clc;
2 clear;
3
4 I = rgb2gray(imread('4.1N.1.jpg')); ✓
5 figure(1)
6 imshow(I);
7 title('Input Image'); ✓
8 I = double(I);
9 msize(I,1);
10 nsize(I,2);
11 Ix=I;
12
13 Iy=I;
14 I_gauss= imgaussfilt(I, 0.5); %Gaussian Blur
15
16 for i=1:n-2
17     Iy(i,:)=I(i,:)-I(i+2,:); %x Gradient ✓
18 end
19 for i=1:m-2
20     Ix(:,i)=I(:,i)-I(:,i+2); %y Gradient ✓
21 end
22
23 angle=imadd(atan2(Ix./Iy), 90);
24 magnitude=sqrt(Ix.^2 + Iy.^2);
25 angle(isnan(angle))=0; ✓
26 magnitude(isnan(magnitude))=0; ✓
27
28 %plot magnitude and angle
29 figure(2)
30 subplot(2,1,1)
31 imshow(uint8(angle)); ✓
32 title('Angle'); ✓
33 subplot(2,1,2)
34 imshow(uint8(magnitude)); ✓
35 title('Magnitude'); ✓
36
37 %Non maximal Suppression
```

So, the corresponding program you can see, I am considering the input image. So, this input image I am converting from RGB to gray, that is RGB image is converted into gray. And this is the input image. After this, I am doing the Gaussian blurring, that is the image is convolve with the Gaussian and we can determine the x gradient and the y gradient, that is the gradient along the x direction and the gradient along the y direction I can determine.

And also I can determine the gradient magnitude and also the angle I can determine. And after this I can plot the gradient magnitude and also the directions, the direction of the edge normals. So, this Eigen plot, so these are the first and the second steps.

(Refer Slide Time: 26:18)

```
38 finalImage = zeros (m, n);
39 for i = 2 : m-1
40     for j = 2 : n-1
41         if (angle(i, j) > 0) && (angle(i, j) < 22.5) || ...
42             (angle(i, j) > 157.5) && (angle(i, j) < 202.5) || ...
43             (angle(i, j) > 337.5) && (angle(i, j) < 360)
44             finalImage(i, j) = magnitude(i, j) == ...
45                 max(magnitude(i, j), magnitude(i, j+1), ...
46                     magnitude(i, j-1));
47         elseif (angle(i, j) > 22.5 && (angle(i, j) < 67.5) ...
48             || (angle(i, j) > 202.5 && (angle(i, j) < 247.5))
49             finalImage(i, j) = (magnitude(i, j) == ...
50                 max(magnitude(i, j), magnitude(i+1, j), ...
51                     magnitude(i-1, j+1)))
52         elseif (angle(i, j) > 67.5 && (angle(i, j) < 112.5) || ...
53             (angle(i, j) > 247.5 && (angle(i, j) < 292.5))
54             finalImage(i, j) = (magnitude(i, j) == ...
55                 max(magnitude(i, j), magnitude(i+1, j), ...
56                     magnitude(i-1, j)))
57         elseif (angle(i, j) > 112.5 && (angle(i, j) < 157.5) ...
58             || (angle(i, j) > 292.5 && (angle(i, j) < 337.5))
59             finalImage(i, j) = (magnitude(i, j) == ...
60                 max(magnitude(i, j), magnitude(i+1, j+1), ...
61                     magnitude(i-1, j-1)));
62         else
63             end;
64         end;
65     end;
66 end;
67 finalImage = finalImage.*magnitude;
68 figure();
69 imshow(finalImage);
70 title('Intermediate Image');
71 imwrite(finalImage, '4.007.2.jpg');
72
73 % Hysteresis Thresholding
74 Threshold_min = 0.075;
75 Threshold_max = 0.175;
76 Threshold_res = zeros (m, n);
77 Threshold_min = Threshold_min + max(max(finalImage));
78 Threshold_max = Threshold_max + max(max(finalImage));
79
80 for i = 1 : m
81     for j = 1 : n
82         if (finalImage(i, j) < Threshold_min)
83             Threshold_res(i, j) = 0;
84         elseif (finalImage(i, j) > Threshold_max)
85             Threshold_res(i, j) = 1;
86         elseif ~connected_components
87             elseif (finalImage(i+1, j) > Threshold_max || ...
88                 finalImage(i-1, j) > Threshold_max || ...
89                 finalImage(i, j+1) > Threshold_max || ...
90                 finalImage(i, j-1) > Threshold_max || ...
91                 finalImage(i-1, j-1) > Threshold_max || ...
92                 finalImage(i+1, j+1) > Threshold_max || ...
```

After this I can consider the non maximum suppression. So, that means, I can consider the quantization of the all possible directions into four or five directions, because I have to search the neighborhood pixels. So, for this I am doing the quantization of all the directions into four directions. So, here you can see, I am considering the directions like this. And based on this I am finding the neighborhood pixels.

After this I have to apply the principle of the non maximum suppression. So, you can see I am applying the non maximum suppression and I am getting the intermediate image. So, this is the intermediate image. After this I am considering the thresholding with hysteresis. So, for this I am considering two thresholds, one is the low threshold, another one is the high threshold. And after this I am applying the thresholding with hysteresis principle, that principle I am applying.

(Refer Slide Time: 27:10)

```
        final_image(i+1, j+1)>Threshold_max || ...
        final_image(i+1, j-1)>Threshold_max)
        Threshold_res(i,j) = 1;
74     end;
75     end;
76     end;
77 end;
78 edges = uint8(Threshold_res.*255);
79 figure(4)
80 imshow(edges);
81 title('Final Image');
82 imwrite(edges, '4.OUT.3.jpg');
```

And corresponding to this, I will be getting the edge pixels, and this edge pixels of the input image you can display, you can show. So, these are, these are the, this is the concept of the edge detection with the help of canny edge detector, that is the MATLAB program.

(Refer Slide Time: 27:30)

```
import numpy as np
import os
import cv2
import matplotlib.pyplot as plt

# defining the canny detector function

# here weak_th and strong_th are thresholds for
# double thresholding step
def Canny_detector(img, weak_th = None, strong_th = None):

    # conversion of image to grayscale
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Noise reduction step
    img = cv2.GaussianBlur(img, (5, 5), 1.4)

    # Calculating the gradients
    gx = cv2.Sobel(np.float32(img), cv2.CV_64F, 1, 0, 3)
    gy = cv2.Sobel(np.float32(img), cv2.CV_64F, 0, 1, 3)

    # Conversion of Cartesian coordinates to polar
    mag, ang = cv2.cartToPolar(gx, gy, angleInDegrees = True)

    # setting the minimum and maximum thresholds
    # for double thresholding
    mag_max = np.max(mag)
```

*Handwritten notes:*  
Pandas  
cv2.imread()

Similarly, I can consider the same thing that is the canny edge detector by considering the Python programming. So, I am just giving one example of Python programming here. So, first, I



am considering import numpy as np. So, what is numpy? Numpy is an open source numerical Python library. So, if you read the Python library, then you will get this one. So, it is an open source numerical Python library. And what is available in the numpy?

So numpy contains a multi dimensional array and matrix data structures, and it can be utilized to perform a number of mathematical operations on areas such as trigonometric operations, statistical operations, algebraic routines, so, all these operations we can perform with the help of this library, that is the numpy library. And one is the pandas, pandas is an open source Python package. And that is most widely used for data science, data analysis or in machine learning also, we used this pandas and it is built on top of the numpy package.

So, this pandas is also another library. And you can see input os, what is os? The os module in Python, provide functions for interacting with the operating system, os comes under python's standard utility module. So, os means, the os module in Python provides function for interacting with the operating systems. So, that is os. After this, next one is input cv2, that is nothing but open cv Python.

So, open cv Python is a library of Python designed to solve computer vision problems. So, suppose if I consider, suppose cv2 dot suppose im read. So, if I consider this statements cv2 im read this one. So, it loads an image from the specified file and suppose if the images cannot be read, because of missing file or unsupported or invalid format, then this method returns an empty matrix.

So, input cv2 means it is an open CV Python. So, it is used for computer vision problems. So after this I am considering import matplotlib, that is a collection of functions that make matplotlib work like MATLAB. So, we can create the figure, we can do the plotting like this with the help of matplotlib. So, this is a library, the import this one. After this I am considering the Canny edge detector in Python.

So, here you can see the conversion of the image to grayscale, again I am converting the image into grayscale. Suppose the RGB image is available, so we can convert the image into grayscale. After this we can do the Gaussian blurring. So, you can see, I am doing Gaussian blurring. After this I am considering the sobel operator, to determine the gradient magnitude. So, we can

determine the gradient magnitude, we can determine the gradient along the x direction and the gradient along the y direction. And from this we can determine the gradient magnitude.

(Refer Slide Time: 31:11)

```
# Looping through every pixel of the grayscale
# image
for i_x in range(width):
    for i_y in range(height):
        grad_ang = ang(i_y, i_x)
        grad_ang = abs(grad_ang-180) if abs(grad_ang)>180 else abs(grad_ang)

        # selecting the neighbours of the target pixel
        # according to the gradient direction
        # In the x axis direction
        if grad_ang<= 22.5:
            neighb_1_x, neighb_1_y = i_x-1, i_y
            neighb_2_x, neighb_2_y = i_x+1, i_y

        # top right (diagonal-1) direction
        elif grad_ang>22.5 and grad_ang<=(22.5 + 45):
            neighb_1_x, neighb_1_y = i_x-1, i_y-1
            neighb_2_x, neighb_2_y = i_x+1, i_y+1

        # In y-axis direction
        elif grad_ang>(22.5 + 45) and grad_ang<=(22.5 + 90):
            neighb_1_x, neighb_1_y = i_x, i_y-1
            neighb_2_x, neighb_2_y = i_x, i_y+1

        # top left (diagonal-2) direction
        elif grad_ang>(22.5 + 90) and grad_ang<=(22.5 + 135):
            neighb_1_x, neighb_1_y = i_x-1, i_y+1
            neighb_2_x, neighb_2_y = i_x+1, i_y-1

        # Now it restarts the cycle
        elif grad_ang>(22.5 + 135) and grad_ang<=(22.5 + 180):
            neighb_1_x, neighb_1_y = i_x-1, i_y
            neighb_2_x, neighb_2_y = i_x+1, i_y

        # Non-maximum suppression step
        if width>neighb_1_x>= 0 and height>neighb_1_y>= 0:
            if mag[i_y, i_x]>mag[neighb_1_y, neighb_1_x]:
                mag[i_y, i_x]= 0
                continue

        if width>neighb_2_x>= 0 and height>neighb_2_y>= 0:
            if mag[i_y, i_x]>mag[neighb_2_y, neighb_2_x]:
                mag[i_y, i_x]= 0

weak_ids = np.zeros_like(img)
strong_ids = np.zeros_like(img)
ids = np.zeros_like(img)
```

After this we are considering, the concept of the non maximum suppression. So, all these angles I am considering, like 22.5 degree plus 90 degree plus 135 degree. So, quantization of all possible directions into four directions by defining the range. So, that means I am defining the range, and I am doing the non maximum suppression. That means I am finding the neighborhood pixels. And based on this, I am doing the comparisons. And from this, I am getting the non maxima suppress image.

(Refer Slide Time: 31:43)

```
if grad_mag < weak_th:
    mag[i_y, i_x] = 0
elif strong_th < grad_mag <= weak_th:
    ids[i_y, i_x] = 1
else:
    ids[i_y, i_x] = 2

# finally returning the magnitude of
# gradients of edges
return mag

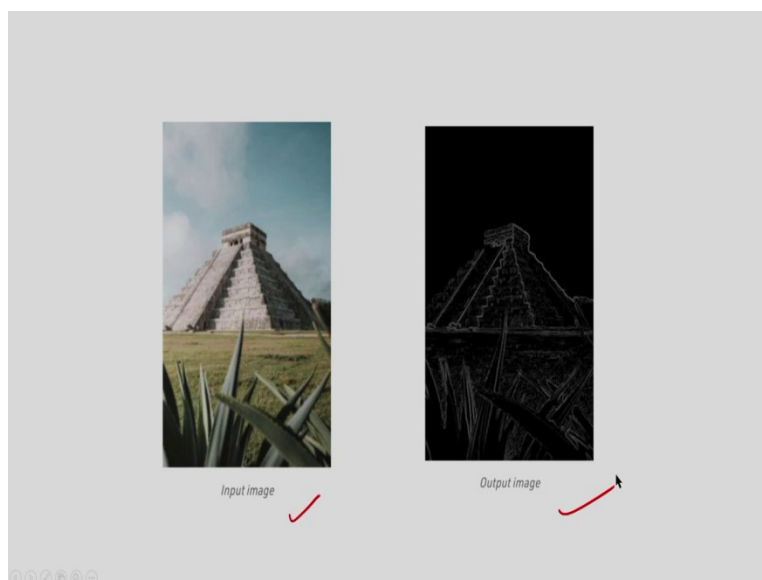
frame = cv2.imread('food.jpeg')

# calling the designed function for
# finding edges
canny_img = Canny_detector(frame)

# Displaying the input and output image
plt.figure()
f, plots = plt.subplots(2, 1)
plots[0].imshow(frame)
plots[1].imshow(canny_img)
```

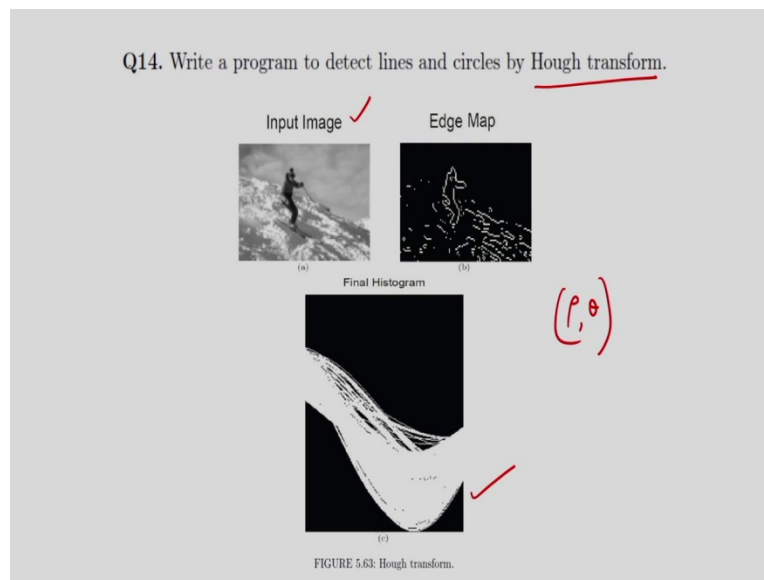
After this, what am I considering? I am considering the thresholding with hysteresis. So, for this, I am considering the strong threshold and the weak threshold. Also I am considering and I will be getting the strong edge pixels and the weak edge pixels. And finally, I can get the edge pixels and I can show the output image and the input images. So, this is the Python programming to show the canny edge detector. So, like this you can write either a MATLAB program or the open cv Python program.

(Refer Slide Time: 32:24)



So, corresponding to this you can see the input image and the output images. So, edges are detected in the output image. That is the edge detection by canny edge detector.

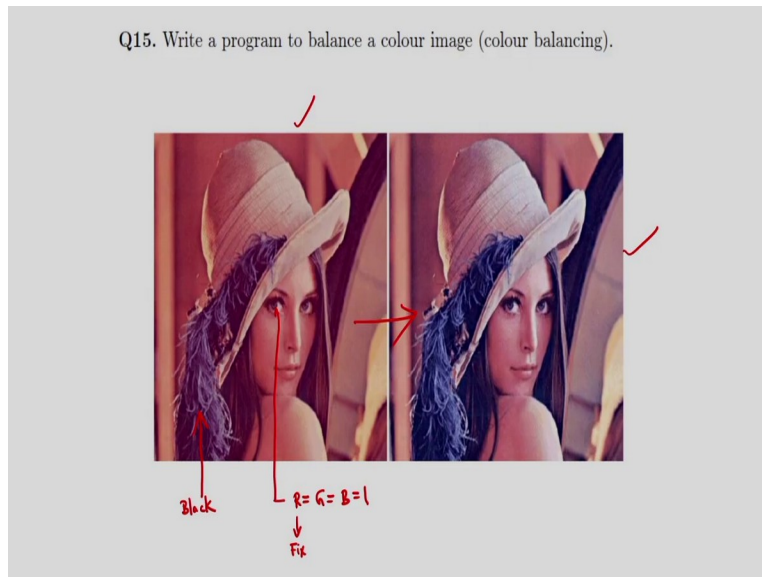
(Refer Slide Time: 32:37)



Next problem is to write a program to detect lines and the circle by Hough transform. So, corresponding to this image, I am determining the lines and the circles by considering the Hough transformation. So, this principle has already I have explained in my class. So, for this we have to consider the parametric space, the parameter space is rho theta is the parameter space. And mainly I have to do the voting.

So, for this I have to first initialize the accumulator and after this I have to go for the voting, corresponding to the pixels, all the edge pixels if I consider. And from this I can determine the or I can detect the lines, I can detect that circles. So, you can see this is the parametric space, I can show the outputs in the parameter space. And based on this booting, I can detect the lines, I can detect the circles. This is question number 14.

(Refer Slide Time: 33:38)



Write a program to balance a color image. So, here you can see, I am considering the input image, that is not color balance, the colors are improper. So, I have to balance the colors. So, the simple procedure is, suppose it corresponds to a known region. Suppose if I consider this is the known region, corresponding to this, this is a, this should be white, this should be white, but in this case I am not getting the white.

So, corresponding to this known white portion, R should be equal to G should be equal to B should be equal to 1, but actually, I am not getting the white color. Similarly, if I consider this is the black. So, this is the black color, but I am not getting the black color because of the imperfection of the image capturing device. So, what can I consider? I have to consider the color balancing principle.

So, corresponding to the known region, suppose white region I am considering R is equal to G is equal to B is equal to 1. Now, suppose I make one component fix, suppose I am making R fix and find a transformation for the other two components that is the G and B. So, that R is equal to G is equal to B is equal to 1. So, I have to make one component fix and find a transformation for other two components. So, that R is equal to G is equal to B is equal to 1.

And apply this transformation for all the pixels of the image. So, apply this transformation for all the pixels of the image. And after this if I apply this transformation, I will be getting the color

balance image, that is the output. So, I am getting the color image, color balance image in the output, that is called the color balancing. So, you can write a MATLAB program or maybe open CV Python program for this problem.

(Refer Slide Time: 35:25)



Next one is to write a program for a vector median filter and show the results for a color image. So, here I am showing you one input image, Lena image. And this image is corrupted by noise, the salt and pepper noise. And after this I can remove the salt and pepper noise by considering the vector median filter. So, I cannot apply the scalar median filter, in case of the color image. So, that concept also I have discussed in the class. So, for removing the salt and pepper noise in the color image I have to apply the vector median filter.

(Refer Slide Time: 36:03)

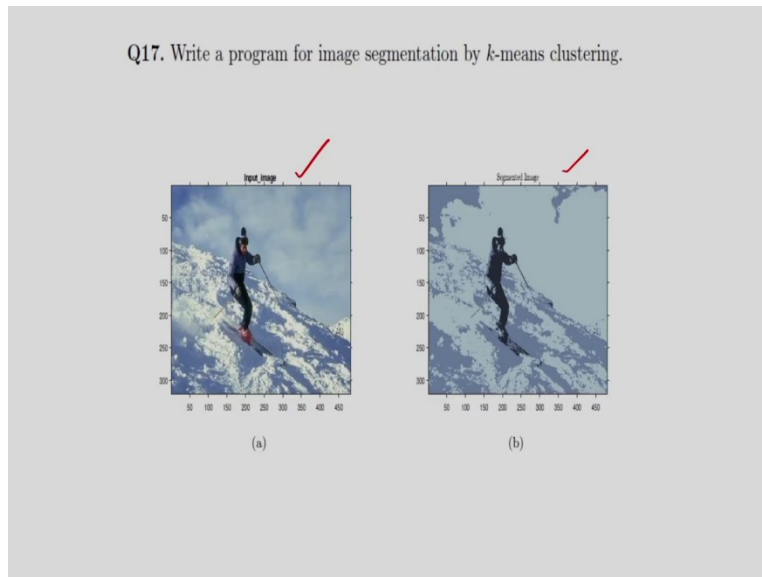
```
1 clear;
2 clear;
3 I = imread('IM1.jpg', 0.5);
4 I = imnoise(I, 'salt & pepper', 0.01);

5 figure(1);
6 imshow(I);
7 title('Original Image with Noise');
8 imshow(I, 'IM1.jpg');
9 m = size(I, 1);
10 n = size(I, 2);
11
12 I = double(I);
13 red = I(:,:,1);
14 green = I(:,:,2);
15 blue = I(:,:,3);
16
17 final_image = zeros(m, n);
18 distance = zeros(3);
19 for i = 2:m-1
20     for j = 2:n-1
21         for ii = -1:1
22             for jj = -1:1
23                 for lfun = -1:1
24                     for rfun = -1:1
25                         # Euclidean = sqrt((red(i-lfun, j-rfun)-lfun)^2 + ...
26                             (blue(i-lfun, j-rfun)-rfun)^2 + ...
27                             (green(i-lfun, j-rfun)-rfun)^2);
28                         tot = tot + euclidean;
29                     end
30                 end
31                 distance(ii+2, jj+2) = tot;
32             end
33         end
34         distance = [distance(1)];
35         [rLow, rCol] = find(min(distance), 1);
36         final_image(i, j, 1) = red(i+rLow, j+rCol);
37         final_image(i, j, 2) = green(i+rLow, j+rCol);
38         final_image(i, j, 3) = blue(i+rLow, j+rCol);
39     end
40 end
41 final_image = uint8(final_image);
42 figure(2);
43 imshow(final_image);
44 title('Final Image');
45 imshow(final_image, 'IM1.jpg');
```

So, what is the program for this? You can see, so first I am considering the original image and after this, I am applying the salt and pepper noise and we can display the images, the original image with noise. After this mainly I have to determine the distances. So, I have to consider one window and in this window I have to consider the distances. So, distances in terms of RGB value. So, that is nothing but the Euclidean distance I have to determine, the distances in terms of RGB value.

So, you can see the distances in terms of the RGB value  $r - r$ ,  $b - b$  like this  $g - g$ . So, I am finding the distance in terms of RGB value and I have to find a minimum distance. So, this is the minimum distance I have to find, and based on this I can find the median value. So, median value I can determine and after this, I can display the output image that is the salt and pepper noise can be removed. So, this is the program for the vector median filter.

(Refer Slide Time: 37:05)



And the question number seventeen is to write a program for image segmentation by  $k$  means clustering. So, the input image I am considering, and you can see I am doing the image segmentation by considering the  $K$  means clustering, first I have to consider the means. So, randomly I have to select the means, after this I have to update the means, based on the minimum distance of the algorithm, the algorithm is a  $k$ - means clustering.

(Refer Slide Time: 37:34)

```
1 clear
2 clc
3 I = double(imread('0201.jpg'))/255;

4 figure(1)
5 imshow(I);
6 title('Input Image');
7
8 J = reshape(I,size(I,1)+size(I,2),3);
9
10 %Initialization
11 mean1 = [120 120 120] / 255;
12 mean2 = [12 12 12] / 255;
13 mean3 = [180 180 180] / 255;
14 centroid = [mean1; mean2; mean3];
15 labels = zeros(size(J,1), 4);
16 n_iter = 50;
17
18 %kmeans
19 for n = 1: n_iter
20     for i = 1:size(J,1)
21         for j = 1:3
22             labels(i,j) = norm(J(i,:) - centroid(j,:));
23         end
24         [Distance, cluster_label] = min(labels(i,1:3));
25         labels(i, 4) = cluster_label;
26     end
27     for i = 1:3
28         h = labels(i,4) == i;
29         centroid(i,:) = mean(J(h,:)); % New Cluster Centers
30     end
31 end
```



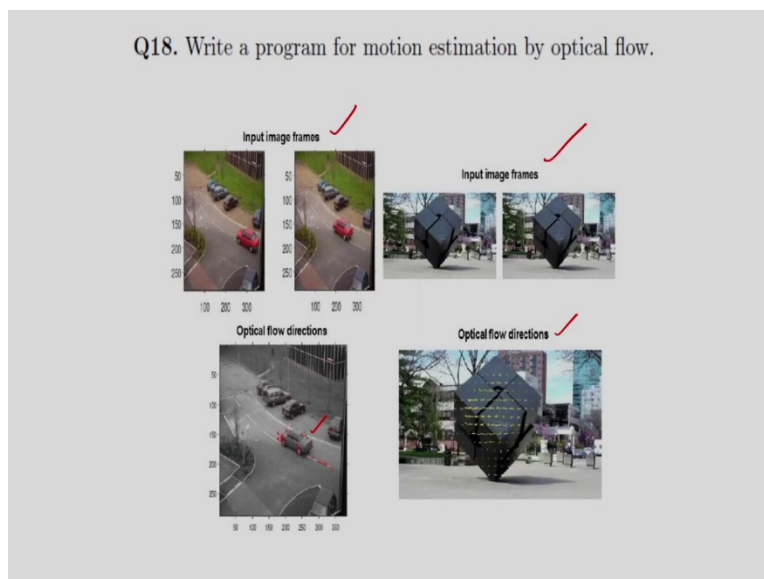
So, you can see the program. So, first I am initializing the means, mean 1, mean 2, mean 3, 3 means I am considering. After this, I have to find the distance between the sample points and the means, and after this I have to update the clusters. So, I will be getting the new cluster centers.

(Refer Slide Time: 37:51)

```
31 end
32
33 X = zeros(size(J));
34 for i = 1:3
35     idx = find(labels(:,4) == i);
36     X(idx,:) = repmat(centroid(i,:),size(idx,1),1);
37 end
38 Final_image = reshape(X,size(I,1),size(I,2),3);
39
40
41 figure(2)
42 imshow(Final_image);
43 title('Segmented Image');
44 imwrite(Final_image, '8_OUT.1.jpg');
```

This process I have to repeat again and again until there is no change of the means. And after this finally, I will be getting the segmented output image. So, that is about the K means clustering.

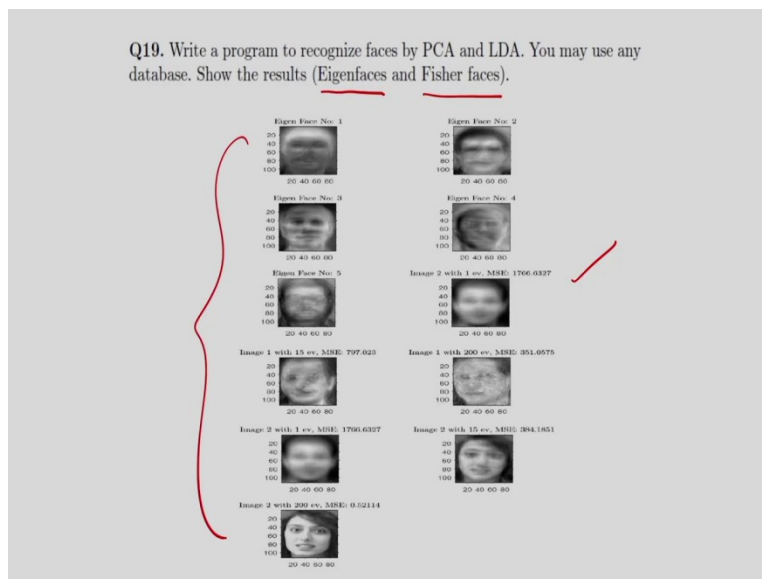
(Refer Slide Time: 38:05)



The problem number eighteen, eighteen is to write a program for motion estimation by optical flow. So, here I am showing two examples, one is the input image. I am considering the frames of the input video, here I am showing two frames only. And similarly, for the second case also I'm showing the input image frames corresponding to a video, particular video. And after this I can apply the optical flow principle.

So, I can determine the optical flow you can see, the direction of the optical flow. And similarly, you can see the optical flow directions I can determine from the input image frames. So, you have to apply the principle of optical flow. So, this also you can do in MATLAB or maybe in open CV Python. So, this is a very interesting program.

(Refer Slide Time: 38:51)



Next is question number nineteen, write a program to recognize features by PCA and the LDA. So, any database I can use. And in this case I have to first determine the Eigenfaces for the PCA. And for the LDA I have to determine the fisher faces. So, here you can see I am showing the Eigen faces. So, the principle is, any unknown face can be represented by a linear combination of Eigen faces.

So, these are the Eigen faces I can determine. These Eigen faces I can determine from the Eigenvectors of the covariance matrix. That means, from the input vector I can determine a covariance matrix, from the covariance matrix I can determine the Eigenvectors. And that can be

displayed as an image, that is nothing but the Eigen faces. So, any face can be represented by a linear combination of the Eigen faces.

(Refer Slide Time: 39:42)



So, these are the Eigen faces.

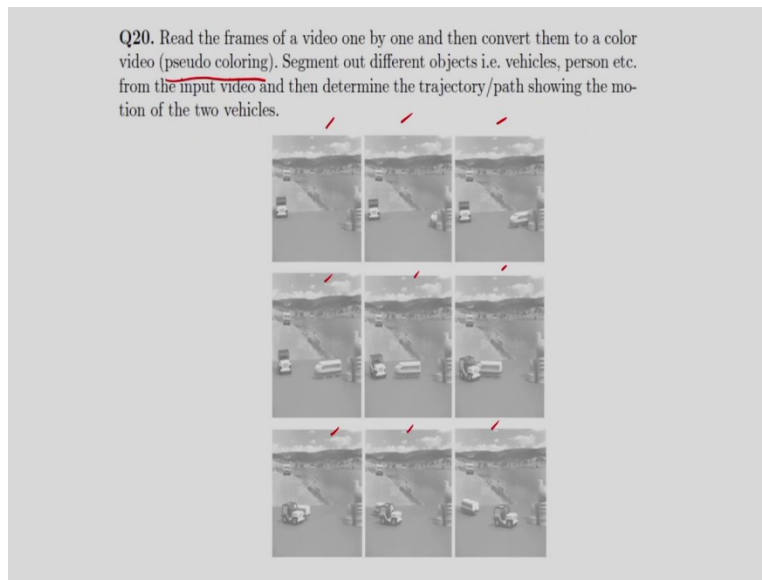
(Refer Slide Time: 39:44)



And similarly, I can also determine LDA faces. So, I have to apply the LDA principle and I can determine the fisher faces. So, that means I have to determine the fisher faces from this matrix

that is the, within scatter matrix inverse into SB, that is the between scatter matrix. So, from this you can determine the fisher faces. So, these are the fisher faces.

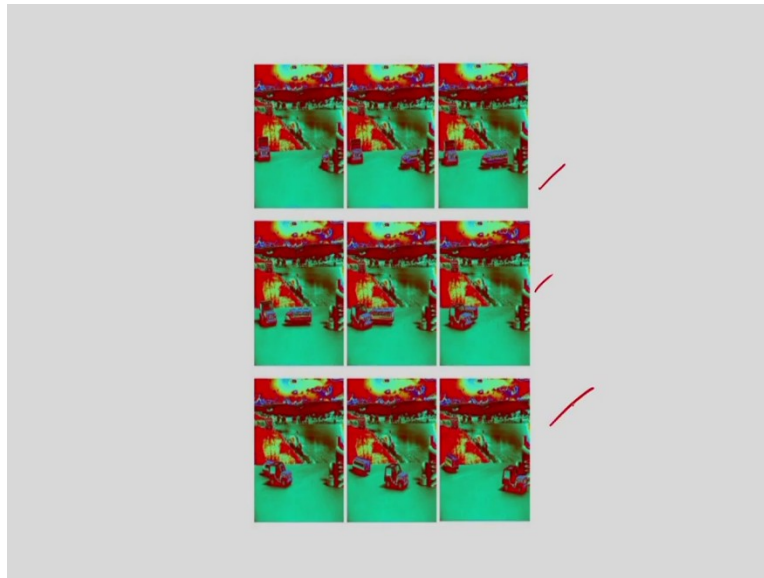
(Refer Slide Time: 40:12)



The last problem is, read the frames of a video one by one and then convert them to a color video. That is nothing but the pseudo coloring, that is the false coloring. Segment out different objects, vehicles, persons etc from the input video and then determine the trajectory path showing the motion of the two vehicles. So, in this case, in this input, you can see I am considering the frames of a video. So, these are the frames of a video and I have to determine the moving objects.

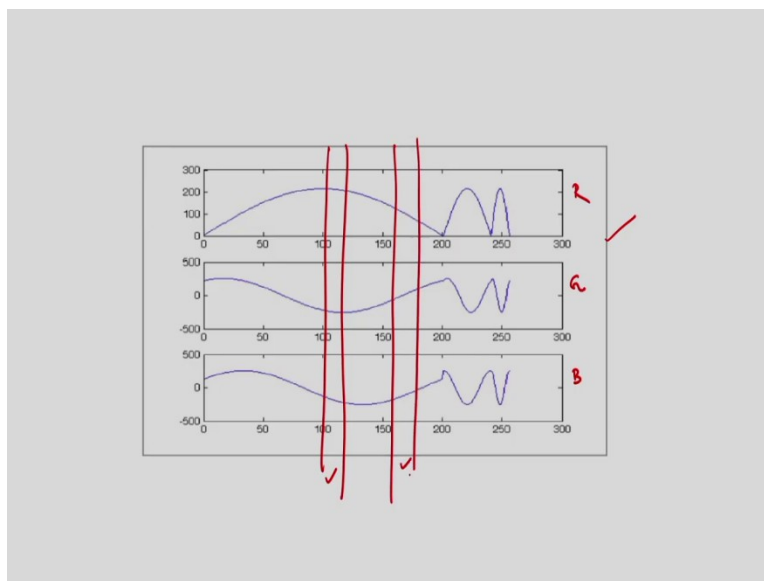
So, simply you can apply the sense detection algorithm, for determining the moving objects or maybe you can apply the optical flow algorithm or any other algorithms you can apply. But the simple one is the sense detection algorithm, to determine the moving objects. And for the pseudo coloring, the pseudo coloring concept can be used to convert the black and white image, into color image, and the grayscale image into color image.

(Refer Slide Time: 41:13)



So, I am applying the pseudo coloring principle and you can see the outputs. So, I am converting the grayscale image into color images.

(Refer Slide Time: 41:21)

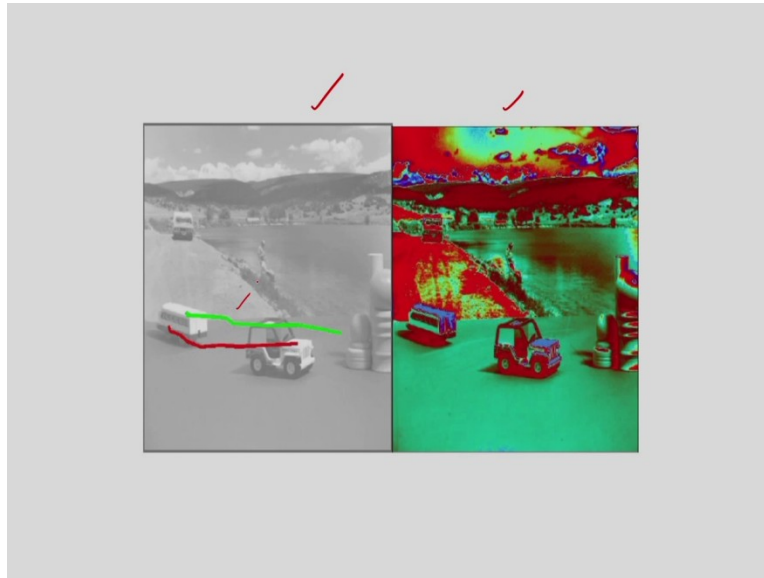


So, this transformation I can apply for pseudo coloring. So, the first transformation is part of the r component. Suppose these are transformations for the g component, and this is the transformation for the blue components. And you can see if I see the transformation, there is a

difference in the face and the frequency. And for this I am getting the false colors and corresponding to each portion I will be getting different, different colors.

So, corresponding to this portion I will be getting one color, corresponding to this portion I will be getting another color. So, this is the concept of pseudo coloring.

(Refer Slide Time: 41:54)



And you can see, corresponding to this input image, I am converting the input image into the color image, And also by applying the sense detection algorithm, I can determine the moving objects and corresponding to these I can determine that trajectory. So, you can even apply the optical flow algorithm. In this class I have given some examples of programming, I have given twenty examples, the programming examples.

I feel you should do programming to understand the concept of image processing, the concept of computer vision and also the machine learning algorithms. If you do programming in MATLAB or any programming languages, your concept will be clear. Initially I thought that I can cover many topics, particularly the applications of computer vision, but in a thirty hours course, it is not possible to cover all the applications of computer vision.

So, that is why I have to stop here because I have already covered forty hours of lectures. So maybe next time I can discuss more applications of computer vision. So, I hope you have enjoyed the course. Thank you.