Name- Nischal katara
Reg no - 21BCE10575

Date:
P. No.:

Hough Transform - line

problem:

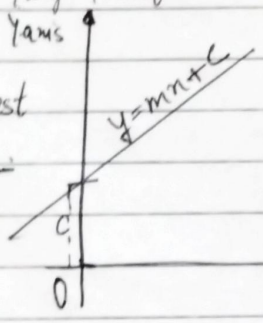
boundary detection - knowing which edges in an image actually correspond the boundary that we are looking for

So, Hough Transform is an ~~efficient~~ very elegant and very powerful way when the boundary is described using a small no of parameters.

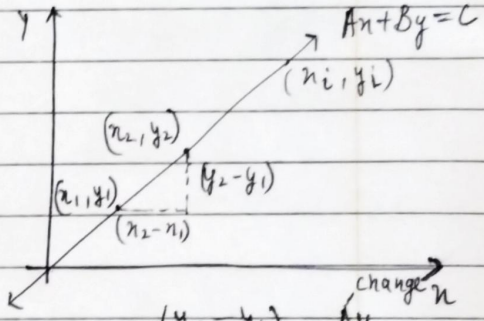
CHALENGES: In the Image Identifying a specific shape, object could be very much difficult makes difficult to Identify
- obstructed or occluded by other objects
- maybe sometimes containing only a certain portion of it
- maybe containing extra edges also with it (Incomplete)
- when we noise in an image may be even a little bit far away which don't correspond to real edges

Line Detection Using Hough Transform

Hough Transform works for simplest Shape like line.



m = slope
c = intercept



$$m = \frac{(y_2 - y_1)}{(x_2 - x_1)} = \frac{\Delta y}{\Delta x}$$

So, this allows us to look at the problem in 2 spaces
- image space (xy space),
- parameter space (mc space)

- $y = mx + c$

so, for, (x_i, y_i) to lie on line we can say,

$$\boxed{y_i = m x_i + c}$$

as we plug (x_i, y_i) we get a line in the mc space (lines in Parameter space)

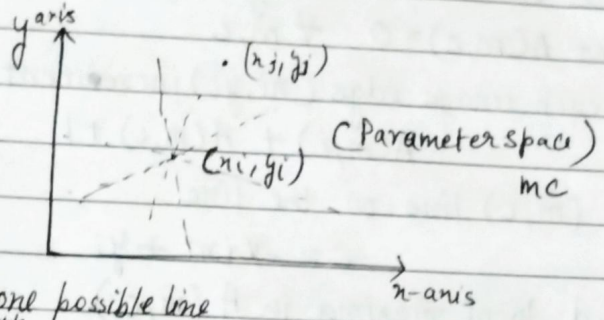
for different values of m and c we could define a line that will pass through the given point (x_i, y_i).

$$\frac{y_i - c}{x_i - 0} = m$$

for some $c = b$, we will get $\frac{y_i - b}{x_i} = m$

a line pass through the given point. ~~Similarly for~~.

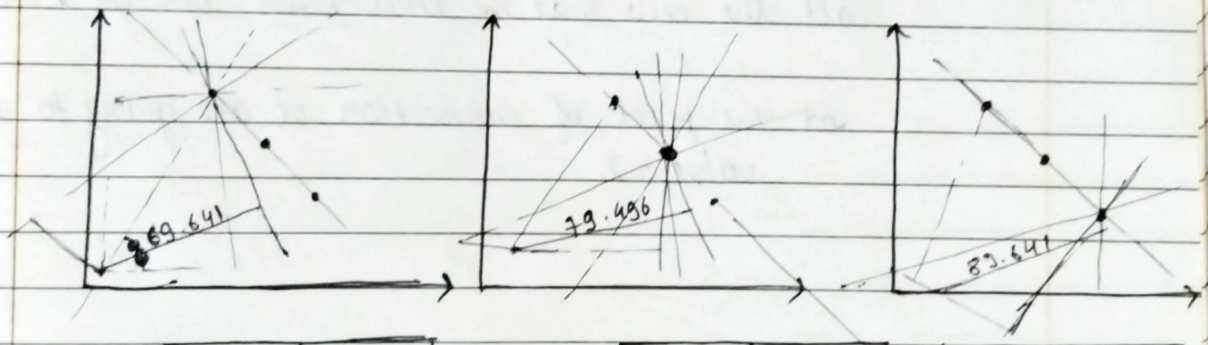
If we take another point



There can be only one possible line that passes through both the given points.

a point in image space maps to a line in parameter space. A line in image space maps to a point in parameter space and vice versa.

Algorithm (Line Detection)



Angle	Dist.
0	40
30	69.6
60	81.2
120	40.6
150	0.4

Angle	Dist
0	57.1
30	79.5
60	80.5
120	23.4
150	-19.5

Angle	Dist.
0	74.6
30	89.6
60	80.6
120	6.6
150	70.6

Date :
P. No. :

Algorithm

Quantize parameter space (m, c)
Create Accumulator Array A (m, c)
Set A(m, c) = 0 ∀ m, c
For each image edge (xi, yi) increament:
A(m, c) + A(m, c) + 1
If (m, c) line on the line:
c = -xim + yi
find local marima in A (m, c)

Accumulator Array :-

for (xi, yi) plugged in to the equation of line $y - mx = c$.

we get, $(y_i - mx_i) = c$

for every m and c we are going to increament all those cells that the lines passes through. Incrementing all the cells that the lines passes through in the space

at the point of intersection we are going to get the value = 2.

Implementation

```
import cv2
import numpy as np

img = cv2.imread('image.jpg')

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

edges = cv2.Canny(gray, 50, 150, apertureSize=3)

lines = cv2.HoughLines(edges, 1, np.pi/180, 200)

for r,theta in lines:
    arr = np.array(r,theta[0], dtype=np.float64)
    r, theta = arr

    a = np.cos(theta)
    b = np.sin(theta)

    x0 = a*r
    y0 = b*r

    x1 = int(x0 + 1000*(-b))

    y1 = int(y0 + 1000*(a))

    x2 = int(x0 + -1000*(-b))

    y2 = int(y0 + -1000*(a))
```

Date :
P. No. :

cv2.line(img, (x1, y1), (x2, y2), (0, 0, 255), 2)

- cv2.imwrite('linesDetected.jpg', img)

plt.imshow(img)