Image Transforms

Hough Transforms

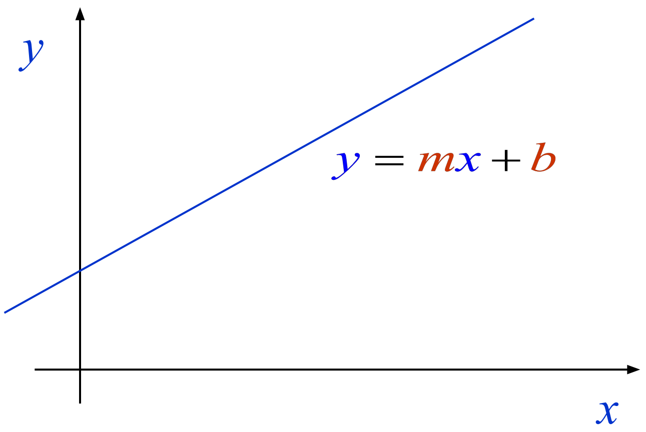
Credited by Image Transforms PPT 虞台文博士, Tatung University, Taiwan

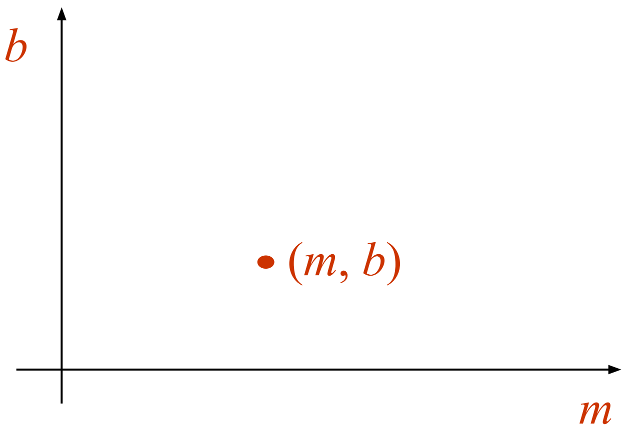
Goal of Hough Transforms

- A technique to isolate the curves of a given shape/shapes in a given image
- Classical Hough Transform
  - can locate regular curves like straight lines, circles, parabolas, ellipses, etc.
- Generalized Hough Transform
  - can be used where a simple analytic description of feature

HT for Line Detection

- A line in *xy*-plane is a point in *mb*-plane.

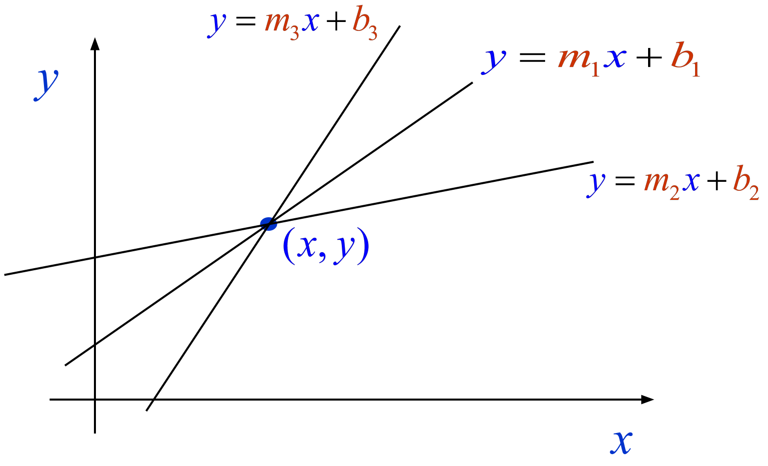


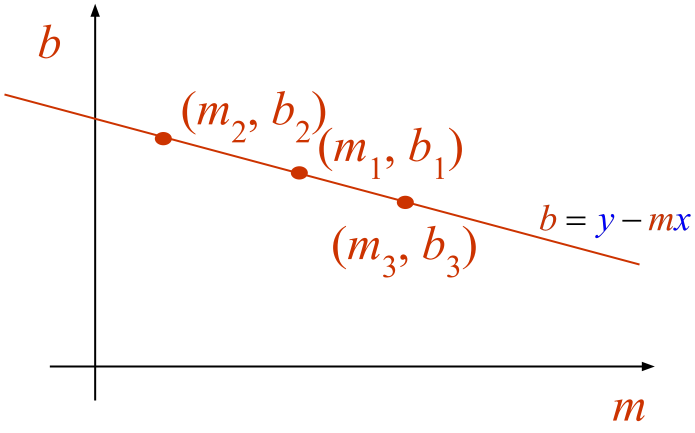


HT for Line Detection

- All lines passing through a point in *xy*-plane is a line in *mb*-plane.

  - A line in *xy*-plane is a point in *mb*-plane.

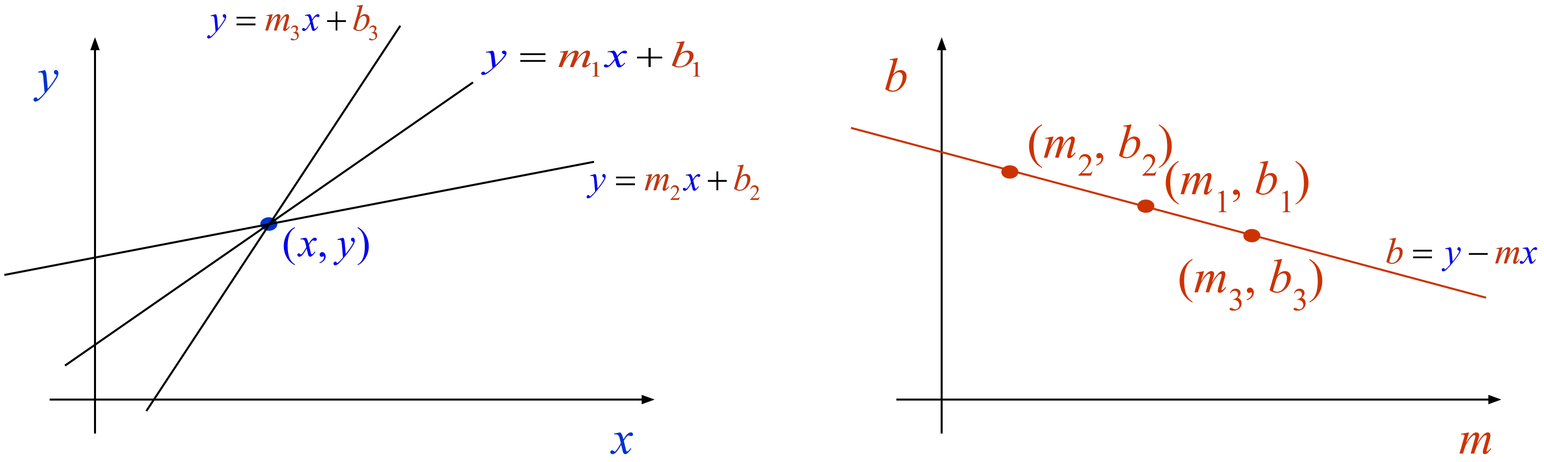




HT for Line Detection

- Given a point in *xy*-plane, we draw a line in *mb*-plane.

- A line in *xy*-plane is a point in *mb*-plane.



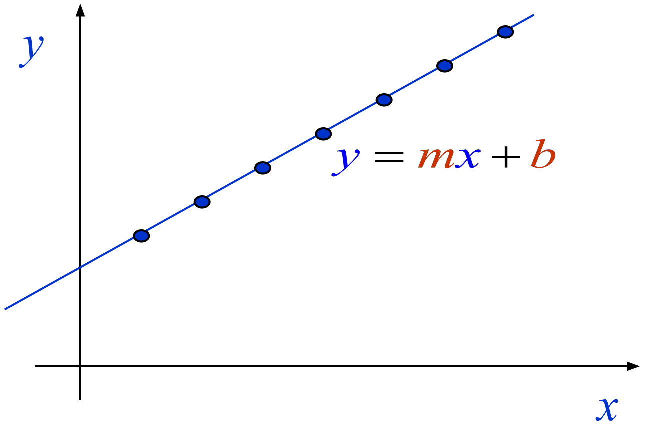
- All lines passing through a point in *xy*-plane is a line in *mb*-plane.

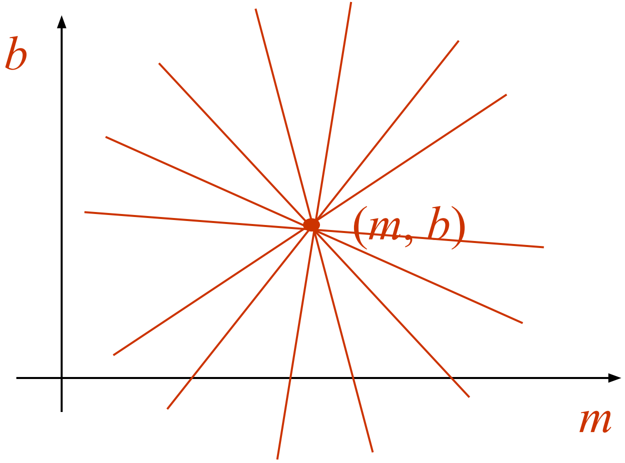
HT for Line Detection

How to implement?

- A line in *xy*-plane is then transformed in to a set of lines in *mb*-plane, which intersect at a common point.

- A line in *xy*-plane is a point in *mb*-plane.

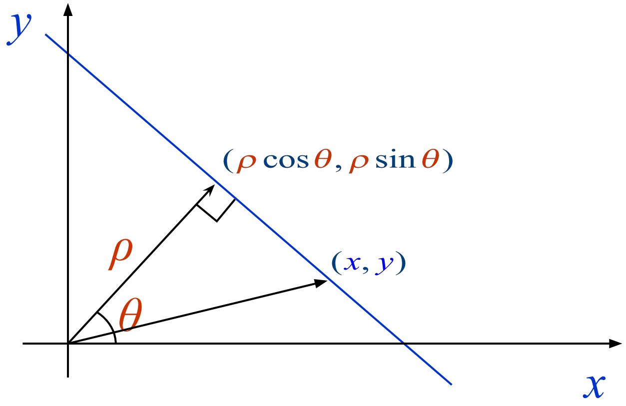




- Given a point in *xy*-plane, we draw a line in *mb*-plane.

HT for Line Detection by $\rho\theta$-representation

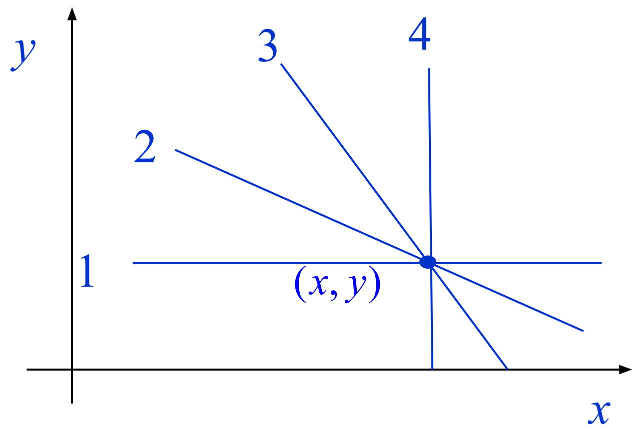
- A line in xy-plane is a point in $\rho\theta$-plane.

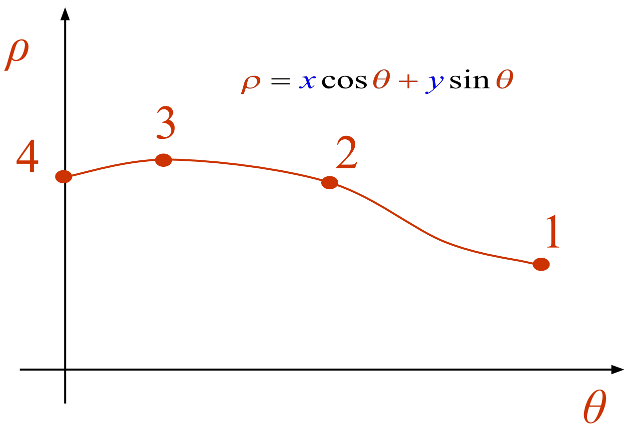


HT Line Detection by $\rho\theta$-representation

- All lines passing through a point in xy-plane is a curve in $\rho\theta$ -plane.

  - A line in xy-plane is a point in $\rho\theta$-plane.



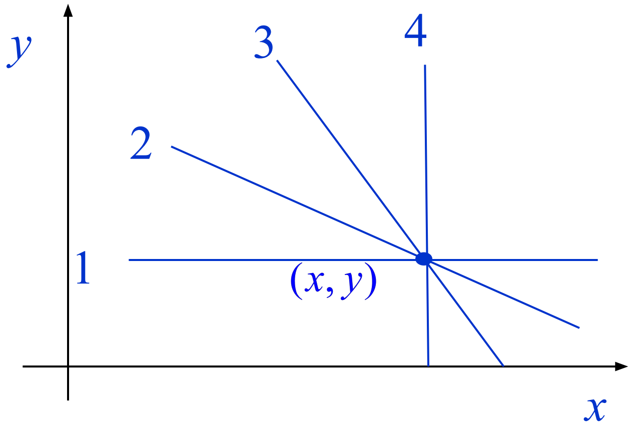


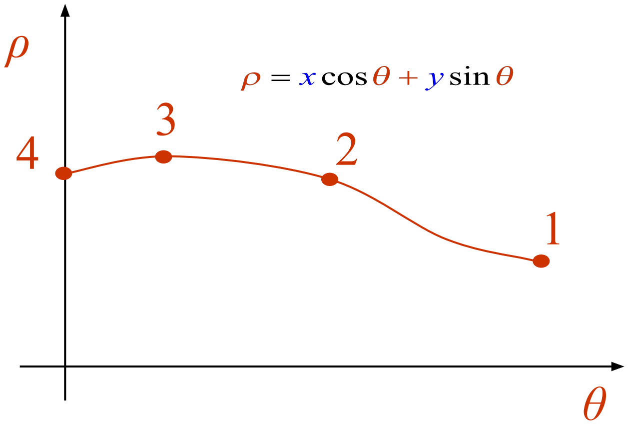
HT Line Detection by $\rho\theta$-representation

- Given a point in xy-plane, we draw a curve in $\rho\theta$ -plane.

  - A line in xy-plane is a point in $\rho\theta$-plane.

  - All lines passing through a point in xy-plane is a curve in $\rho\theta$ -plane.

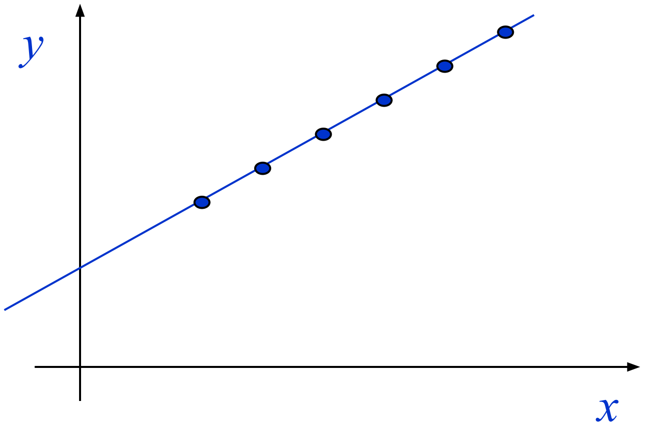


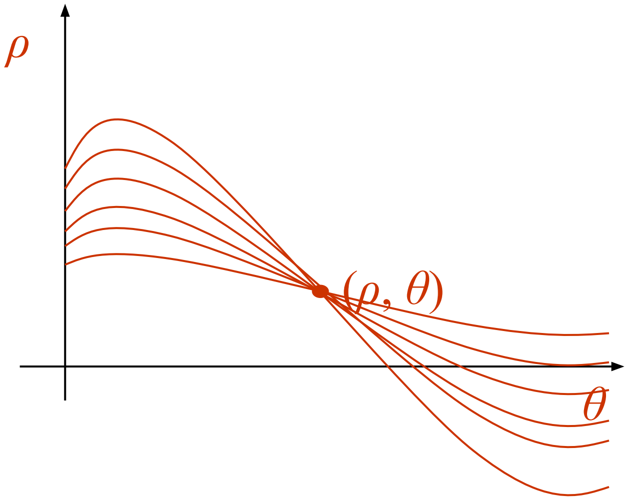


HT Line Detection by *ρθ*-representation

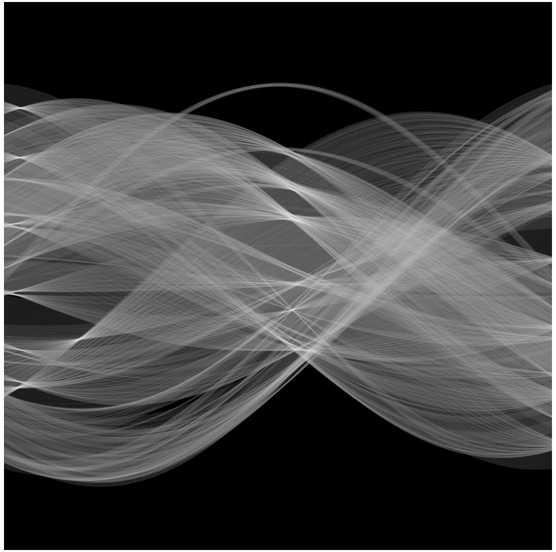
- A line in *xy*-plane is then transformed in to a set of curves in $\rho\theta$ -plane, which intersect at a common point.

  - Given a point in *xy*-plane, we draw a curve in $\rho\theta$ -plane.





HT Line Detection by $\rho\theta$-representation



A line in xy-plane is a point in $\rho\theta$-plane.

Given a point in xy-plane, we draw a curve in $\rho\theta$ -plane.

A line in xy-plane is then transformed in to a set of curves in $\rho\theta$ -plane, which intersect at a common point.

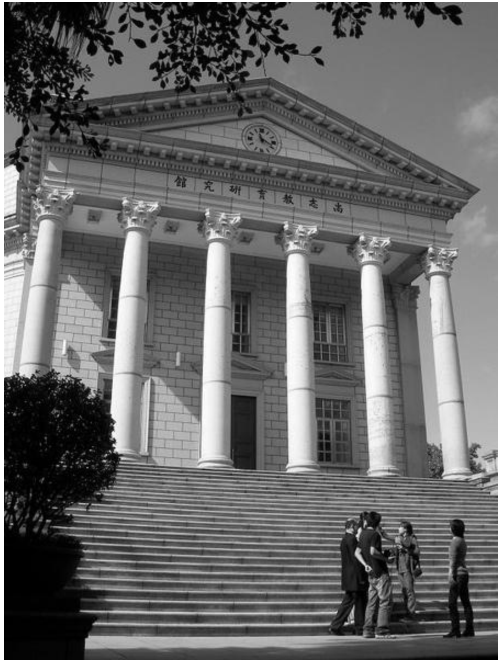
OpenCV —Hough Line Transform

```
# Standard Hough Line Transform
lines = cv.HoughLines(dst, 1, np.pi / 180, 150, None, 0, 0)
```

- *dst* : Output of the edge detector. It should be a grayscale image (although in fact it is a binary one)
- *lines* : A vector that will store the parameters (r,θ) of the detected lines
- *rho* : The resolutio of the parameter r in pixels. We use **1** pixel.
- *theta* : The resolution of the parameter θ in radians. We use **1 degree** (CV_PI/180)
- *threshold* : The minimum number of intersections to "*detect*" a line
- *srn* and *stn* : Default parameters to zero. Check OpenCV reference for more info.

```
lines = cv2.HoughLines(edges,1,np.pi/180,200)
for rho,theta in lines[0]:
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a*rho
    y0 = b*rho
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))
    cv2.line(img,(x1,y1),(x2,y2),(0,0,255),2)
```

Example: Hough Line Transform



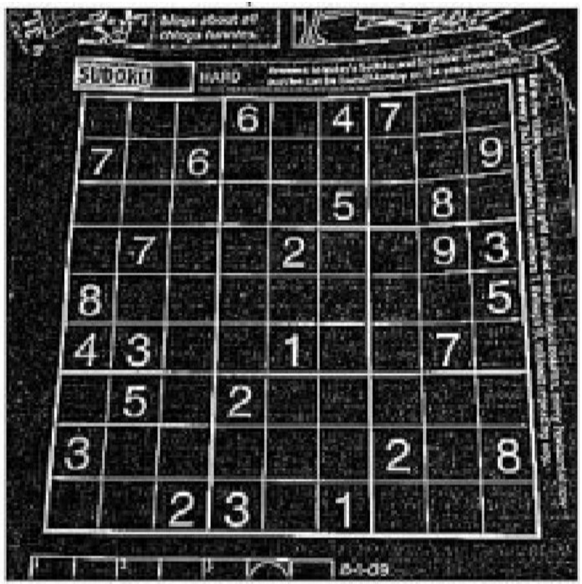
OpenCV —Hough Line Transform

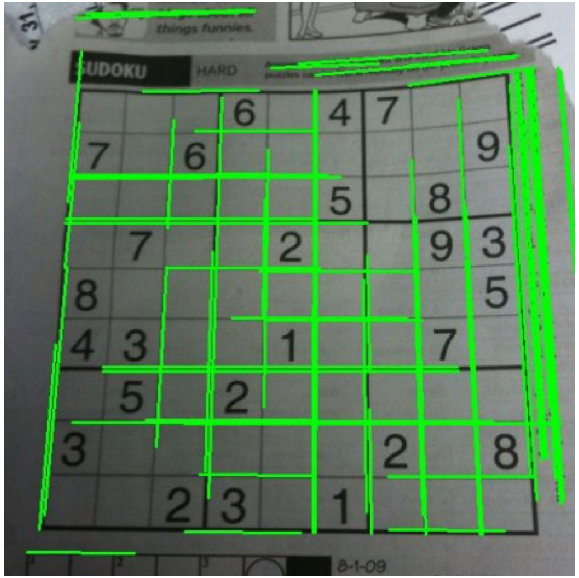
```
# Probabilistic Line Transform
linesP = cv.HoughLinesP(dst, 1, np.pi / 180, 50, None, 50, 10)
```

- dst: Output of the edge detector. It should be a grayscale image (although in fact it is a binary one)
- lines: A vector that will store the parameters (xstart,ystart,xend,yend) of the detected lines
- rho : The resolution of the parameter r in pixels. We use 1 pixel.
- theta: The resolution of the parameter θ in radians. We use 1 degree (CV_PI/180)
- threshold: The minimum number of intersections to "*detect*" a line
- minLinLength: The minimum number of points that can form a line. Lines with less than this number of points are disregarded.
- maxLineGap: The maximum gap between two points to be considered in the same line.

```
minLineLength = 100
maxLineGap = 10
lines = cv2.HoughLinesP(edges,1,np.pi/180,100,minLineLength,maxLineGap)
for x1,y1,x2,y2 in lines[0]:
    cv2.line(img,(x1,y1),(x2,y2),(0,255,0),2)
```

Example: Hough Line Transform





Hough Circle Transform

Circle equation : $(x-a)^2+(y-b)^2=r^2$

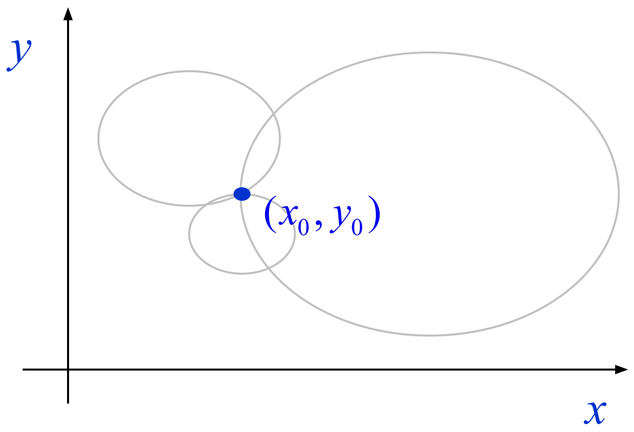
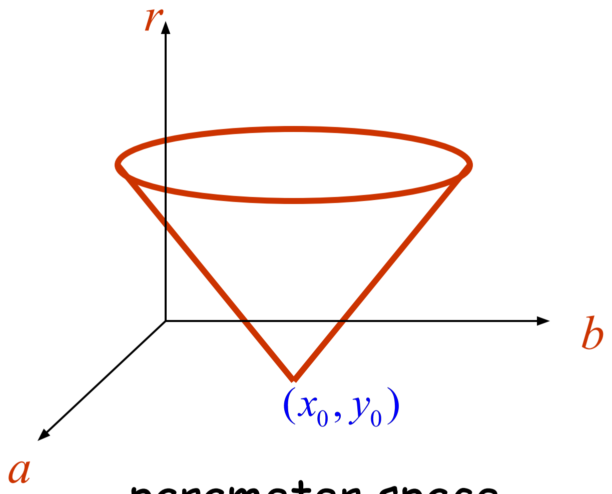


image space



parameter space

OpenCV —Hough Circle Transform

```
circles = cv.HoughCircles(gray, cv.HOUGH_GRADIENT, 1, rows / 8,
    param1=100, param2=30, minRadius=1, maxRadius=30)
```

- *gray* : Input image (grayscale).
- *circles* : A vector that stores sets of 3 values: xc,yc,r for each detected circle.
- *HOUGH_GRADIENT*: Define the detection method. Currently this is the only one available in OpenCV.
- *dp = 1* : The inverse ratio of resolution.
- *min_dist = gray.rows/16*: Minimum distance between detected centers.
- *param_1 = 200*: Upper threshold for the internal Canny edge detector.
- *param_2* = 100*: Threshold for center detection.
- *min_radius = 0*: Minimum radius to be detected. If unknown, put zero as default.
- *max_radius = 0*: Maximum radius to be detected. If unknown, put zero as default.

```
img = cv2.imread('opencv_logo.png',0)
img = cv2.medianBlur(img,5)
cimg = cv2.cvtColor(img,cv2.COLOR_GRAY2BGR)
circles = cv2.HoughCircles(img,cv2.HOUGH_GRADIENT,1,20,
    param1=50,param2=30,minRadius=0,maxRadius=0)
circles = np.uint16(np.around(circles))
for i in circles[0,:]:
    # draw the outer circle
    cv2.circle(cimg,(i[0],i[1]),i[2],(0,255,0),2)
    # draw the center of the circle
    cv2.circle(cimg,(i[0],i[1]),2,(0,0,255),3)
```

Example: Hough Circle Transform