CSE 573 Project 3

* NIKHIL SRIHARI

50291966

nikhilsr

1. **MORPHOLOGY IMAGE PROCESSING:**

**1B:**

In the first case, we first open and then close. And in the second we first close and then open. Both remove noises in the same way. There is not much of a difference there. The difference comes however, when we look closely at the edges of the 2 images.

The basic effect of an opening is somewhat like erosion, in that it tends to remove some of the foreground (bright) pixels from the edges of regions of foreground pixels.

Closing is similar in some ways to dilation, in that it tends to enlarge the boundaries of foreground (bright) regions in an image (and shrink background color holes in such regions).

In first image, opening happens first eroding the edges and smoothen it out. Then we close to close up gaps in the gaps. This gives comparatively smoother edges, with very less gaps.

In the second image, closing happens first, filling up the gaps first and thus making the edges straighter due to filling. Then we open, eroding the edges a bit.

But still, the overall effect is that in the 2nd images, edges are straighter and more pronounced. Also note that, since rectangle has straight line edges with no protrusions, it remains the same using both methods.

**Output Images:**

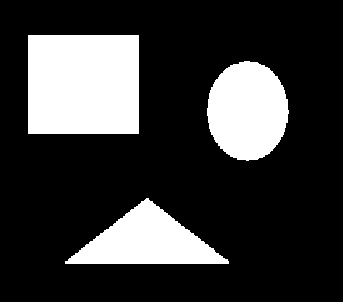


Fig 1A.1. red\_noise1.jpg (Open -> Close)

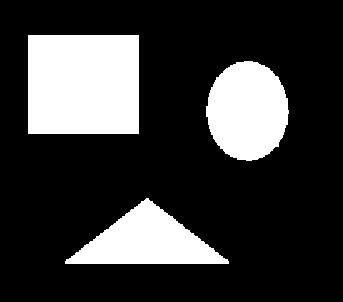


Fig 1A.2. red\_noise2.jpg (Close -> Open)

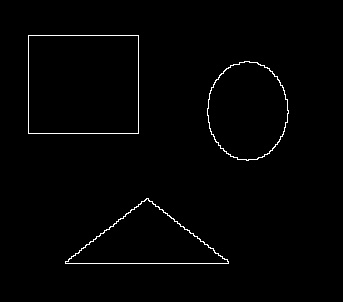


Fig 1C.1. res\_bound1.jpg

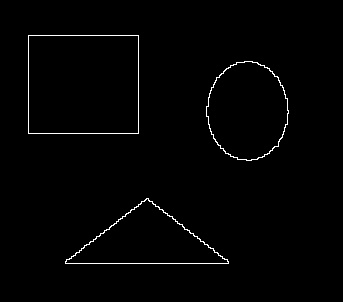


Fig 1C.2. res\_bound2.jpg

**2. IMAGE SEGMENTATION AND POINT DETECTION:**

**Code Output:**

**Comment on Output Image:**

For 2A, first image uses image given on Piazza, whereas the 2nd uses a High Quality image found online. 3rd one uses HQ image after sharpening. Through out these images, some points on the sides are also being identified due to the image nature.

**Output Images:**

*2 A, Method 3:*

*Number of points = 10*

*The points detected are at : [(4, 477), (4, 478), (4, 479), (4, 480), (4, 481), (4, 482), (4, 483), (4, 484), (4, 485), (254, 447)]*

*2 B:*

*[(138, 22), (425, 22), (425, 284), (138, 284)]*

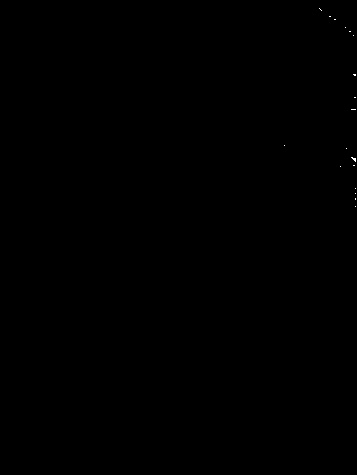


Fig 2A.1. Task2A\_1.jpg

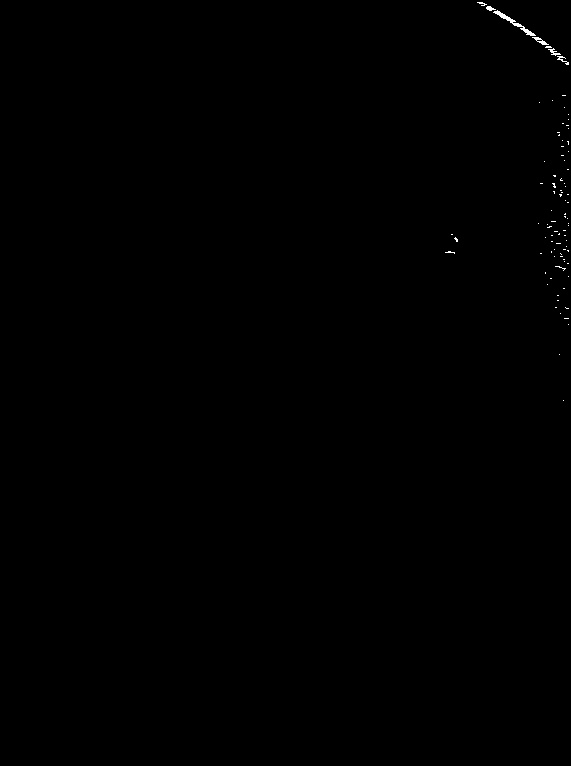


Fig 2A.2. Task2A\_2.jpg

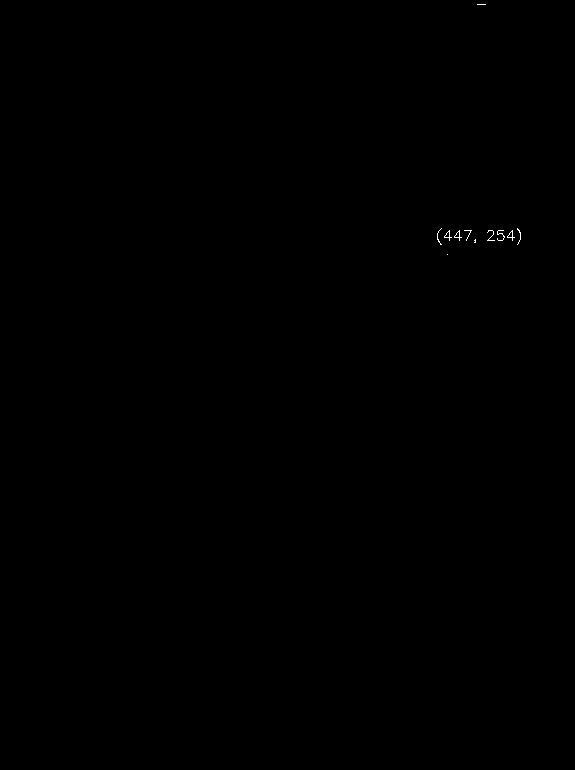


Fig 2A.3. Task2A\_3.jpg

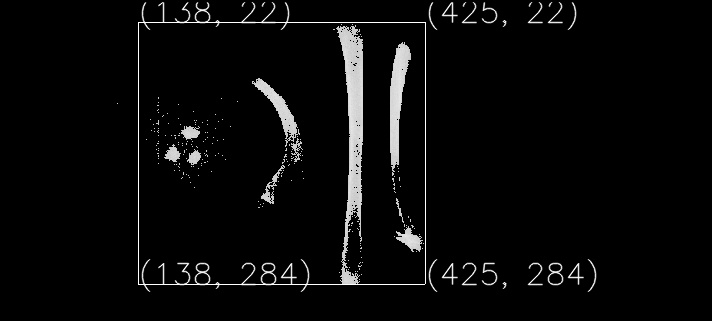


Fig 2B.1. Task2B\_Final.jpg

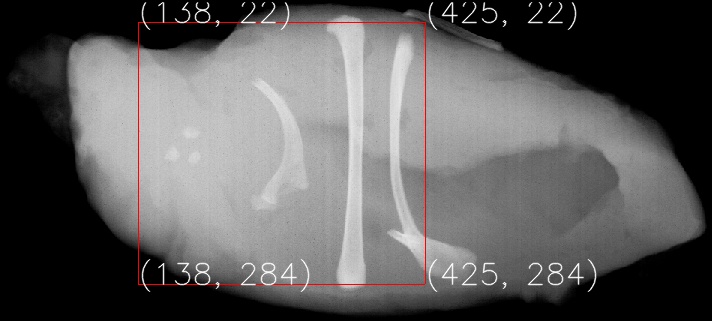


Fig 2B.2. Task2B\_FinalColor.jpg

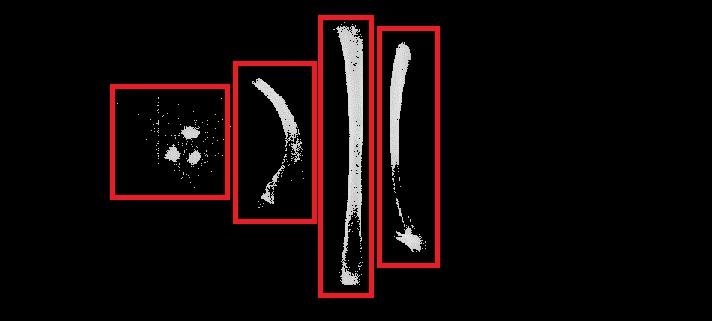


Fig 2B.3. Task2B\_Final\_withIndiBoxes.jpg

**3. Hough Transform:**

**Explanation:**

Every line in Cartesian plane can be expressed as y = mx + c. Here m and c are unique. To each such line, a perpendicular can be dropped from the origin intersecting every line at a unique point (x0,y0). This point can also be represented as (p,o) in Polar form. Thus, any line in Cartesian line can be identified by a unique point(p,o).

Now, let us take these 2 spaces – x Vs y and p Vs o. A point in x Vs y plane can give rise to infinite number of lines in this plane. These can be expressed as a sine wave in the p Vs o plane(also called Hough’s plane). The point where most of these sine waves intersect are the (p,o) values which denote lines in x Vs y plane.

Thus to detect lines, these are the steps to be taken:

1. Get the edge detected image I from the original image.
2. Create a zero array called AccMatrix of size (2\*diagonal +1, 181)
3. For each point P(x,y) in I above a certain threshold:
   1. For every o value from -90 to +90, calculate the corresponding p value using the equation, p = x coso + y sino.
   2. To the point (p,o)(after the necessary adjustment) in AccMatrix, increment its current count.
4. At the end of the above process, we get an AccMatrix filled with overlapping sine waves.
5. The points in the matrix with highest count correspond to the lines’ unique (p,o) value. An additional constraint is placed on these points based on the orientation of lines that we want to detect – Vertical, o = -5 to 5. Slant, o = -36 to -38.
6. Now that we have (p,o), we can get 2 points (x1,y1) and (x2,y2) in Cartesian plane to draw the line, by taking some value of y and then by using equation x = (p – y sino) / coso .
7. Once we have 2 points, line can be drawn by using cv2.line function.

**Code Output:**

*VERTICAL :*

*Number of lines detected : 26*

*Line (p,o) values : [(664, 0), (663, 0), (584, 3), (581, 2), (580, 2), (579, 2), (578, 2), (489, 3), (488, 3), (486, 2), (485, 2), (484, 2), (483, 2), (390, 2), (296, 2), (295, 2), (294, 2), (203, 3), (199, 2), (198, 2), (107, 3), (106, 3), (103, 2), (102, 2), (2, 0), (1, 0)]*

*SLANT :*

*Number of lines detected : 48*

*Line (p,o) values : [(690, 37), (612, 37), (611, 37), (610, 37), (609, 37), (536, 37), (535, 37), (534, 37), (533, 37), (532, 37), (531, 37), (530, 37), (466, 37), (465, 37), (464, 37), (463, 37), (462, 37), (461, 37), (460, 37), (459, 37), (458, 37), (396, 37), (395, 37), (394, 37), (393, 37), (392, 37), (391, 37), (390, 37), (389, 37), (388, 37), (324, 37), (323, 37), (322, 37), (321, 37), (320, 37), (319, 37), (318, 37), (317, 37), (316, 37), (250, 37), (249, 37), (248, 37), (247, 37), (246, 37), (245, 37), (244, 37), (243, 37), (168, 37)]*

*[Finished in 343.5s]*

**Comments on Output:**

As can be seen above, huge numbers of lines have been detected. The reason behind this is that many of these points (p,o) actually correspond to the same line. As can be seen from the edge detected image, every line has some thickness. Hence, this thickness causes multiple points to be detected in the same line. It, however, becomes apparent from the final resultant image that there are 6 vertical and 8 slant lines.

**Output Images:**

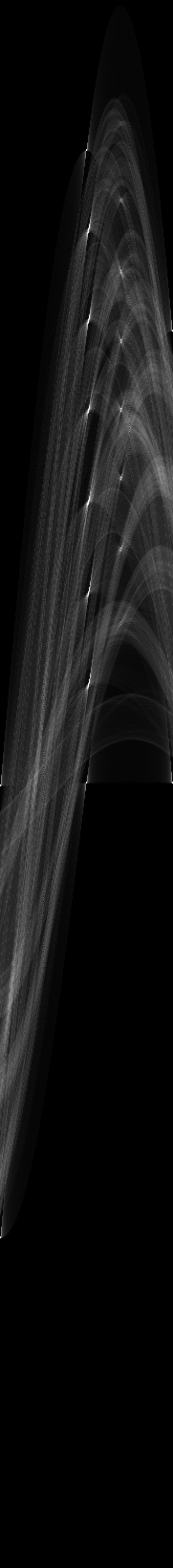


Fig 3.1. Vertical Lines Sine wave

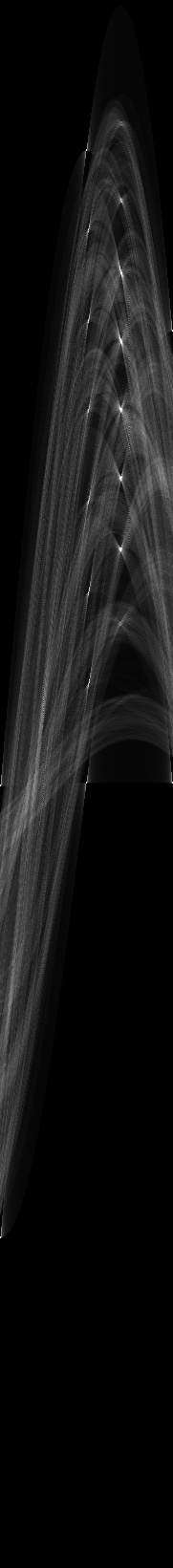


Fig 3.2. Slant Lines Sine wave



Fig 3.3. Slant Lines

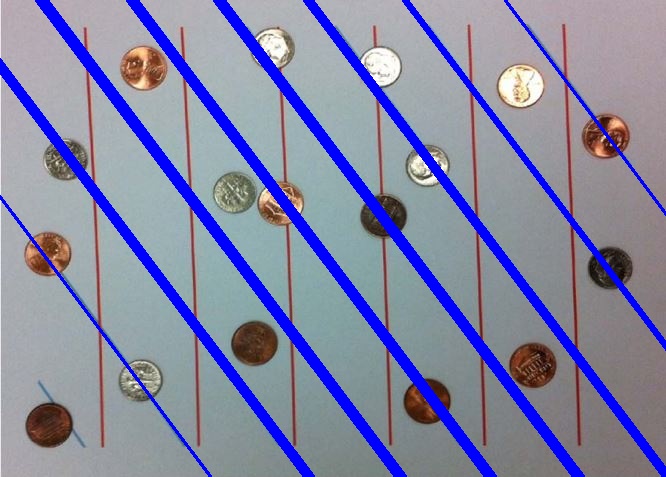


Fig 3.4. Slant Lines



Fig 3.5. Vertical Lines

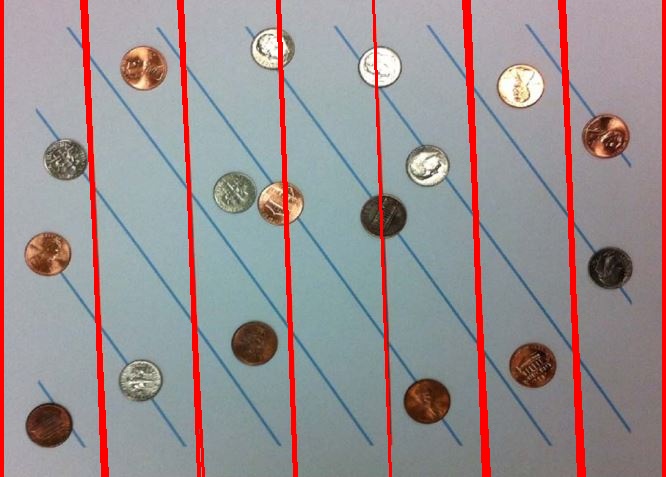


Fig 3.6. Vertical Lines

**3 BONUS:**

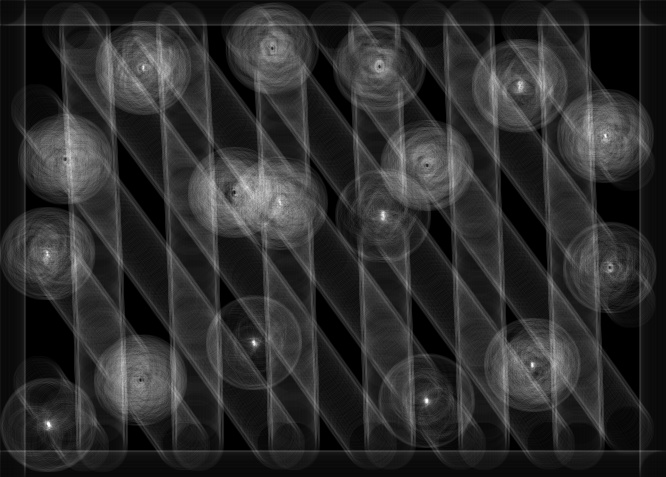
**Explanation:**

Every circle in Cartesian plane can be expressed with 3 pieces of information – center (a,b) and radius r, due to the line equations x = a + r coso and y = b + r sino. Thus if we assume the radius to be a constant, we need only a and b to uniquely identify a circle. This is exactly what we do. We try to denote every circle by a unique point in a Vs b space for a fixed r. This a Vs b is now our Hough’s plane. So each circle in Cartesian plane is a point in Hugh’s plane. And each point (x1, y1), in Cartesian plane, spans a circle in Hough’s plane as a = x1 – r coso and b = y1 – r sino.

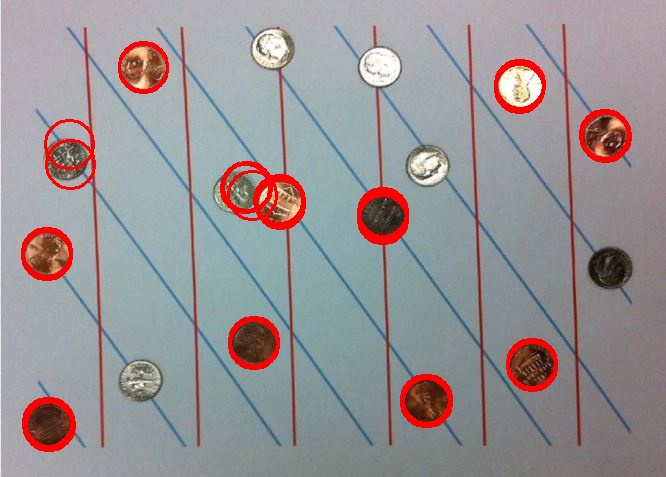
Thus to detect circles, these are the steps to be taken:

1. Get the edge detected image I from the original image.
2. Create a zero array called AccMatrix of size I.
3. Assume some radius value r.
4. For each point P(x,y) in I above a certain threshold:
   1. For every o value from 0 to 360, calculate the corresponding a and b value using the equations, a = x1 – r coso and b = y1 – r sino.
   2. To the point (a,b)(after the necessary adjustment) in AccMatrix, increment its current count.
5. At the end of the above process, we get an AccMatrix filled with overlapping circles.
6. The points in the matrix with highest count correspond to the circles’ unique (a,b) value for this radius r.
7. Now that we have center (a,b) and radius r, we can draw the circle in Cartesian plane, i.e., our image, using cv2.circle function.

**Output Images:**

****

**Fig 3C.1 Hough’s Plane**

****

**Fig 3C. 2 Final OP**

**Output:**

Number of coins detected : 91

circleCenters (a,b) values : [(143, 66), (143, 67), (143, 68), (519, 84), (520, 84), (519, 85), (520, 85), (519, 86), (519, 87), (519, 88), (604, 134), (605, 134), (604, 135), (605, 135), (605, 136), (606, 136), (605, 137), (69, 144), (70, 164), (246, 186), (247, 186), (246, 187), (247, 187), (245, 188), (252, 195), (279, 199), (278, 200), (279, 201), (280, 201), (279, 202), (280, 202), (280, 203), (280, 204), (382, 212), (383, 212), (382, 213), (383, 213), (382, 214), (383, 214), (382, 215), (383, 215), (383, 216), (383, 217), (382, 218), (382, 219), (383, 219), (46, 252), (47, 252), (47, 254), (46, 255), (47, 255), (253, 341), (254, 341), (253, 342), (254, 342), (254, 343), (253, 344), (254, 344), (531, 363), (532, 363), (531, 364), (532, 364), (531, 365), (532, 365), (532, 366), (425, 399), (426, 399), (427, 399), (425, 400), (426, 400), (427, 400), (425, 401), (426, 401), (427, 401), (425, 402), (426, 402), (427, 402), (47, 422), (48, 422), (49, 422), (47, 423), (48, 423), (49, 423), (47, 424), (48, 424), (49, 424), (48, 425), (49, 425), (50, 425), (48, 426), (49, 426)]

**Comments on Output:**

Although a lot of circle centers are shown, they are close by points. This results because of the thickness of coin edge. Actual number of coins detected are 12, with 1 coin being slightly ambiguously detected.

**References:**

1. <http://www.aishack.in/tutorials/circle-hough-transform/>
2. <https://homepages.inf.ed.ac.uk/rbf/HIPR2/open.htm>
3. <https://homepages.inf.ed.ac.uk/rbf/HIPR2/close.htm>
4. http://aishack.in/tutorials/hough-transform-basics/