# SciComp with Py

# Hough Transform

Vladimir Kulyukin
Department of Computer Science
Utah State University

# Outline

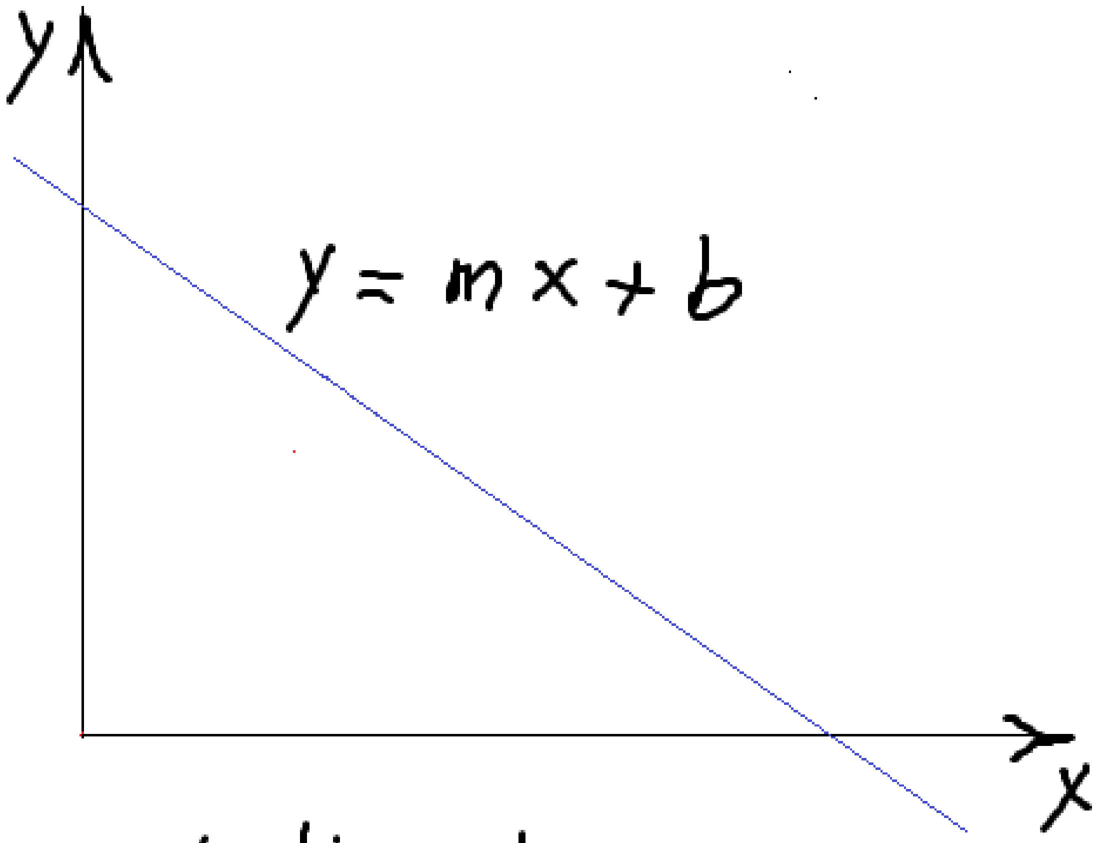
- Basic Concepts
    - Point-slope representation of lines
    - Parameterized representation of lines
    - 2D points & sine curves
- Hough Transform Algorithm
- Hough Transform in OpenCV

# Basic Concepts

# Hough Transform

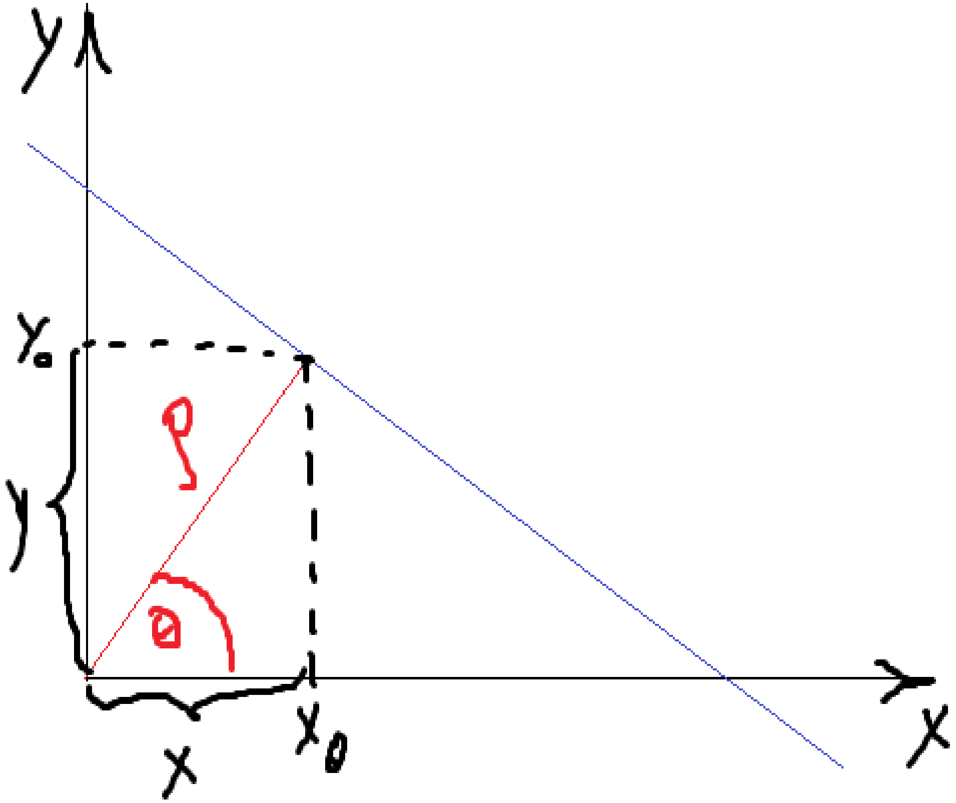
- Paul Hough invented the concept in 1959 and patented it in 1962
- Richard Duda and Peter Hart used Hough's idea in image analysis and introduced it to computer vision and image analysis as the **Hough Transform**
- It is used in computer vision primarily for line detection

# Point-Slope Representation of Lines

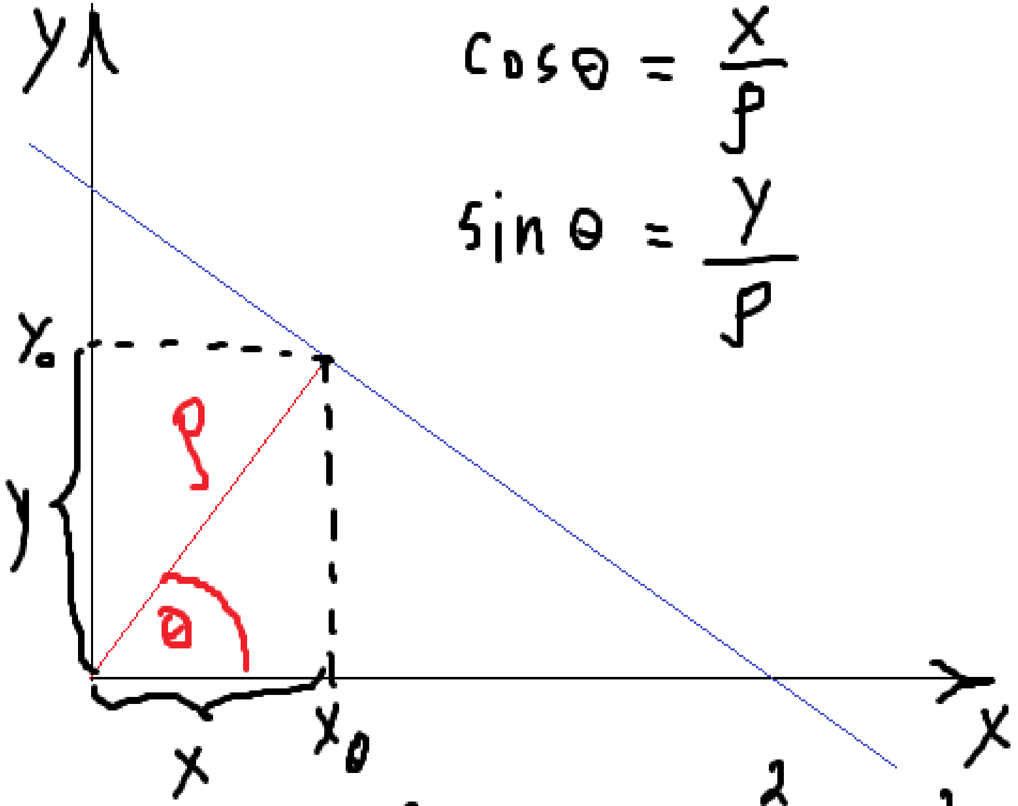


Each line is represented as $(m, b)$

# Parameterized Representation of Lines



Each line is represented as $(\rho, \theta)$



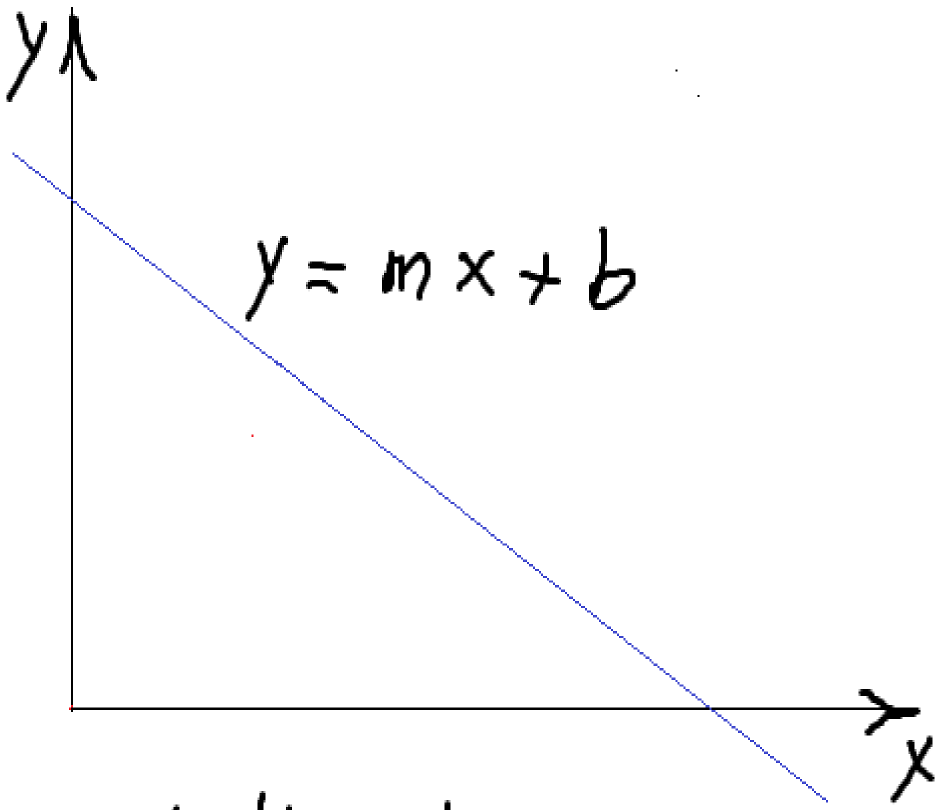
$$\cos\theta = \frac{x}{\rho}$$

$$\sin\theta = \frac{y}{\rho}$$

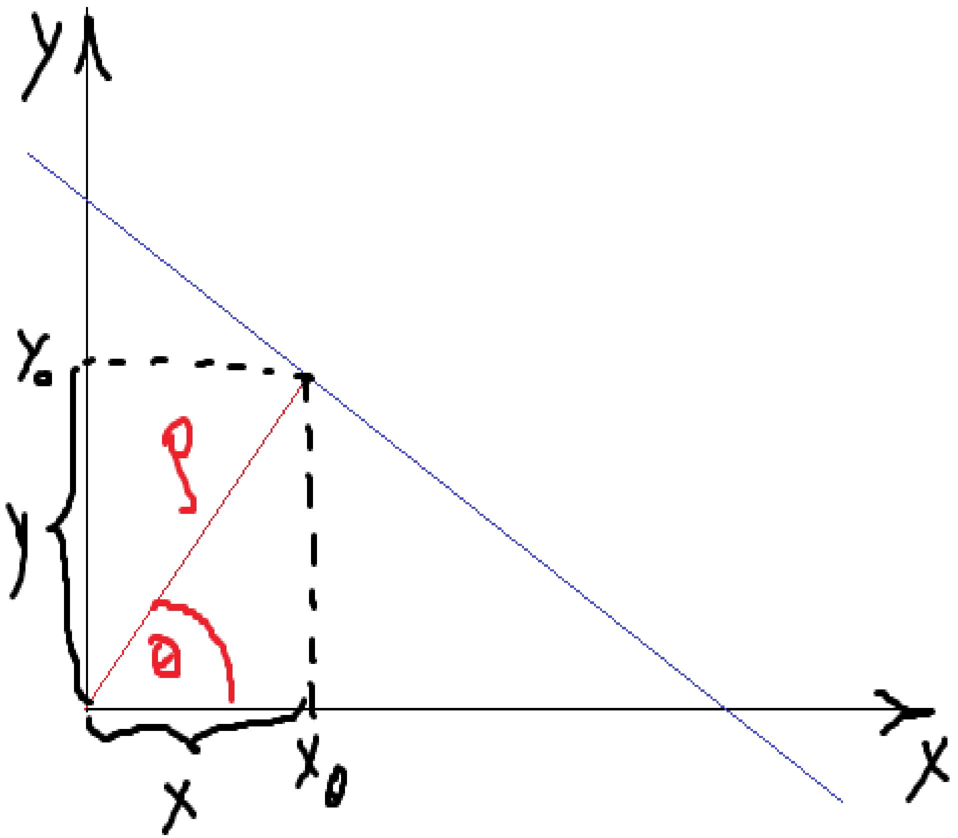
$$\rho^2 = x^2 + y^2 \Rightarrow \rho = \frac{x^2}{\rho} + \frac{y^2}{\rho} =$$

$$x\frac{x}{\rho} + y\frac{y}{\rho} = x\cos\theta + y\sin\theta$$

# Two Representations Side by Side

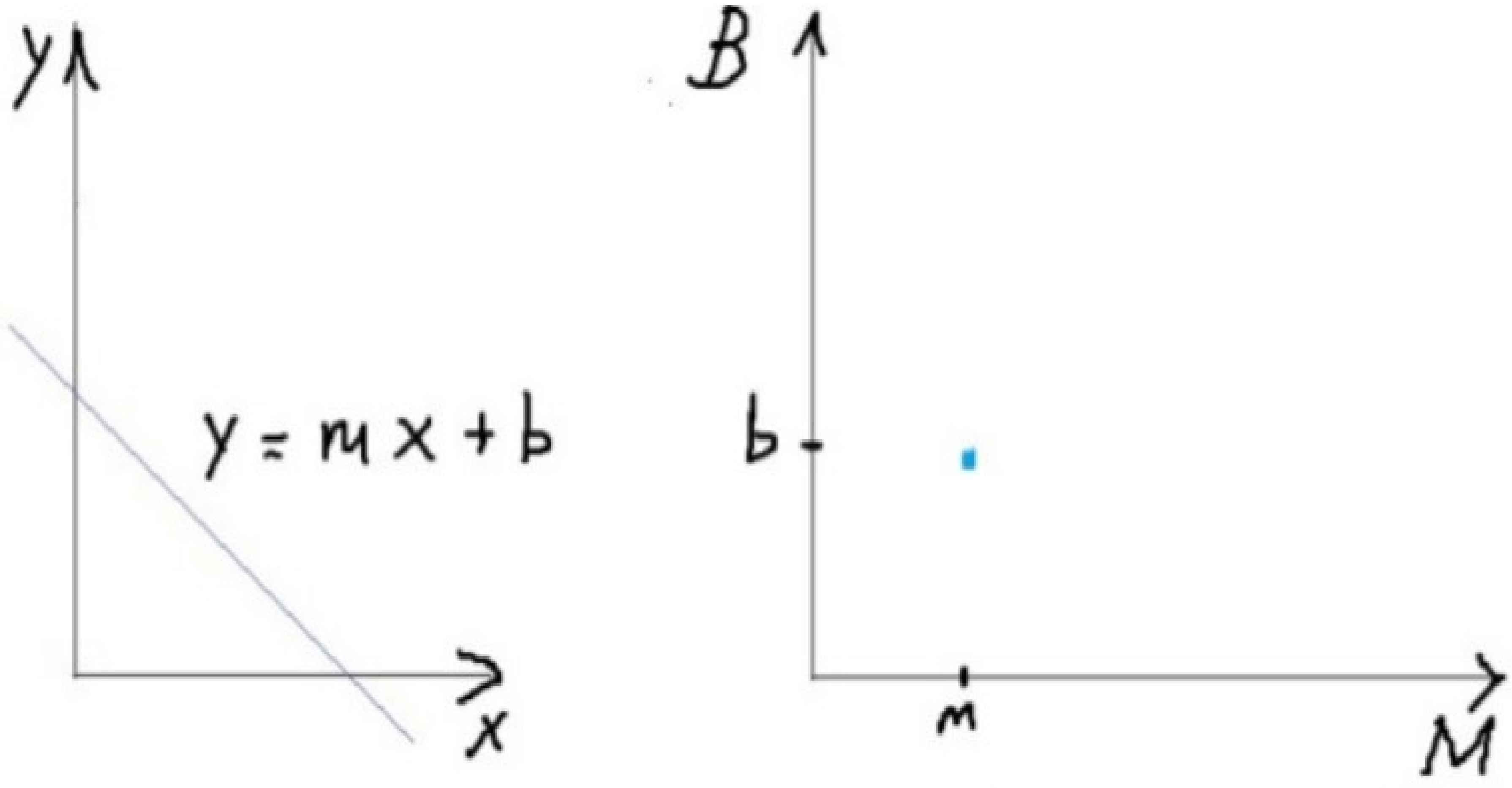


Each line is represented as (m, b)



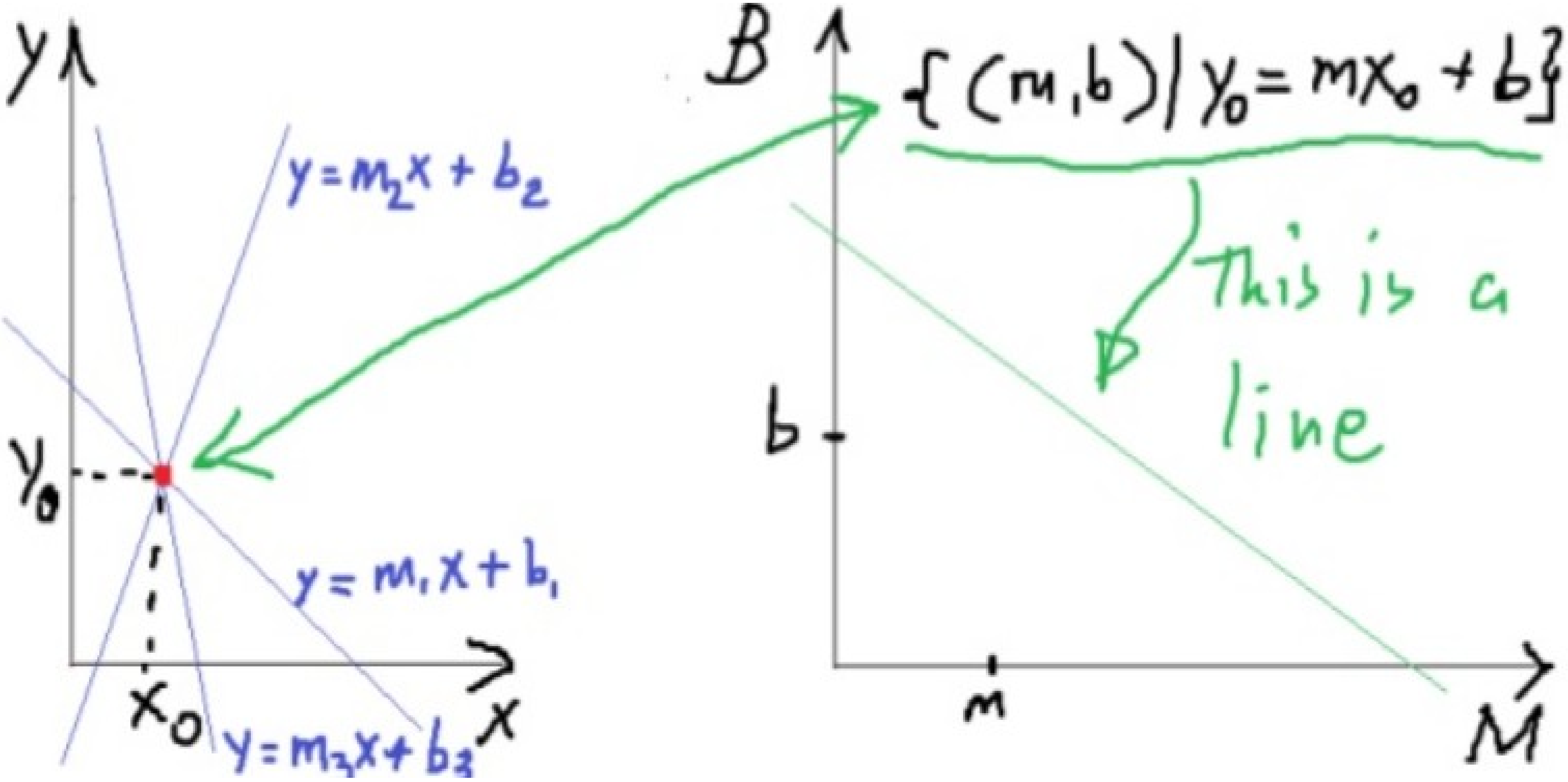
Each line is represented as (ρ, θ)

# From Euclid Plane Lines to Hough Plane Points



A line y = mx+b in Euclid Plane (left) corresponds to a point (m, b) in Hough Plane (right)

# From Euclid Points to Hough Plane Lines

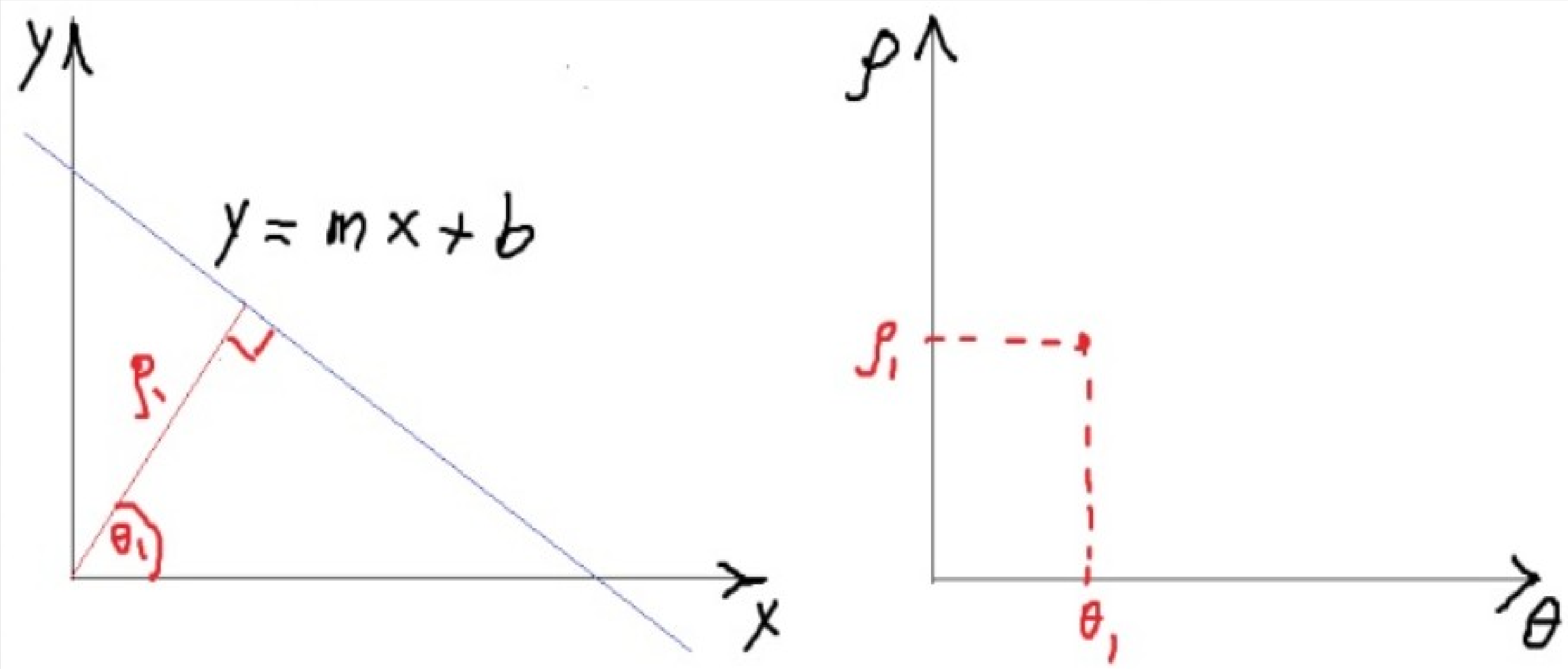


A point (x0, y0) in Euclid Plane (left) corresponds to a line in Hough Plane (right)

# Problems with Hough Plane

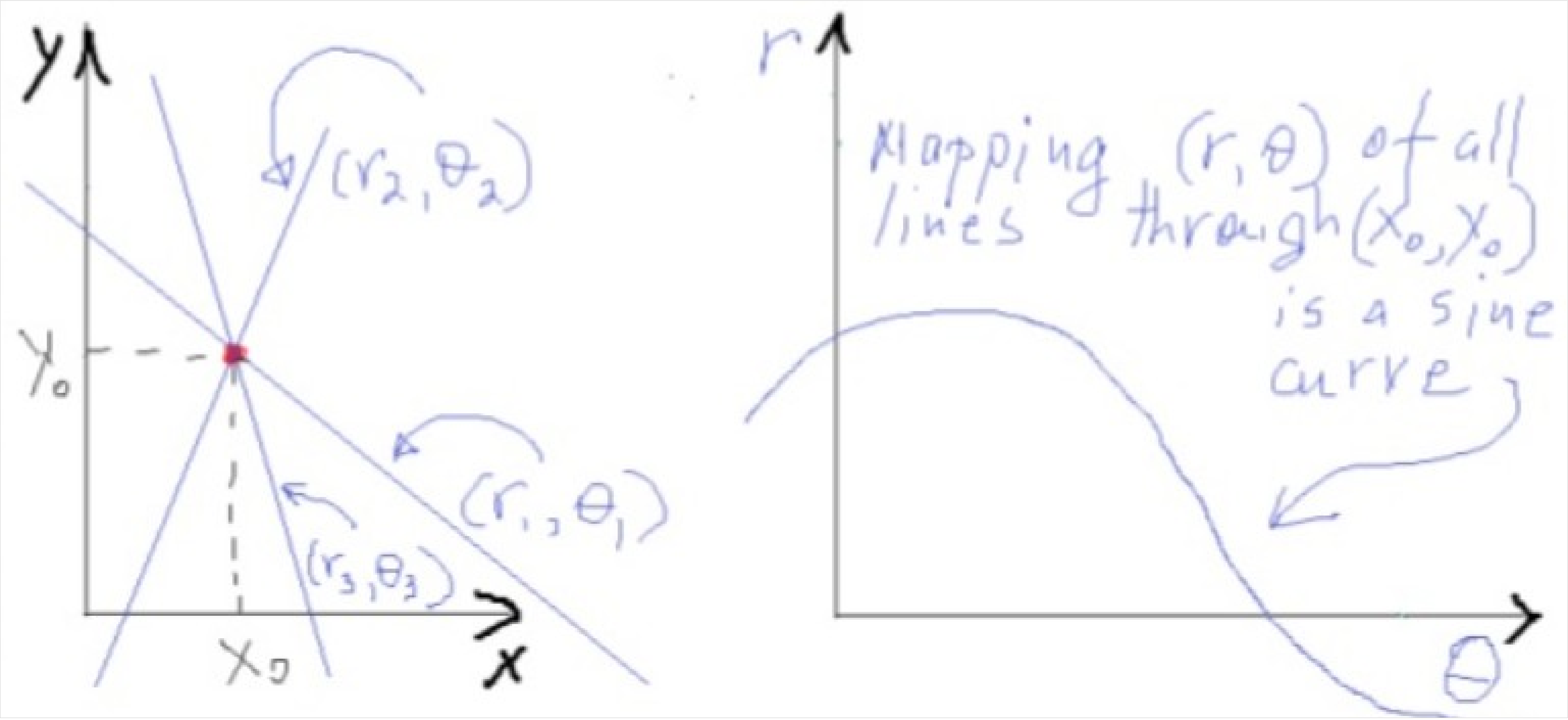
- Problem 1: Vertical lines cannot be represented because their slopes are not defined
- Problem 2: Theoretically, values of M and B range from -Infinity to +Infinity
- Problem 1 can be solved with meaningful domain-specific thresholds
- Problem 2 can be solved with finite M x B matrices

# Rho-Theta Representation of Hough Plane



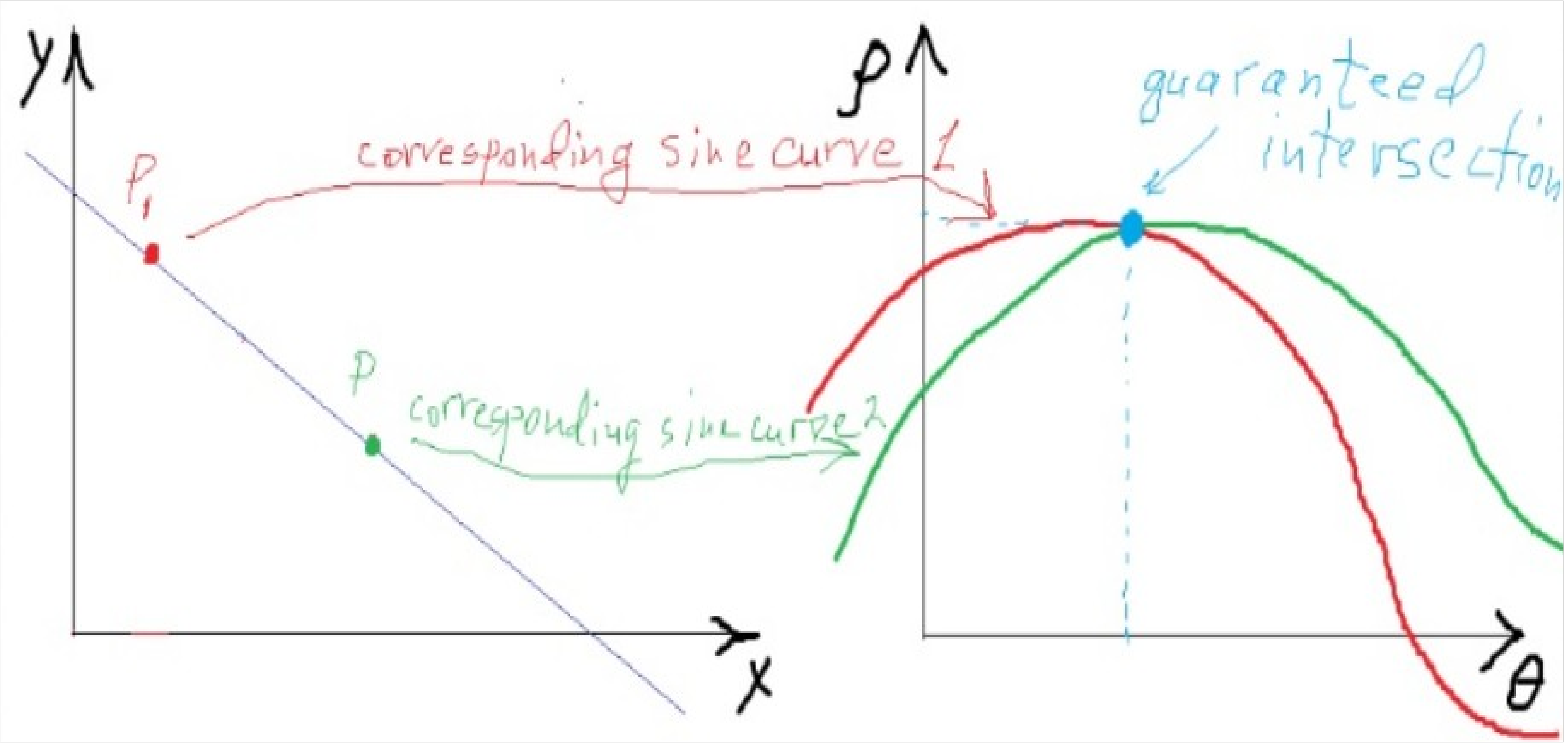
Instead of using M and B to represent Hough Plane, we can use Rho and Theta

# Rho-Theta Representation


y
x
$y_0$
$x_0$
$(r_2, \theta_2)$
$(r_1, \theta_1)$
$(r_3, \theta_3)$
r
θ
Mapping $(r, \theta)$ of all lines through $(x_0, y_0)$ is a sine curve

All (r, theta) pairs corresponding to all lines passing through a point (x0, y0) in Euclid Plane form a sine curve in Hough Plane; this is astonishing when you think about it!

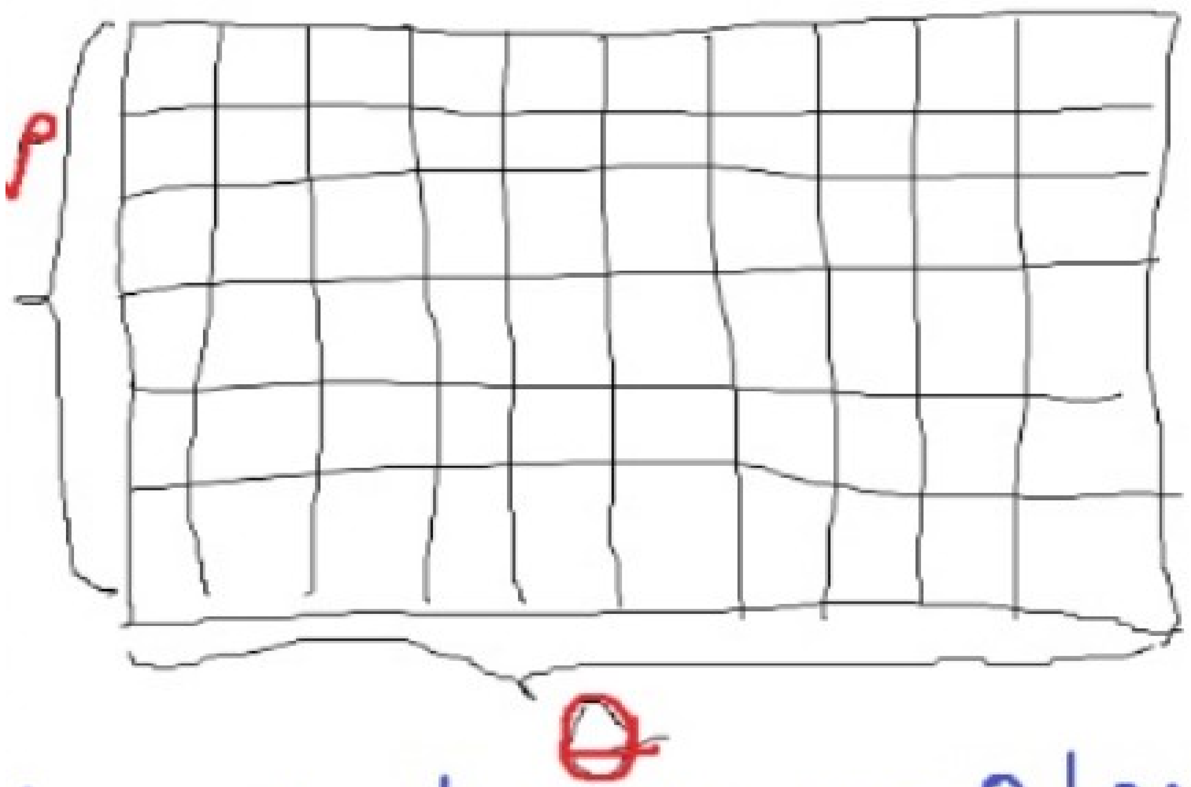
# Rho-Theta Representation



Another remarkable fact: The sine curves that correspond to any two collinear points in Euclid Plane (left) are guaranteed to intersect in Hough Plane (right)

# HT Algorithm

# Step 1: Create a Rho-Theta Table



lower $\rho$ bound $\leq \rho \leq$ upper $\rho$ bound

lower $\theta$ bound $\leq \theta \leq$ upper $\theta$ bound

Choose suitable integer bounds for Rho and Theta and create 2D matrix; let us call this matrix HT (i.e., Hough Transform)

# Step 2: Compute Gradients

Given image Img (2D matrix), compute gradients at each cell of Img (see lecture on edge detection on how to compute gradients).

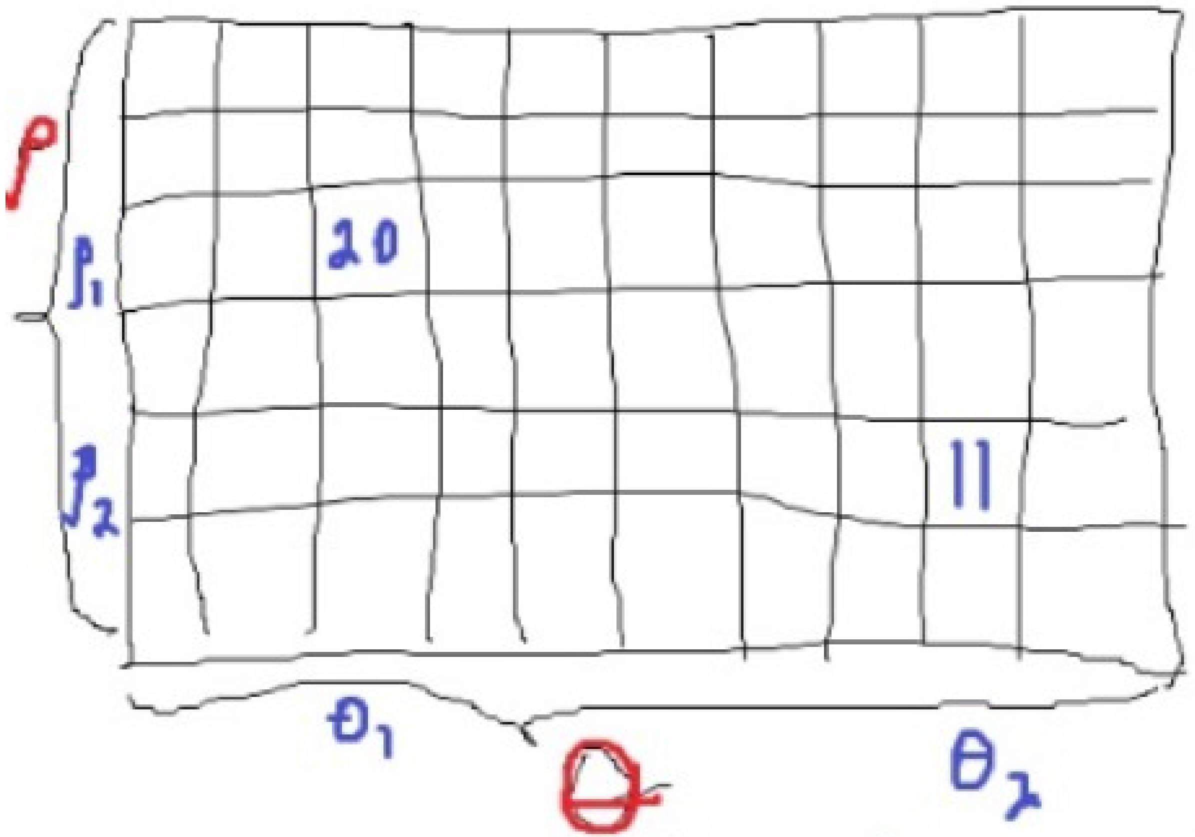
# Step 3: Compute HT Values

```
For each point P(x, y) in Img with sufficiently large gradient
    For each value th of Theta in [0, 180]
        rho = int(x * cos(th) + y*sin(th))
        HT[th, rho] += 1
```

# Step 4: Select HT Cells

Select those cells in HT[th, rho] for which the integer value in HT[th, rho] is above a threshold. Recall that each cell in HT[th, rho] represents a line in Euclid Space. The selected cells correspond to likely lines. The integer values in HT are sometimes called support levels.

# Step 4: Example



Suppose support level = 10, then $(\rho_1, \theta_1)$ and $(\rho_2, \theta_2)$ clear it

# Fundamental Question

What does it mean when HT[rho, theta] has a large support level?

# Answer

It means that there is likely to be a (rho, theta) line in the image Img.

# Hough Transform in OpenCV

# Two HT Methods in OpenCV

Determines number of rows in HT table

Determines number of columns  in HT table

This is support level threshold

```
cv2.HoughLines(image, rho_accuracy, theta_accuracy, support_level)

cv2.HoughLinesP(image, rho_accuracy, theta_accuracy, support_level,
                min_len, max_gap)
```

Minimum length of lines

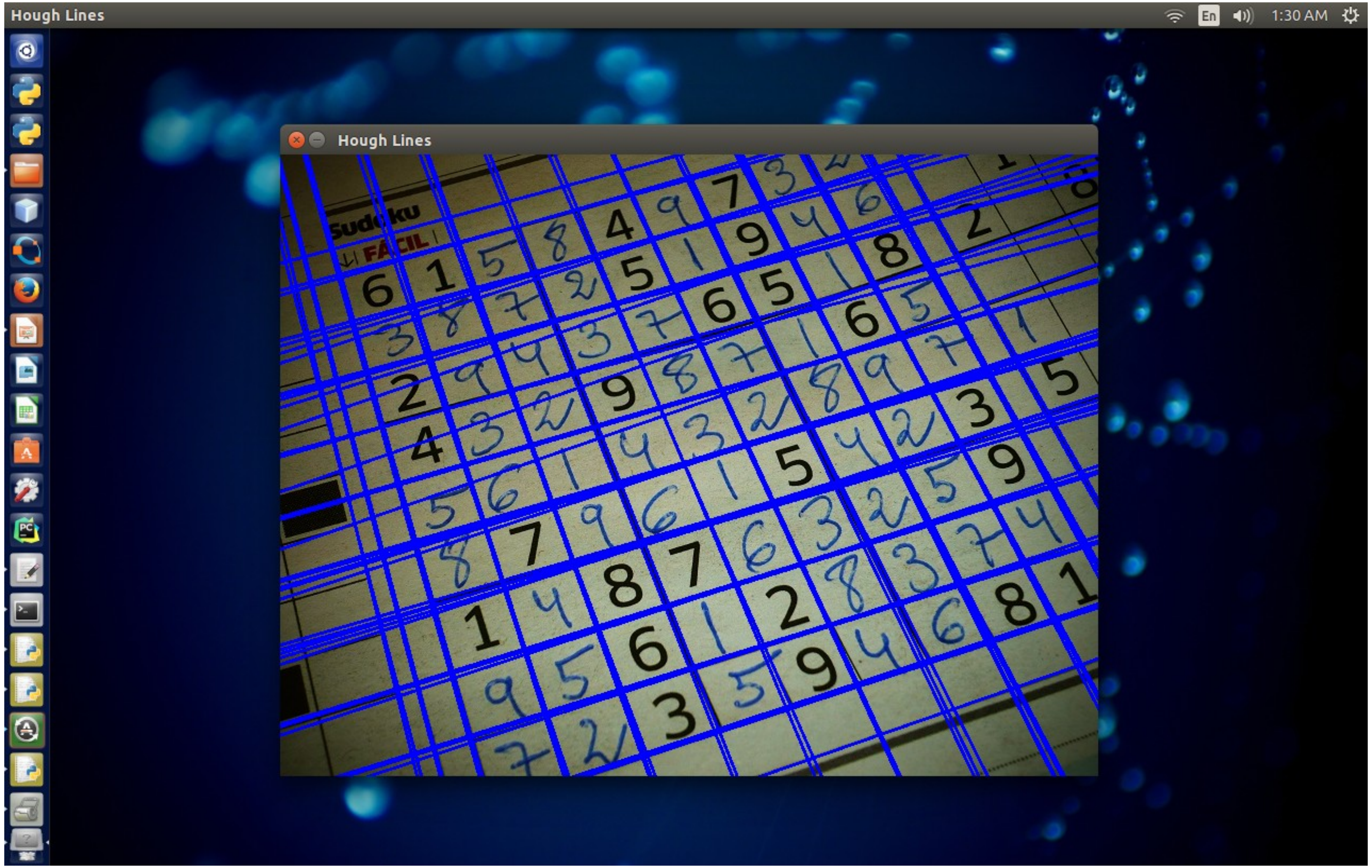
Max gap in lines

# Problem

Write a program that takes all required parameters to run cv2.HoughLines() and displays all detected lines in the original image as well as all intermediate images generated to detect the lines.

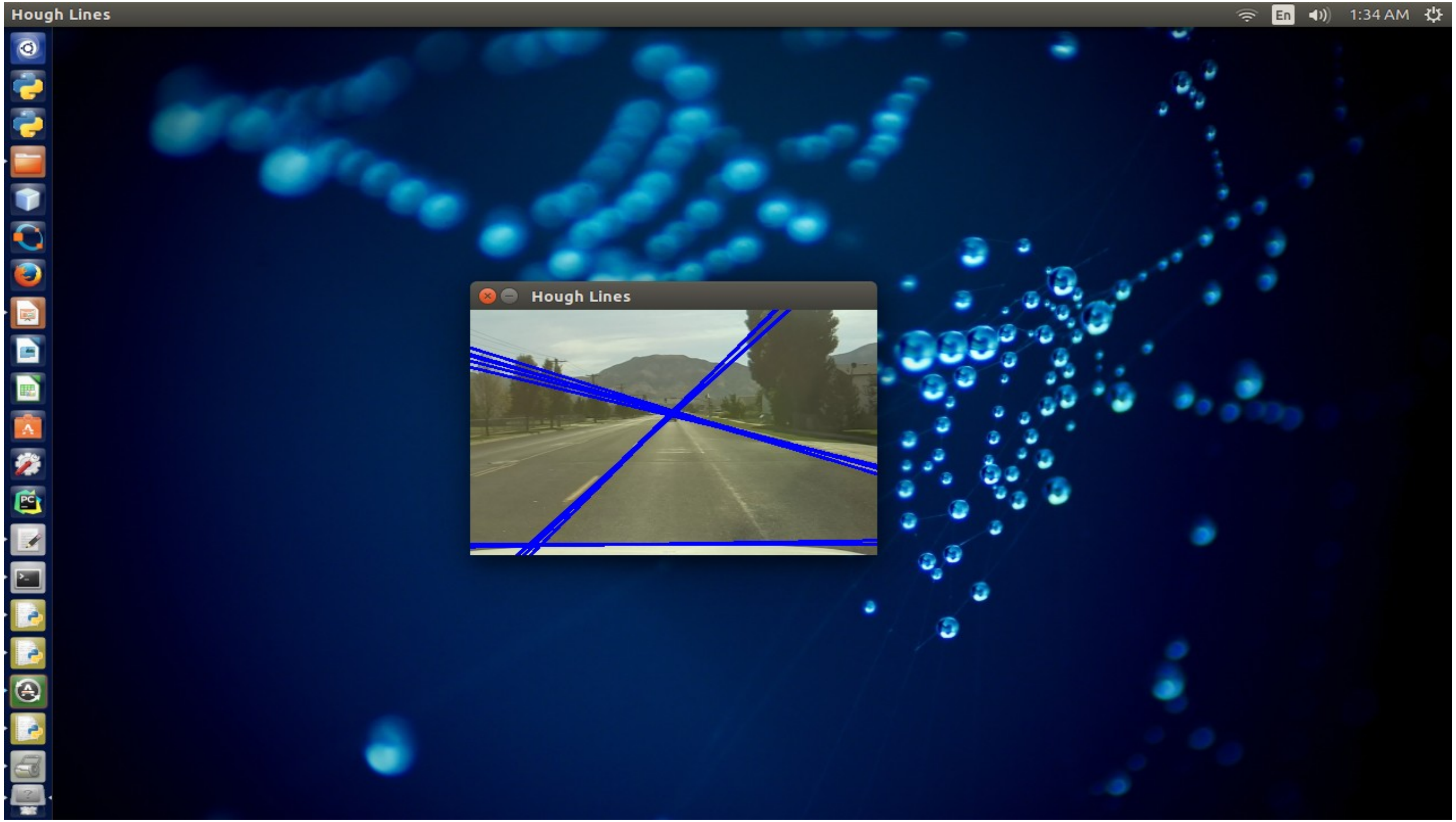
py souce  in houghlines.py

# Sample Output



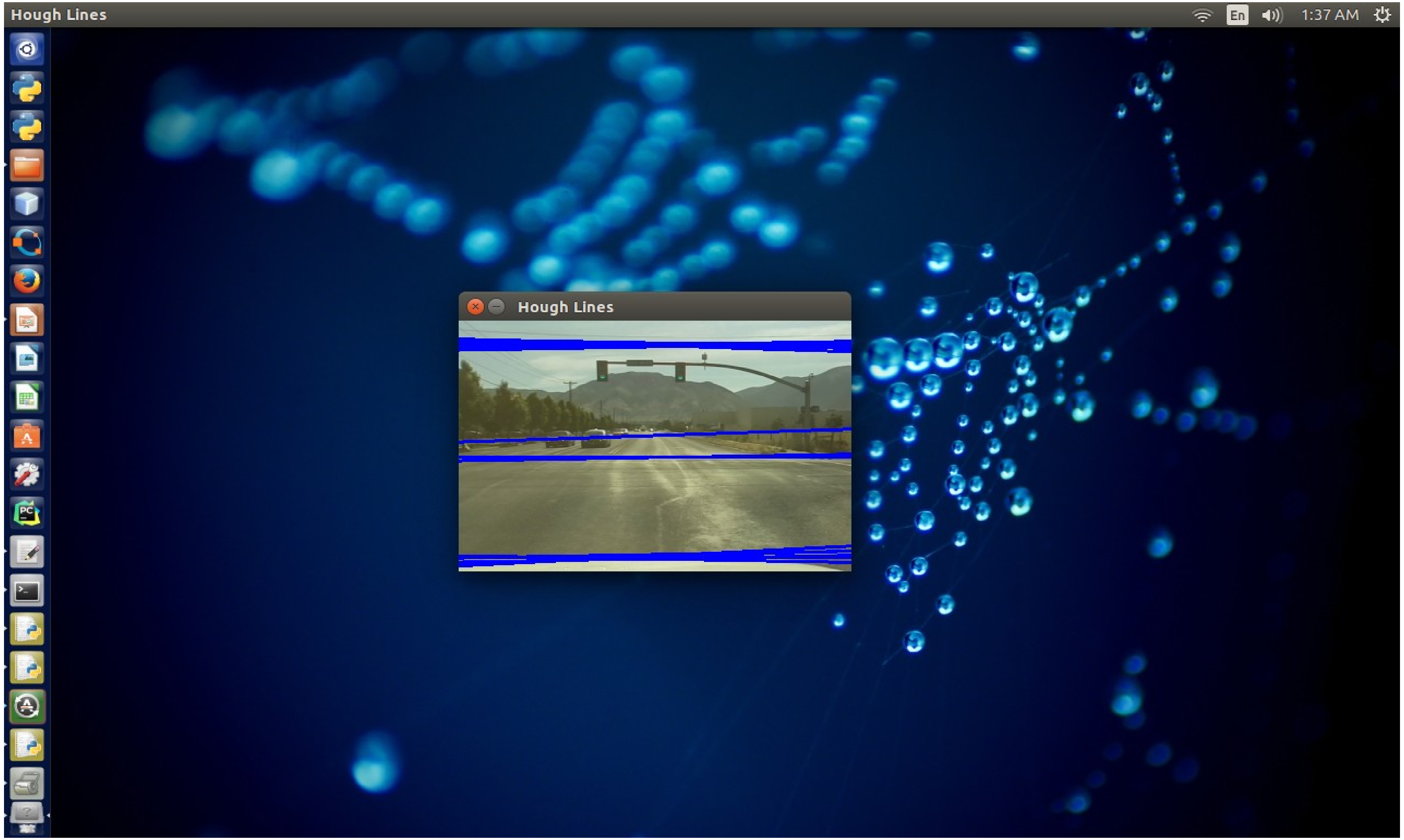
```
$ python houghlines.py -i sudoku.jpg -spl 200
```

# Sample Output



```
$ python houghlines.py -i 01.png -spl 100
```

# Sample Output



```
$ python houghlines.py -i 02.png -spl 100
```

# Solution

```
ap = argparse.ArgumentParser()
ap.add_argument('-i', '--img', required=True, help='path to image')
ap.add_argument('-spl', '--spl', required=True, help='support level', type=int)
args = vars(ap.parse_args())

# load the image
image = cv2.imread(args['img'])
# Grayscale and apply Canny edge detector
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray, 100, 170, apertureSize = 3)
```

# Solution

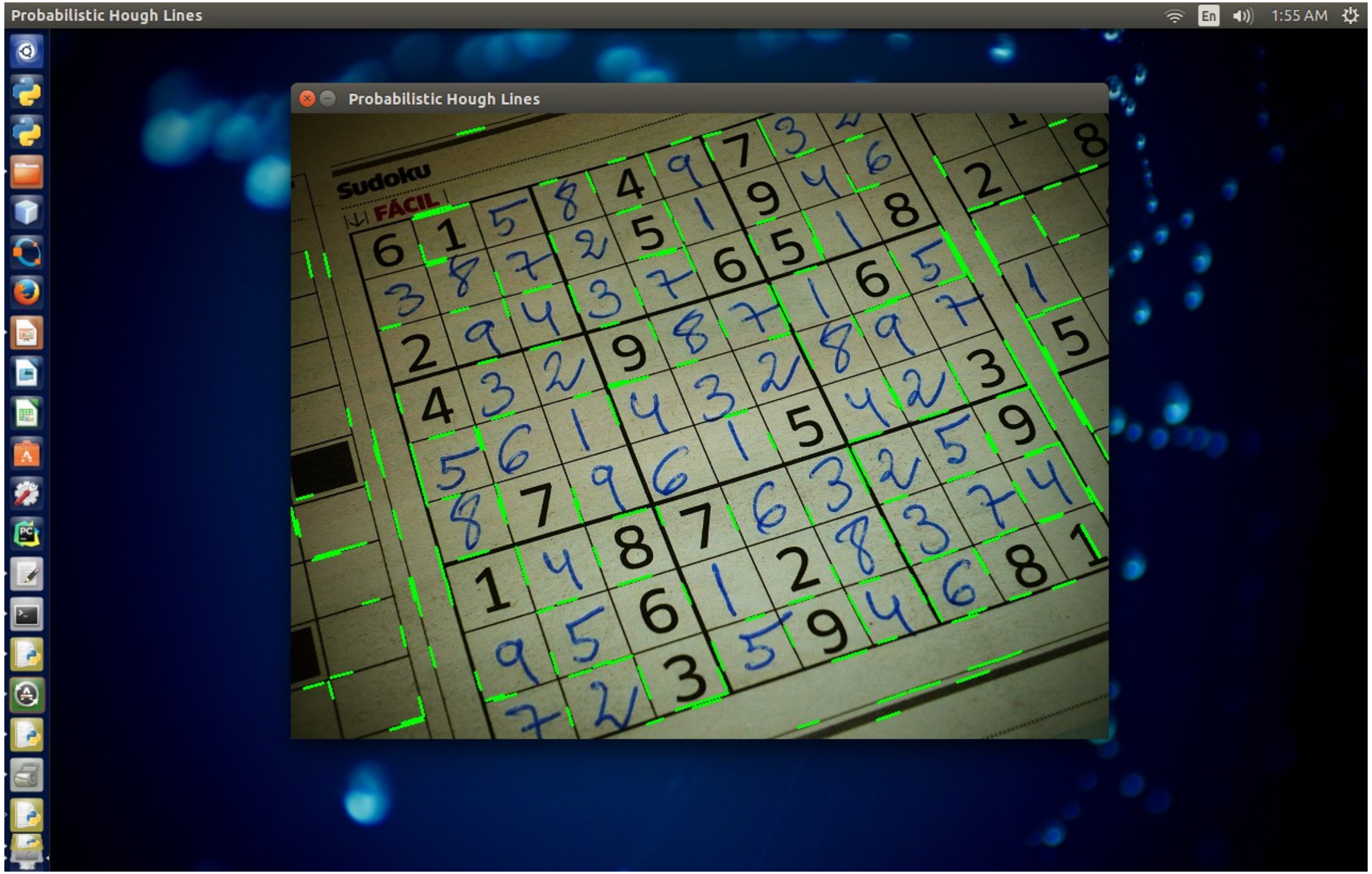
```
lines = cv2.HoughLines(edges, 1, np.pi/180, args['spl'])
# Iterate through each line and convert it to the format required by cv.lines  (i.e. requiring end points)
if not lines is None:
    for ln in lines:
        rho, theta = ln[0]
        # this is some trigonometry to convert rho and theta to two points on the rho-theta line: (x1, y1) and (x2, y2).
        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(image, (x1, y1), (x2, y2), (255, 0, 0), 2)
```

# Problem

Write a program that takes all required parameters to run cv2.HoughLinesP() and displays all detected lines in the original image as well as all intermediate images in generates to detect the lines.

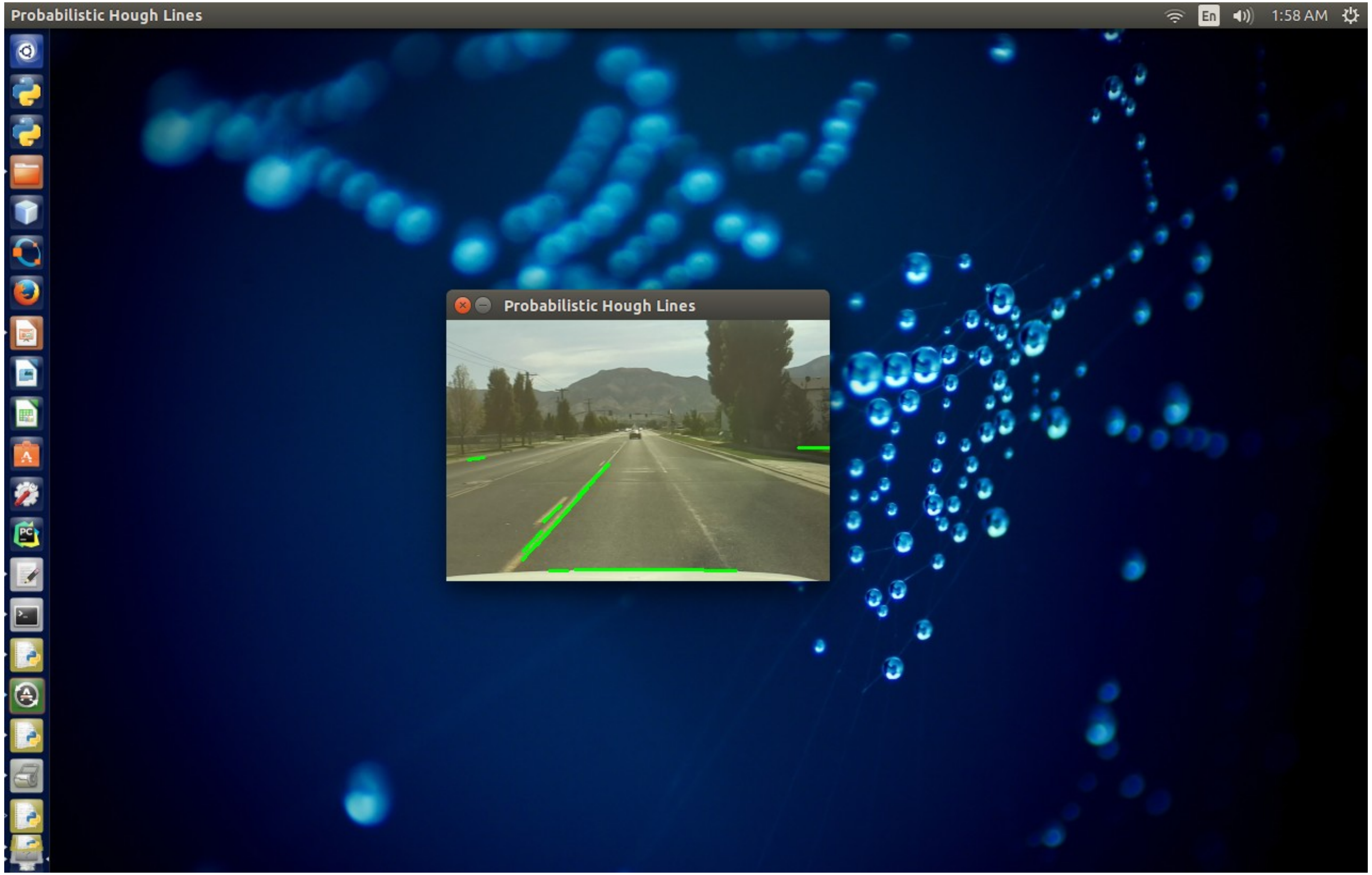
py souce  in prob_houghlines.py

# Sample Output



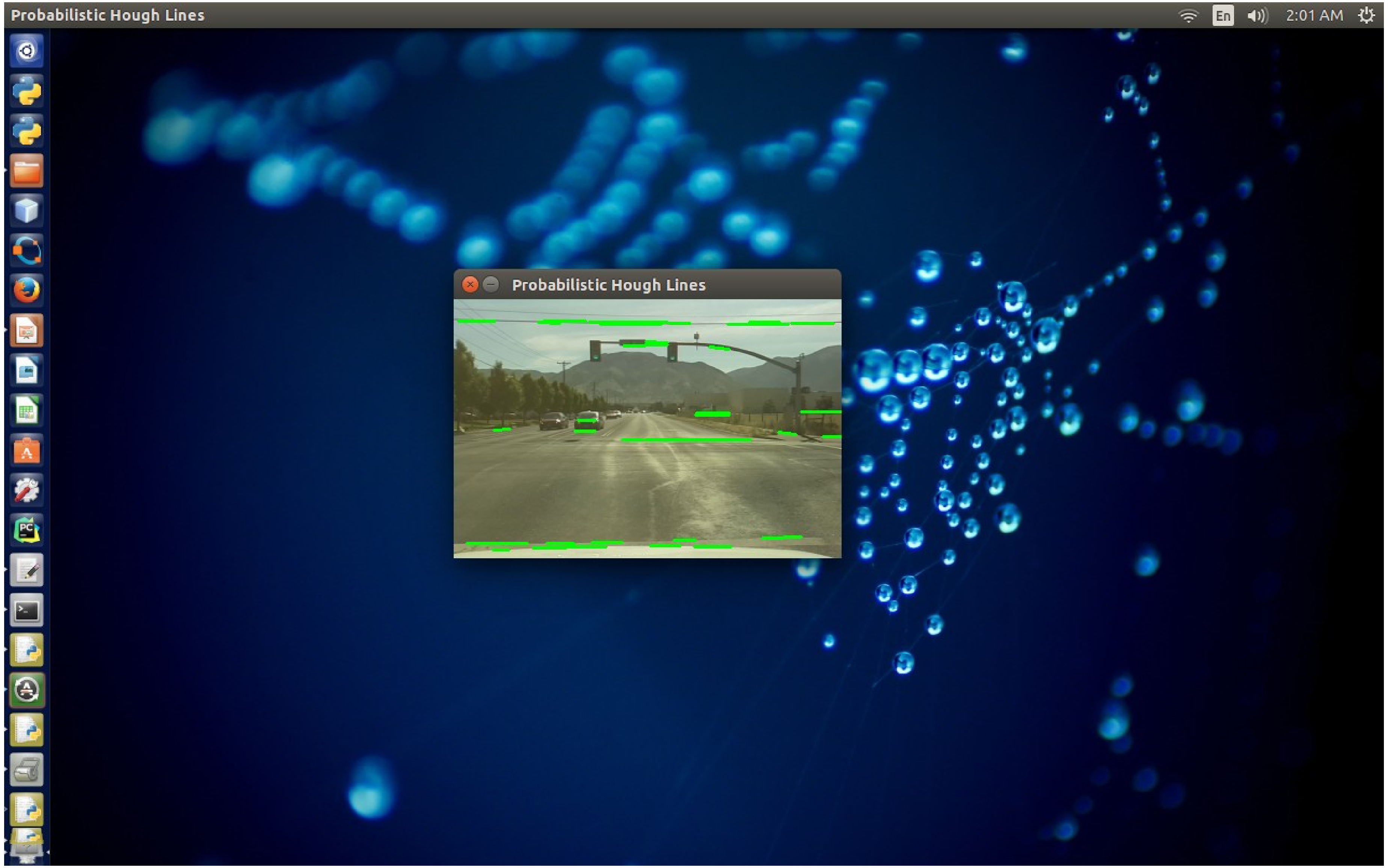
```
$ python prob_houghlines.py -i sudoku.jpg -spl 50
```

# Sample Output



$ python prob_houghlines.py -i 01.png -spl 50

# Sample Output



```
$ python prob_houghlines.py -i 02.png -spl 50
```

# Solution

```
# Let's load the image
image = cv2.imread(args['img'])
# Grayscale and Canny Edges extracted
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray, 100, 170, apertureSize = 3)
# Run HoughLines using a rho accuracy of 1 pixel
# theta accuracy of np.pi / 180 which is 1 degree at
# the user specified support level
lines = cv2.HoughLinesP(edges, 1, np.pi/180, args['spl'], 10, 15)
```

# Solution

```
# iterate through each line and convert it to the format
# required by cv.lines (i.e. requiring end points)
for ln in lines:
    x1, y1, x2, y2 = ln[0]
    cv2.line(image, (x1, y1), (x2, y2), (0, 255, 0), 2)
```

# Observations on Hough Transform

- Thresholds that work in one domain may not (and typically do not) work in a different domain
- While probabilistic HT tends to be more flexible, the detected lines tend to be choppier than with deterministic HT

# References

- http://en.wikipedia.org/wiki/OpenCV
- http://opencv.org/
- R. Laganiere. OpenCV 2 Computer Vision Application Programming Cookbook.