Marker based localization

Line detection

Team members

Niharika Jayanthi Dheeraj Kamath

Under the guidance of
Sanam Shakya

Goals

- Learn how Canny edge detector works.
- Understand concepts of Hough line transform.
- Learn about cv2.Canny() and cv2.HoughLines()

Theory

Canny Edge Detector

Developed by John F. Canny, this edge detection algorithm helps reduce the number of pixels to be processed in an image when extracting information from it. Following are three criteria to be satisfied in edge detection-

1. **Low error rate** - It should detect as many edges as possible and ignore all other non-edges.
2. **Localization of edges** - This means that the detected edge must not be too far away from the actual edge.
3. **Minimum response** - More than one edges should not be detected for one real edge.

Canny edge detection algorithm has five stages, which are explained below-

1. **Smoothing** - This includes filtering out any noise present in the original image. Noise is undesirable as it can be wrongly detected as an edge. Gaussian filter is used for this purpose with a convolution mask that is generally much smaller than the image itself. The detector's sensitivity to noise is inversely proportional to the width of the Gaussian mask.

2. **Finding Gradient** - After removing noise, we find intensity gradient of the image using a Sobel operator in x- and y-directions. After this, edge strengths can be found by applying the Euclidean distance measure. If Gx and Gy are gradients in x and y-directions respectively, then

   edge gradient will be-

   G = squareRoot((Gx ^ 2) + (Gy ^ 2))

   and direction will be -

   angle = invtan(Gy / Gx)

3. **Non-maxima suppression** - This step is included to sharpen the edges which were blurred in the previous steps. This is done by suppressing all gradient values to 0 except the local maxima. This is basically an edge-thinning technique.

4. **Double Thresholding** - To determine potential edges, thresholding is performed. In this, edge pixels with a value higher than the threshold value are marked as strong, those lesser are suppressed and the ones in between as marked as weak. This way, only edges stronger than a certain value are preserved.

5. **Hysteresis Tracking** - This step eliminates weak edges or streaking. Weak edges are those which are not connected to a strong edge. This is performed using BLOB(Binary Large OBject) analysis. In this analysis, a pixel and its 8-connected neighbourhood(BLOB) are analysed. If at least one strong pixel is identified in a BLOB, then it is preserved.

Example

Consider the following image-

On applying Canny edge detection, we get the following result(depicted using trackbar)-

Hough lines transform

The Hough transform is a feature extraction technique.The purpose of the technique is to find imperfect instances of objects within a certain class of shapes

Figure 1: Sunflower

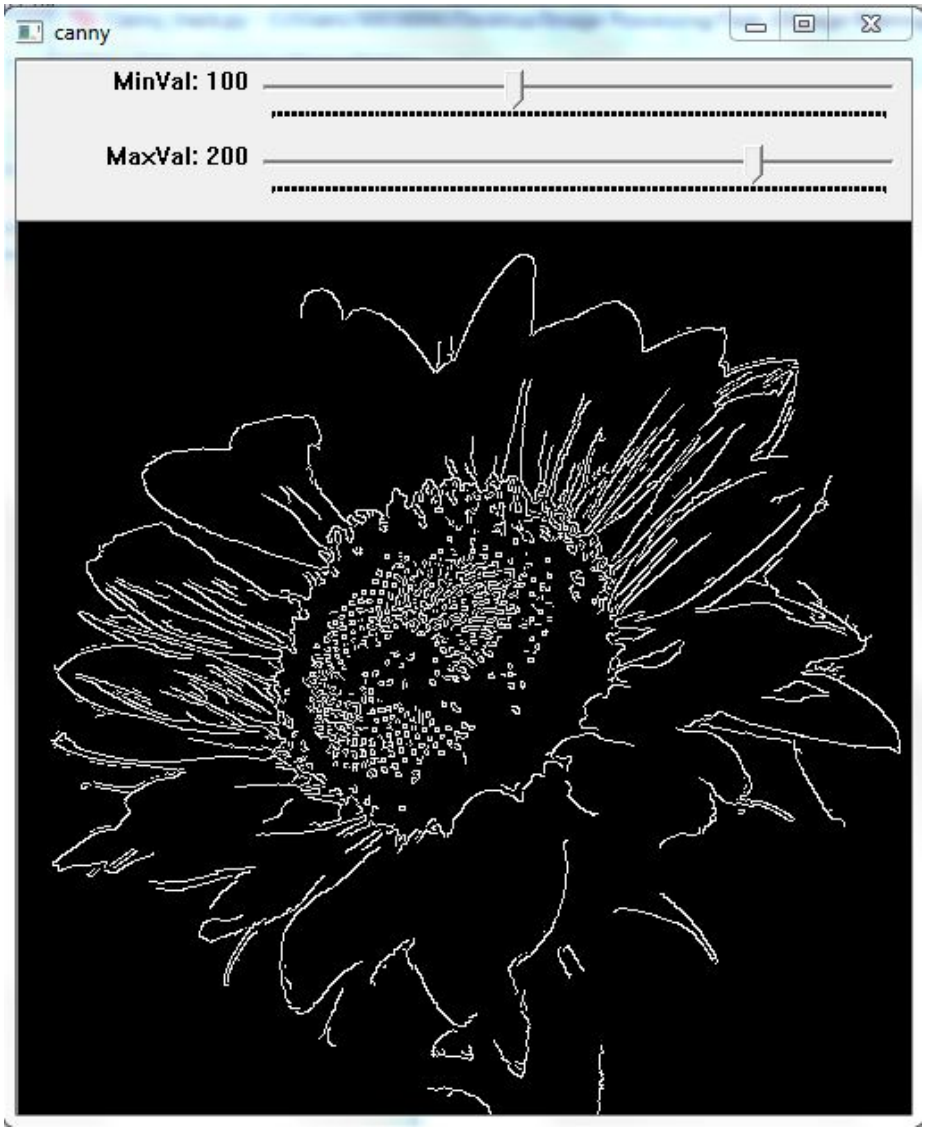

Figure 2: canny

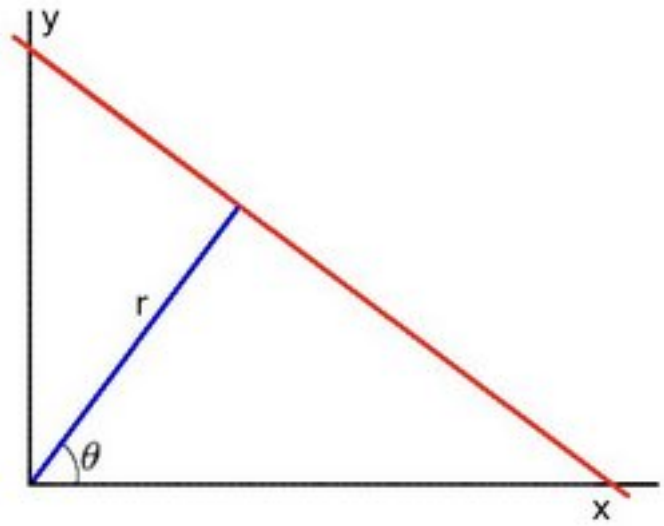

Figure 3: Polar System

by a voting procedure. It means that in general, a line can be detected by finding the number of intersections between curves.

Working principle

A line in image space can be represented in two ways:
Cartesian coordinate system
Parameter coordinate system
Consider the line represented in the polar coordinate system as shown below:

Arranging the terms, we can say in general for each point (x_0, y_0), we can define the family of lines passing through the point as :

$$r_\theta = x_0 \cdot \cos\theta + y_0 \cdot \sin\theta$$

Hence each point (r_θ, θ) represents each line that passes through (x_0, y_0).

Function

`cv2.HoughLines(image, rho, theta, threshold[, lines[, srn[, stn]]])`
* **image** – 8-bit, single-channel binary source image. The image may be modified by the function.
* **lines** – Output vector of lines. Each line is represented by a two-element

vector (r_θ, θ) . ρ is the distance from the coordinate origin (0,0) (top-left corner of the image). θ is the line rotation angle in radians.
* **rho** – Distance resolution of the accumulator in pixels.
* **theta** – Angle resolution of the accumulator in radians.
* **threshold** –Accumulator threshold parameter. Only those lines are returned that get enough votes (>threshold).
* **srn** – For the multi-scale Hough transform, it is a divisor for the distance resolution rho . The coarse accumulator distance resolution is rho and the accurate accumulator resolution is rho/srn . If both srn=0 and stn=0 , the classical Hough transform is used. Otherwise, both these parameters should be positive.
* **stn** – For the multi-scale Hough transform, it is a divisor for the distance resolution theta.

Code

Canny Edge Detection

The main function used is-

```
cv2.Canny(src, threshold1, threshold2[, edges[, apertureSize[, L2gradient]]])
```

Parameters-

- **src**- Single-channel 8-bit input image
- **threshold1**- First threshold for the hysteresis procedure
- **threshold2**- Second threshold for the hysteresis procedure
- **edges**- Output edge map; it has the same size and type as src
- **apertureSize**- aperture size for the Sobel() operator
- **L2gradient**- Flag indicating whether L2 or L1 norm should be used to calculate the image gradient magnitude

We shall now have a look at a simple canny edge detection code.

```
#Import opencv
import cv2

#Read the image
img = cv2.imread('example.jpg')

#Canny edge detect
edges = cv2.Canny(img, 100, 200)
```

```
#Display image
cv2.imshow('Canny', edges)

#Exit
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Hough Line Transform

```
 #Import modules
 import cv2
 import numpy as np
 import matplotlib.pyplot as plt

 #Read the image
 img = cv2.imread('road3.jpg')

 gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
 edges = cv2.Canny(gray,220,150,apertureSize = 3)


 lines = cv2.HoughLines(edges,1,np.pi/180,90,10)

 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)

cv2.imshow('canny',edges)
cv2.imshow('Hough',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

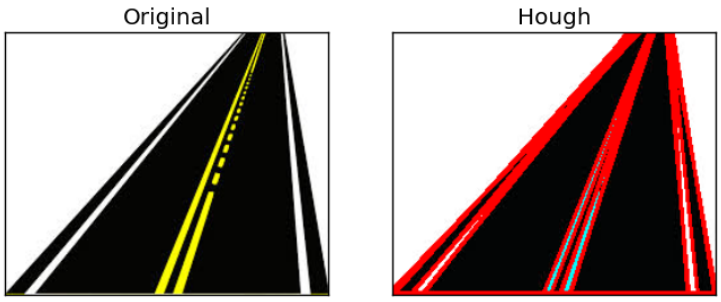

Figure 4: Output Image

Resources

- cse.iitd - pdf
- Canny edge detection
- Canny - opencv-python tutorials
- Canny edge detector - Wiki
- Opencv docs