# CVIP- Project 3

## 1 MORPHOLOGY IMAGE PROCESSING

IN THIS TASK WE USE MORPHOLOGICAL ALGORITHMS SUCH AS EROSION, DILATION OR THE COMBINATION OF THE BOTH TO ACHIEVE THE TASKS ON THE GIVEN IMAGE.

### 1.1 MORPHOLOGY ALGORITHM TO REMOVE NOISE

I have used two morphology algorithms as follows:
Algorithm 1: Opening followed by Closing. The resultant image is as shown below.

- Erosion: A kernel or a window (for Eg: 3x3) is considered or made to slide through the image. A pixel in the original image is considered 1 only if all the pixels under the kernel Is 1, otherwise it is eroded (made to 0). It removes the white noises in an image.
- Dilation: It is opposite to erosion, here the pixel element is 1 if at least one element under the kernel is '1'. We do this after erosion as erosion shrinks the image, dilation increases the object size.
- Opening: It includes Erosion followed by Dilation.
- Closing: It includes Dilation followed by Erosion.

Res_noise1.jpg

Algorithm 2: Closing followed by Opening.

- Erosion: A kernel or a window (for Eg: 3x3) is considered or made to slide through the image. A pixel in the original image is considered 1 only if all the pixels under the kernel Is 1, otherwise it is eroded (made to 0). It removes the white noises in an image.
- Dilation: It is opposite to erosion, here the pixel element is 1 if at least one element under the kernel is '1'. We do this after erosion as erosion shrinks the image, dilation increases the object size.
- Opening: It includes Erosion followed by Dilation.
- Closing: It includes Dilation followed by Erosion.

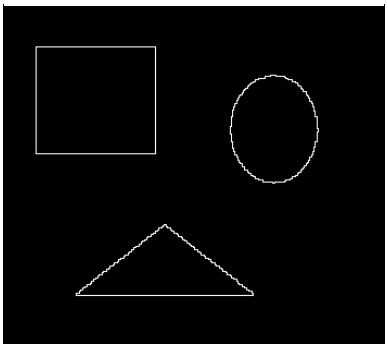
Res_noise2.jpg

## 1.2 COMPARE THE TWO RESULTS

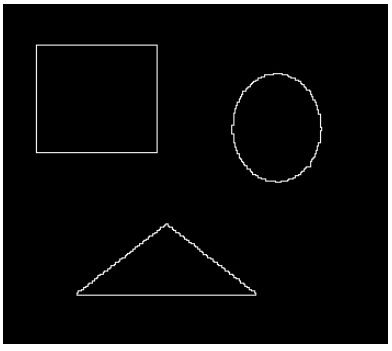
The two results res_noise1.jpg and res_noise2.jpg is compared. The result is 'FALSE'. That is, they **don't match**, as the two images follow two different algorithms to remove the noise.

## 1.3 BOUNDARY EXTRACTION

For the resultant images namely 'res_noise1.jpg' and 'res_noise2.jpg' we extract the boundaries by applying Erosion operation on them and them subtracting it from the original image. The results are as shown below.



Res_bound1.jpg



Res_bound2.jpg

# 2 POINT DETECTION AND IMAGE SEGMENTATION

## 2.1 POINT DETECTION

We have to detect the porosity on the given X-ray image. For this I have used a positive Laplacian kernel as given below. We mask the kernel over the image with a constant threshold set by us ( T = 310). Laplacian kernel (Z) =

| | | |
|---|---|---|
| 0 | 1 | 0 |
| 1 | -4 | 1 |
| 0 | 1 | 0 |

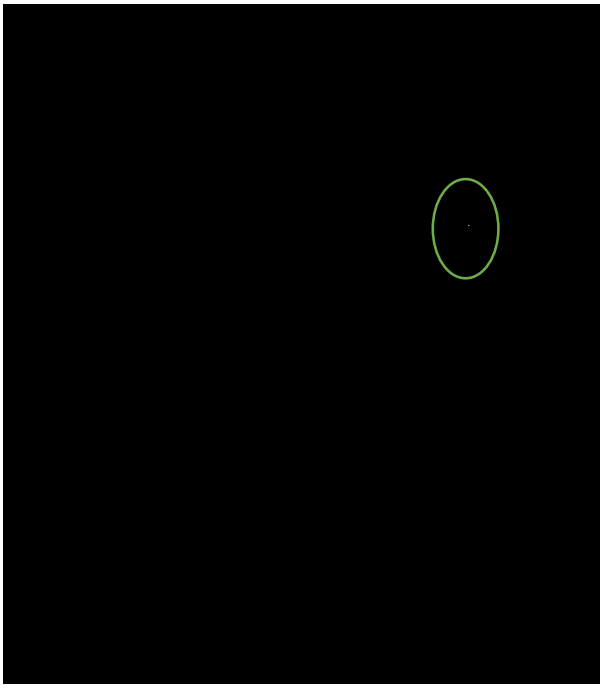
The below formulation measures the weighted difference between the center point and its neighbors. R is the sum of products of the coefficients with the gray levels contained in the region encompassed by the mark.

$$R = \sum_{i=1}^{9} w_i z_i$$

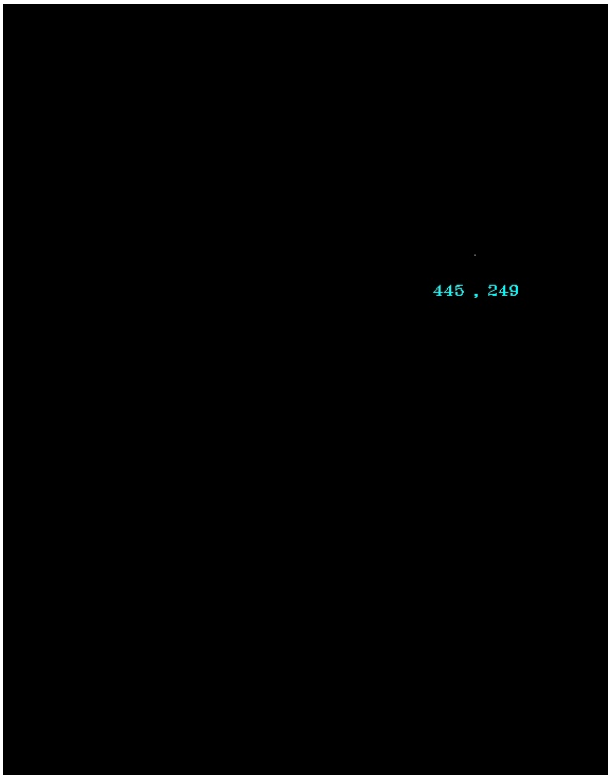
A point has been detected at the location on which the mask is centered if |R| > T.

The below image shows the point detected from the given image and the coordinates for that point.

The point detected were of coordinate **(445,249)**.



Point Detected Image

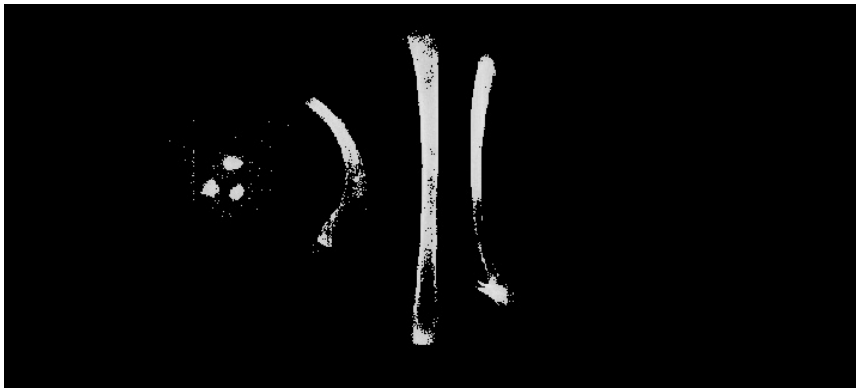



Point Labelled Image

## 2.2 Segmentation of the Image

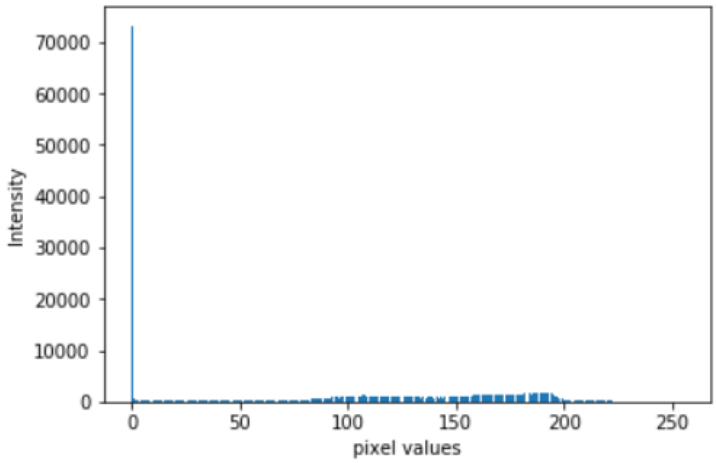
Segmentation is to sub-divide the image into its constituent regions or objects. Segmentation usually is to find the object of interest in an image. In the given image, we have to segment the bones from the background.

I have used normal thresholding method to segment the image. By looking at the histogram plot, I have set a threshold which segments the image into foreground (setting them to a high value) and background (setting them to 0). I have done this on an empty image. The threshold is kept at 202. The image is as shown below.



Segmented Image

An intensity plot of the image is shown below.




A bounding box is drawn on the segmented image, by finding the max and min along x and y direction with the coordinates - *(138 ,22), (425,22), (425, 284),(138 ,284)*. The image is shown below.

Segmented image with the bounding box

# 3 HOUGH TRANSFORM

Hough transform is a technique to detect regular lines, circles and ellipses in an image. A line can be represented as y=mx+c or in parametric form, as ρ=xcosθ+ysinθ where ρ is the perpendicular distance from origin to the line, and θ is the angle formed by this perpendicular line and horizontal axis measured in counter-clockwise. All the points in the image should fall into the same point in the parametric space.

$$Y=MX+C$$

$$rho = x*cos(theta) + y*sin(theta)$$

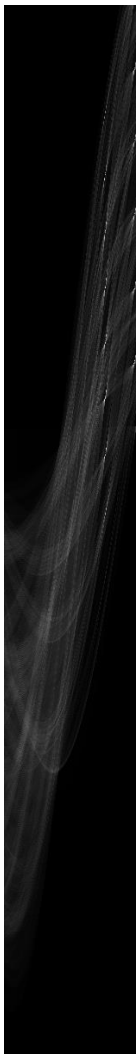
## 3.1 RED LINES DETECTION

The problem is to detect the red lines in an image, that is the vertical lines. If line is passing below the origin, it will have a positive rho and angle less than 180. If it is going above the origin, instead of taking angle greater than 180, angle is taken less than 180, and rho is taken negative. Any vertical line will have 0 degrees.

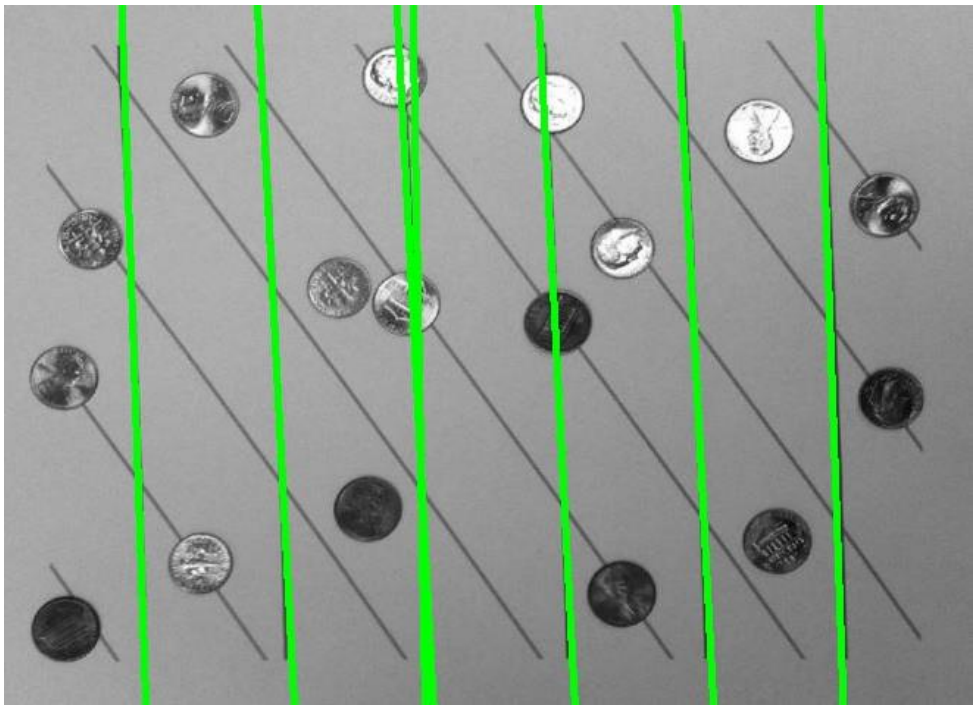
- The image edges are detected with Sobel edge detector.
- The edge image is passed to accumulate (ρ, θ) in an 2D array, initially it is set to zero. The function used in the code is 'hough_lines'.
- As the algorithm runs, each Equation ($X_i$, Yi) is transformed into a discretized (ρ, θ) curve and the accumulator cells which lie along this curve are incremented.
- The accumulated array is passed to the function 'hough_peaks' to find the peaks values in this space which represents the potential line in the image. Resulting peaks in the

accumulator array represent strong evidence that a corresponding straight line exists in the image. The number of peaks is kept at 18, with threshold at 150 and nhood size at 20.

- (Houghl_lines_draw) function finds the endpoints of the line segments corresponding to peaks in the Hough transform and it draws them on the image. I.e. the perpendicular dawn from the origin on to the line.
- For vertical lines, we get 6 peaks which will be our vertical lines, this lines are segregated when drawing the lines on the image using the theta value. Or it can also be done by manually checking the coordinates where the coordinate of pt1 is lesser than a certain value.
- The number of lines detected is all the 6.



Peaks – sine wave image



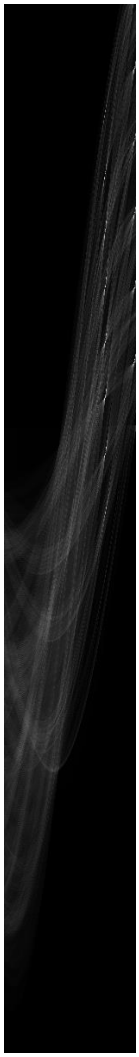
Red_lines.jpg

## 3.2 BLUE LINES DETECTION

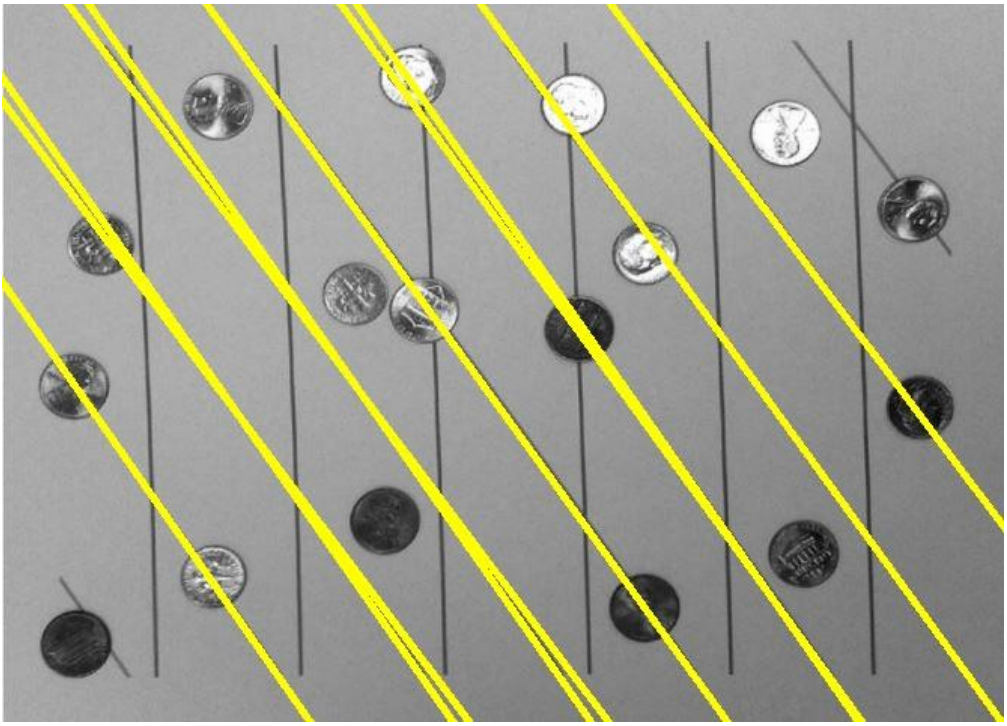
The problem is to detect the blue lines in the image, that is the diagonal lines. The theta is divided by 180.0 to get the diagonal lines in the image

- The image edges are detected with Sobel edge detector.
- The edge image is passed to accumulate (ρ, θ) in an 2D array, initially it is set to zero. The function used in the code is 'hough_lines'.
- As the algorithm runs, each Equation ($X_i$, Yi) is transformed into a discretized (ρ, θ) curve and the accumulator cells which lie along this curve are incremented.
- The accumulated array is passed to the 'hough_peaks' to find the peaks values(local_maxima) in this space which represents the potential line in the image. Resulting peaks in the accumulator array represent strong evidence that a corresponding straight line exists in the image. I,e the perpendicular dawn from the origin on to the line.
- (Houghl_lines_draw) function finds the endpoints of the line segments corresponding to peaks in the Hough transform and it draws them on the image.
- For diagonal lines, the theta is divided by 180.0 or theta appears to be between certain angle. It can also be done by manually checking the coordinates where the coordinate of pt1 is greater than a certain value.

- The number of lines detected is the **7 out of 9.**



Peaks – sine wave image

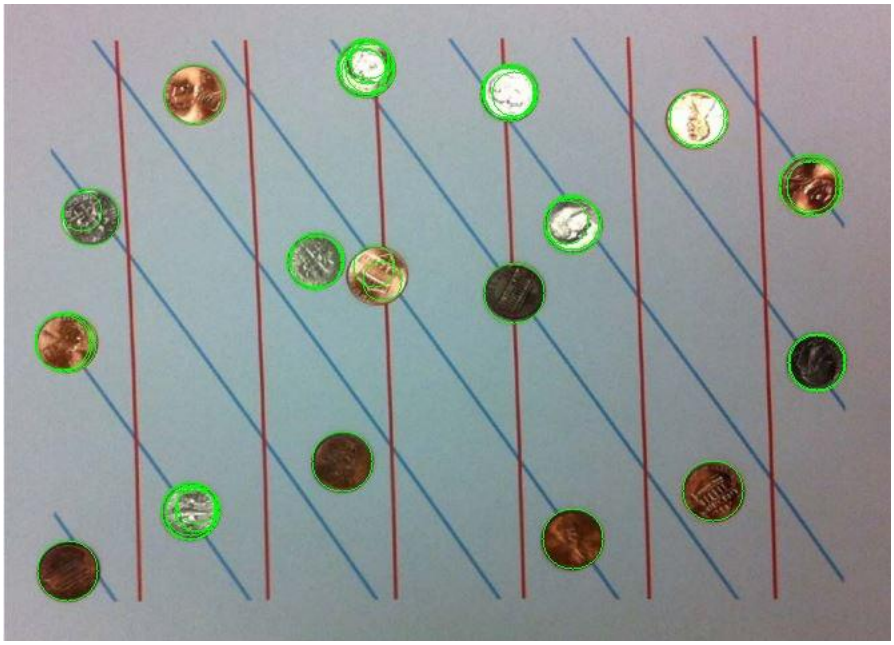


Blue_lines detected

## 3.3 COIN DETECTION

This task is to detect coins in the image, which is nothing but to detect circles with Hough transform. It is analogous to Hough line transform but for circle we need 3 parameters namely (rhos, theta, radius). With the radius fixed, we need to find only two parameters which represents the center of the circle.

- Smoothen the image with Gaussian kernel.
- Find the edge image with Sobel edge detector.
- With the 'Hough circles' function,
    - we initialize the angles 0 to 360.
    - Set the range of radius from 18 to 25
    - Get the rhos and thetas and store it in an accumulator.
    - Use the accumulator we detect the circles with a radius r, where we also check for a threshold.
    - We average the acc_cell, i.e. we find the local maxima, which gives us the center of the image
    - A circle is drawn around the centers found.
- All the 16 coins were detected except 1 which had two small circles inside it.



Coin detected image

## 4.REFERENCES


1) http://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm
2) https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html
3) http://matlab.izmiran.ru/help/toolbox/images/enhanc11.html
4) github.com/SunilVasu/Circle-Houg-Transform
5) https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_circle/hough_circle.html
6) https://en.wikipedia.org/wiki/Circle_Hough_Transform