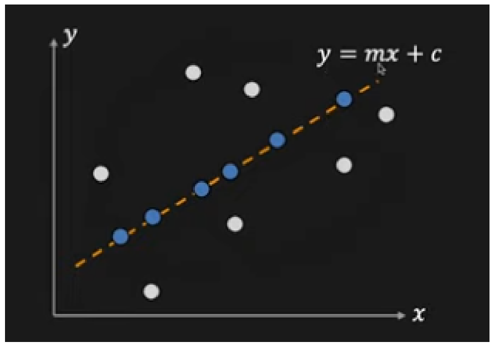
Hough Transform

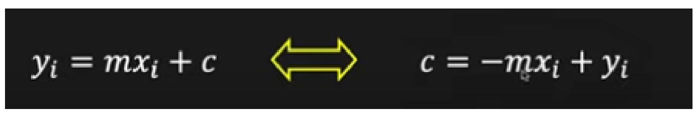
Line Detection:

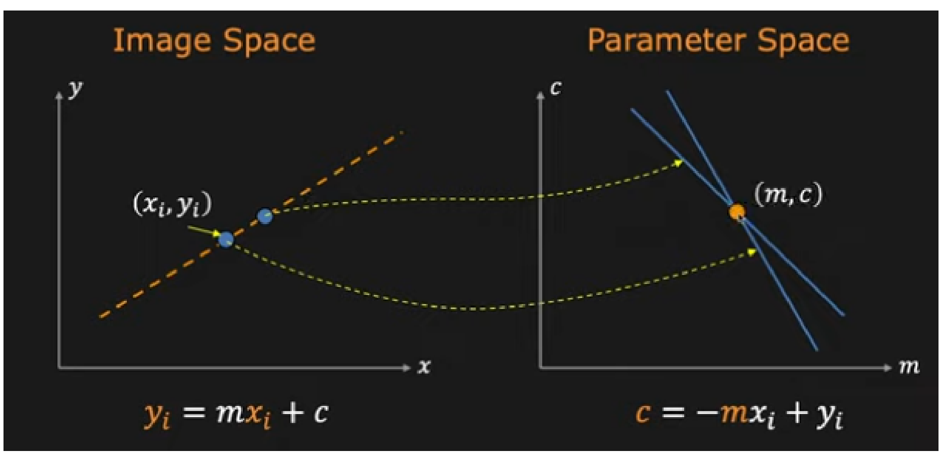
Give a set a edge points the task is to detect a line among these points



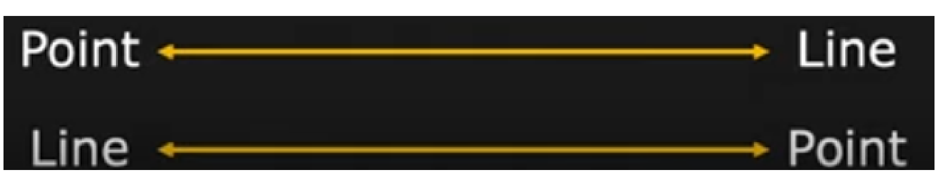
For a point x_i,y_i

can be represented in the following two ways





each edge point in the image space can be represented as a line in the parameter space whose every point corresponds to (m,c) values and a line with each of these (m,c) values passes through x_i,y_i.if we plot lines for every point of image space to parameter space we get a set of lines which intersect each other at a point this point represents a line passing through all points in the image space that lie on the line.



we observe this mapping from image space to parameter space where a point in image space is represented as a line in parameter space and the intersection point of multiple lines in parameter space represents a line in image space.

Line Detection Algorithm

Step 1. Quantize parameter space (m, c)

Step 2. Create accumulator array $A(m, c)$

Step 3. Set $A(m, c) = 0$ for all (m, c)

Step 4. For each edge point (x_i, y_i),

$$A(m, c) = A(m, c) + 1$$

if (m, c) lies on the line: $c = -mx_i + y_i$

and then find the local maxima in the accumulator array

Issues

Issue: Slope of the line $-\infty \leq m \leq \infty$

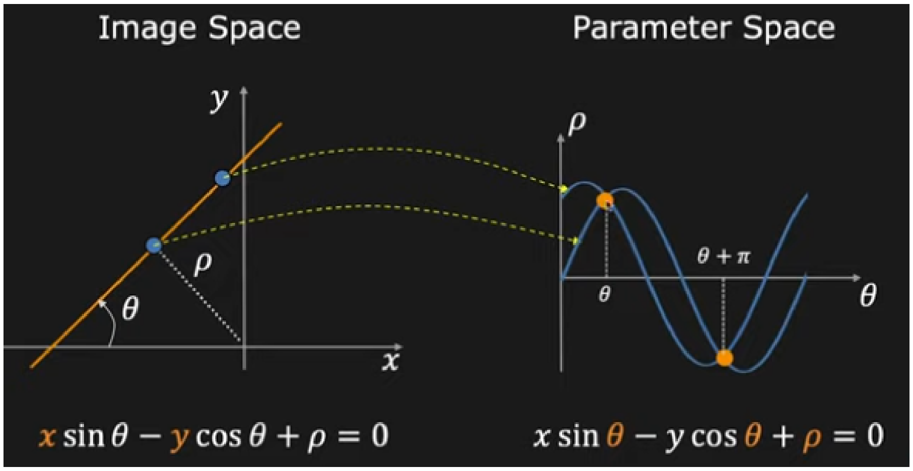
- Large Accumulator
- More Memory and Computation

Solution

Solution: Use $x \sin\theta - y \cos\theta + \rho = 0$

- Orientation θ is finite: $0 \leq \theta < \pi$
- Distance ρ is finite

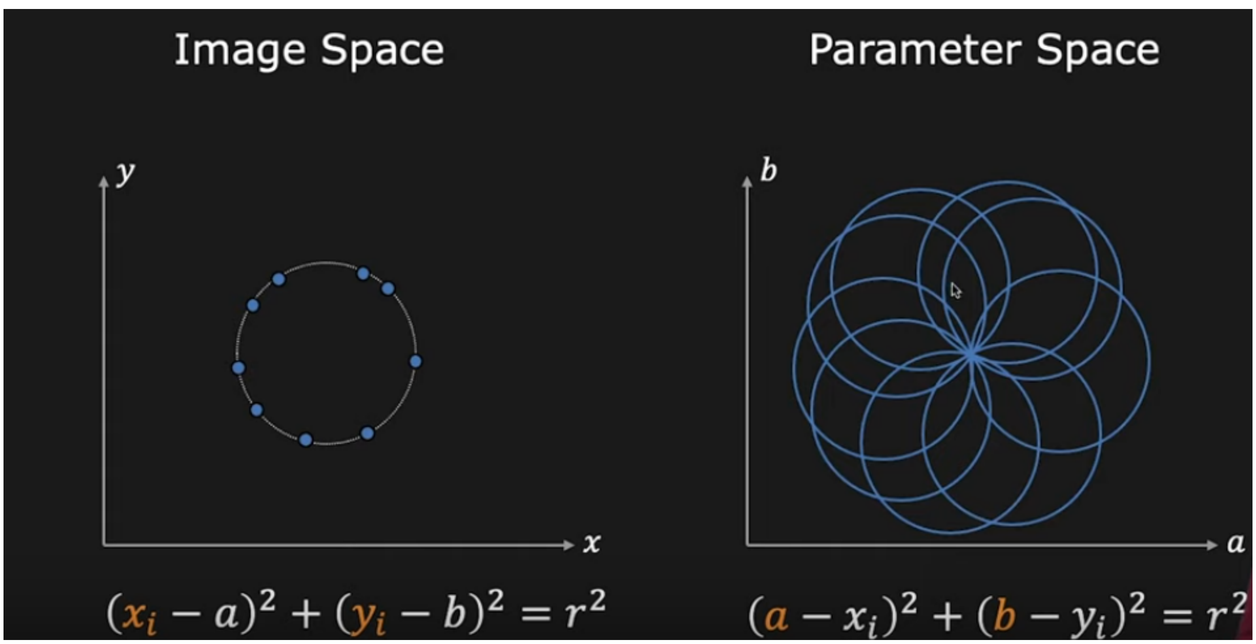
Using (p,theta) parameterization



each point in image space corresponds to a sinusoidal wave in the parameter space. These sinusoidal waves intersect each other at two points but they occur at period of pi i.e. (at theta and theta+pi)so we don't count the second one.

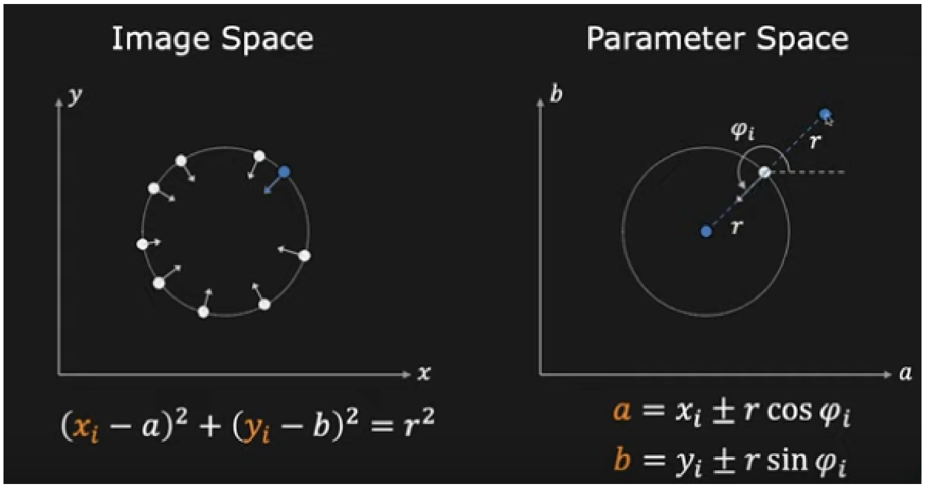
Hough Circle Detection

Considering the case where r is known:



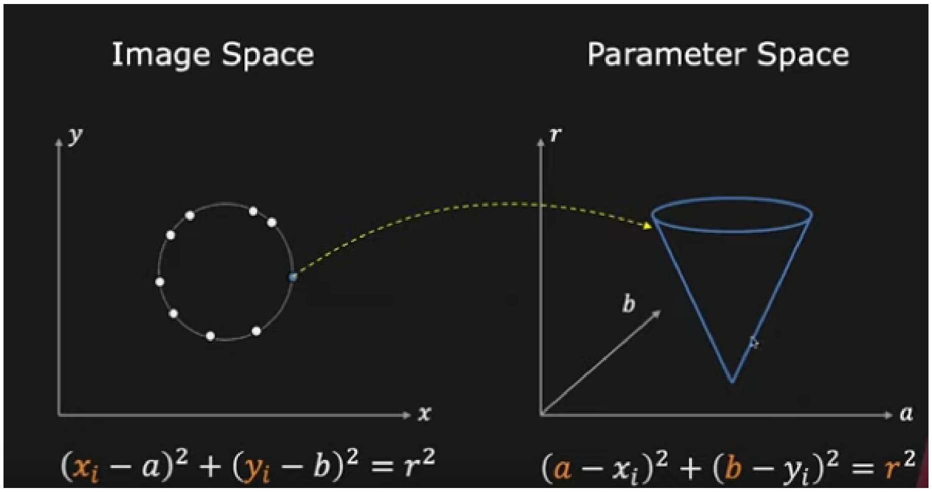
every point in image space can be represented as a circle in parameter space(every value of (a,b) on the circle in parameter space a circle can be drawn to pass through x_i,y_i with the (a,b) value from parameter space and radius r. In this way if we draw the circles in parameter space for points in the image space we get a point of intersection in the parameter space which represents a circle in the image space.

Considering the case where we also have gradient information of the points i.e.(the orientation of the point) along with the radius of the circle.



since we have information about the gradient at each point the center must lie along the gradient at a distance r from the point in the parameter space therefore instead of a circle of possible points we get only two points in the parameter space.

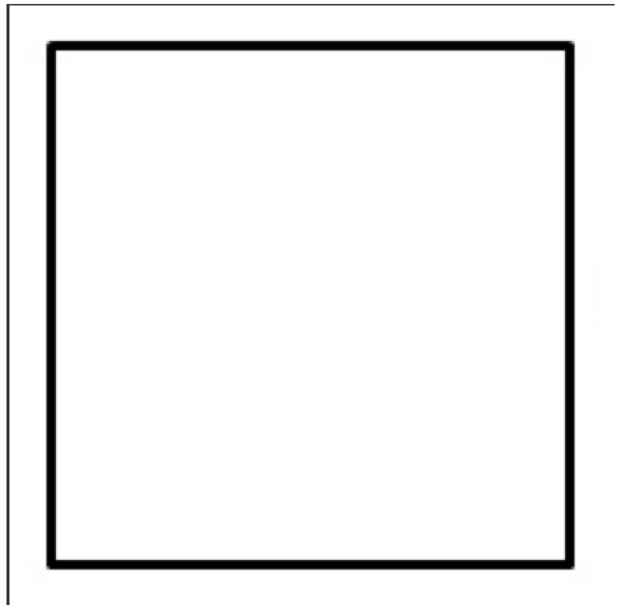
In the case where radius in unknown the locus of each point is a cone in parameter space



Implementation of Hough Line Detection:

```
import cv2
from matplotlib import pyplot as plt
import matplotlib.lines as mlines
import numpy as np
image = cv2.imread('square.png')
height, width, _ = image.shape
rgb_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

plt.imshow(rgb_image)
plt.axis('off')  # Remove the axis labels
plt.show()
```

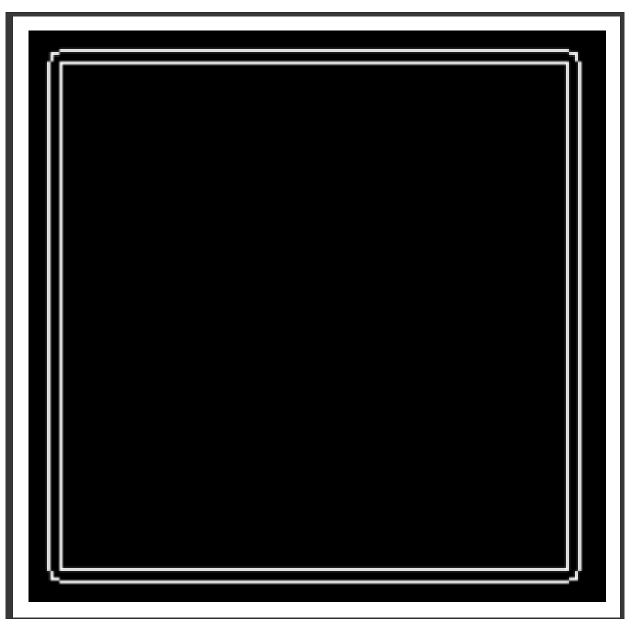


```
image_np = np.array(image)

# Apply Gaussian blur to reduce noise (optional)
blurred = cv2.GaussianBlur(image_np, (5, 5), 0)

# Perform Canny edge detection
edges = cv2.Canny(blurred, threshold1=30, threshold2=100)

plt.imshow(edges, cmap='gray')
plt.axis('off')  # Remove the axis labels
plt.show()
```



```
from PIL import Image

image = Image.open('square.png')
width, height = image.size
resolution = 10  # Desired resolution for the accumulator array
```

```
num_rows = int(np.sqrt(edges.shape[0]**2 + edges.shape[1]**2))  # Adjust the number of rows based on your specific needs
num_columns = 180  # Adjust the number of columns based on the desired resolution of theta
theta = np.linspace(0, np.pi, 180)
accumulator = np.zeros((num_rows, num_columns))

for y in range(edges.shape[0]):
    for x in range(edges.shape[1]):
        if edges[y, x] != 0:  # Check if the pixel is an edge pixel
            for t in range(len(theta)):
                rho = x * np.cos(theta[t]) + y * np.sin(theta[t])
                rho_index = int(rho)
                theta_index = int((theta[t] / np.pi) * (num_columns - 1))
                accumulator[rho_index, theta_index] += 1
```

```
threshold = 100
parameters=[]
xinter=[]
for rho_index in range(accumulator.shape[0]):
    for theta_index in range(accumulator.shape[1]):
        value = accumulator[rho_index, theta_index]

        # Check if the value exceeds the threshold
        if value > threshold:
            # Get the ρ and θ values from the index pair
            rho = rho_index *1
            theta = (theta_index * np.pi) / accumulator.shape[1]

            # Convert ρ and θ to the form of y = ax + b
            a = -np.cos(theta) / np.sin(theta)
            b = rho / np.sin(theta)
            xinter.append(rho/np.cos(theta))
            # Print or store the line parameters (a and b) for further use
            print("Detected line: y = {}x + {}".format(a, b))
            parameters.append([a,b])
parameters_array=np.array(parameters)
print(parameters_array)
```

```
for i in range(len(parameters_array)):
  a = parameters_array[i][0]
  b = parameters_array[i][1]
  if np.isfinite(a) and np.isfinite(b):
    x = np.linspace(0, width, 100)

    # Calculate the y-coordinates using the slope and intercept
    y = a* x + b

    # Plot the line
    plt.plot(x, y, 'r-', label='Line')
  else:
    plt.axvline(x=xinter[i]
                , color='r', linestyle='-', label='Line')

  # Set labels and title
plt.xlabel('X')
plt.ylabel('Y')
```

```
plt.title('Line Plot')

  # Add legend
plt.legend()

  # Show the plot
plt.show()
```