دانشگاه شهید چمران اهواز

کلاس برنامه نویسی پایتون

موضوع: الگوریتم Harris وتبدیل Hough

ارائه دهنده: مرجان مودت

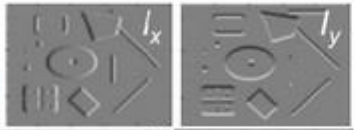
Harris Corner Detector

تشخیص گوشه با الگوریتم هریس

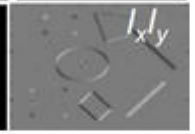
matrix

$$\mu(\sigma_I,\sigma_D) = g(\sigma_I) * \begin{bmatrix} I_x^2(\sigma_D) & I_x I_y(\sigma_D) \\ I_x I_y(\sigma_D) & I_y^2(\sigma_D) \end{bmatrix}$$

1. Image derivatives (optionally, blur first)



2. Square of derivatives



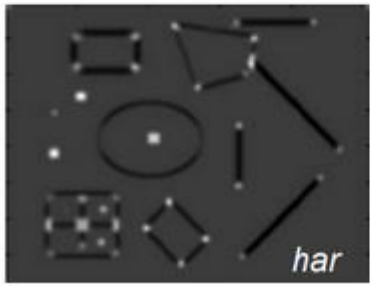
$$\det M = \lambda_1\lambda_2$$
$$\text{trace } M = \lambda_1 + \lambda_2$$

3. Gaussian filter $g(\sigma_I)$

4. Cornerness function – both eigenvalues are strong

$$har = \det[\mu(\sigma_I,\sigma_D)] - \alpha[\text{trace}(\mu(\sigma_I,\sigma_D))^2] =$$
$$g(I_x^2)g(I_y^2) - [g(I_x I_y)]^2 - \alpha[g(I_x^2) + g(I_y^2)]^2$$

5. Non-maxima suppression



Harris Corner Detector

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
```

Sobel and gaussian kernels defined here with numpy

```
# Sobel x-axis kernel
SOBEL_X = np.array((
    [-1, 0, 1],
    [-2, 0, 2],
    [-1, 0, 1]), dtype="int32")

# Sobel y-axis kernel
SOBEL_Y = np.array((
    [-1, -2, -1],
    [0, 0, 0],
    [1, 2, 1]), dtype="int32")

# Gaussian kernel
GAUSS = np.array((
    [1/16, 2/16, 1/16],
    [2/16, 4/16, 2/16],
    [1/16, 2/16, 1/16]), dtype="float64")
```

Convolution function from scratch. Alternatively, you can use opencv, skimage, pillow libraries.

```
def convolve(img, kernel):
    if kernel.shape[0] % 2 != 1 or kernel.shape[1] % 2 != 1:
        raise ValueError("Only odd dimensions on filter supported")

    img_height = img.shape[0]
    img_width = img.shape[1]
    pad_height = kernel.shape[0] // 2
    pad_width = kernel.shape[1] // 2

    pad = ((pad_height, pad_height), (pad_height, pad_width))
    g = np.empty(img.shape, dtype=np.float64)
    img = np.pad(img, pad, mode='constant', constant_values=0)
    # Do convolution
    for i in np.arange(pad_height, img_height+pad_height):
        for j in np.arange(pad_width, img_width+pad_width):
            roi = img[i - pad_height:i + pad_height +1, j - pad_width:j + pad_width + 1]
            g[i - pad_height, j - pad_width] = (roi*kernel).sum()

    if (g.dtype == np.float64):
        kernel = kernel / 255.0
        kernel = (kernel*255).astype(np.uint8)
    else:
        g = g + abs(np.amin(g))
        g = g / np.amax(g)
        g = (g*255.0)
    return g
```

Harris function

```
def harris(img, threshold=0.6):

    img_cpy = img.copy() # copying image
    img1_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # grayscaling (0-1)

    dx = convolve(img1_gray, SOBEL_X) # convolving with sobel filter on X-axis
    dy = convolve(img1_gray, SOBEL_Y) # convolving with sobel filter on Y-axis
    # square of derivatives
    dx2 = np.square(dx)
    dy2 = np.square(dy)
    dxdy = dx*dy #cross filtering
    # gauss filter for all directions (x,y,cross axis)
    g_dx2 = convolve(dx2, GAUSS)
    g_dy2 = convolve(dy2, GAUSS)
    g_dxdy = convolve(dxdy, GAUSS)
    # Harris Function
    harris = g_dx2*g_dy2 - np.square(g_dxdy) - 0.12*np.square(g_dx2 + g_dy2) # r(harris) = det -
k*(trace**2)
    # Normalizing inside (0-1)
    cv2.normalize(harris, harris, 0, 1, cv2.NORM_MINMAX)

    # find all points above threshold (nonmax supression line)
    loc = np.where(harris >= threshold)
    # drawing filtered points
    for pt in zip(*loc[::-1]):
        cv2.circle(img_cpy, pt, 3, (0, 0, 255), -1)

    return img_cpy,g_dx2,g_dy2,dx,dy,loc
```

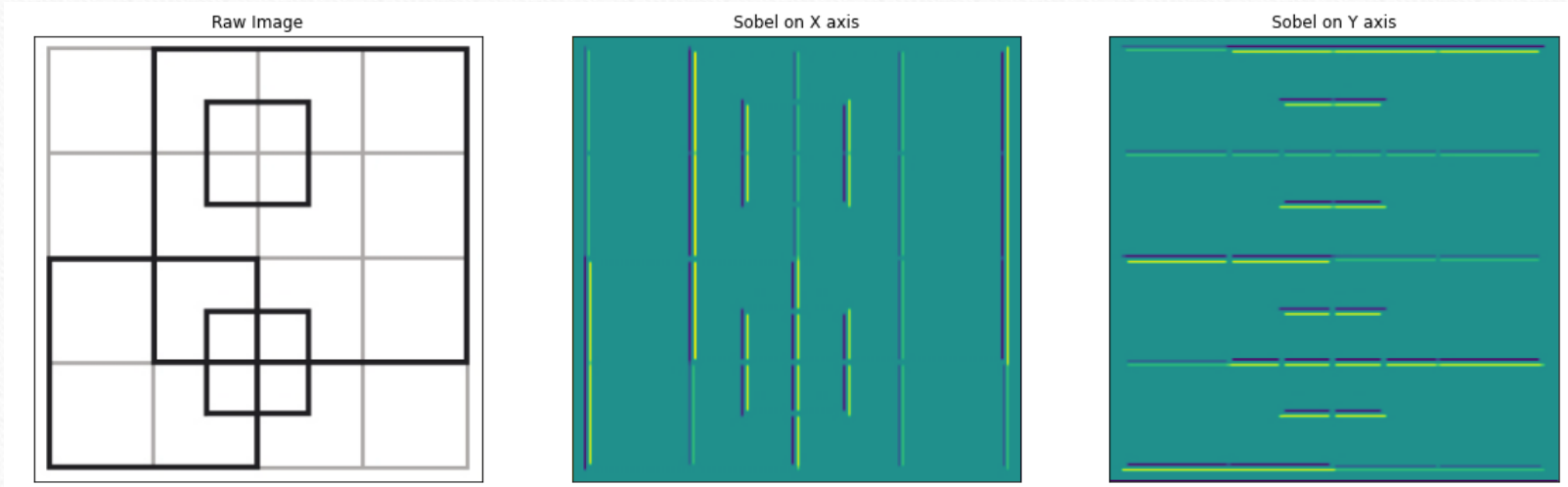
$$har = \det[\mu(\sigma_I,\sigma_D)] - \alpha[\text{trace}(\mu(\sigma_I,\sigma_D))^2] =$$
$$g(I_x^2)g(I_y^2) - [g(I_xI_y)]^2 - \alpha[g(I_x^2) + g(I_y^2)]^2$$

گوشه ها را آبی کن

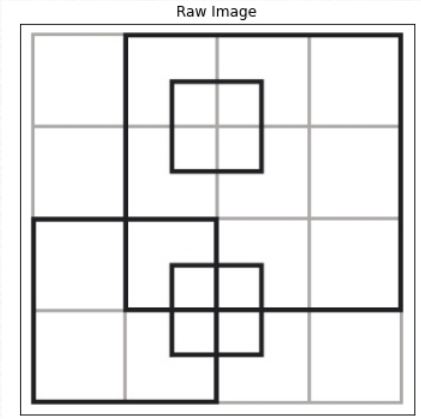
Image Reading and Filtering Area

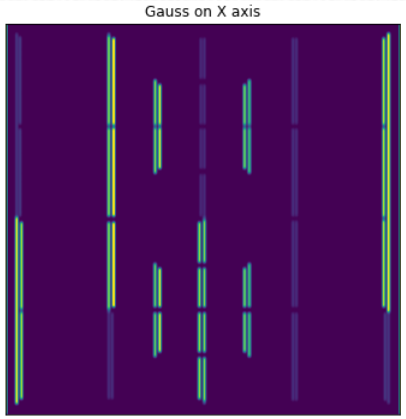
```
img = cv2.imread('/content/drive/MyDrive/box.PNG')
#img = cv2.imread('/content/drive/MyDrive/chess.png')

corners,g_dx2,g_dy2,dx,dy,loc = harris(img,0.7)
                                          threshold
```

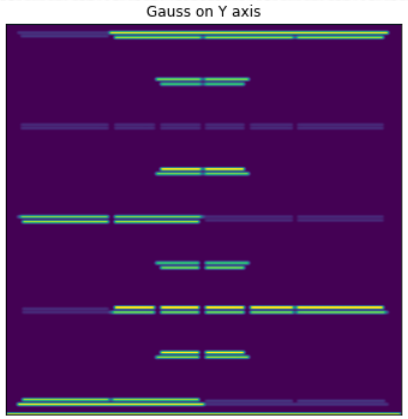
Raw Image and Sobel Filtered Images Visualization



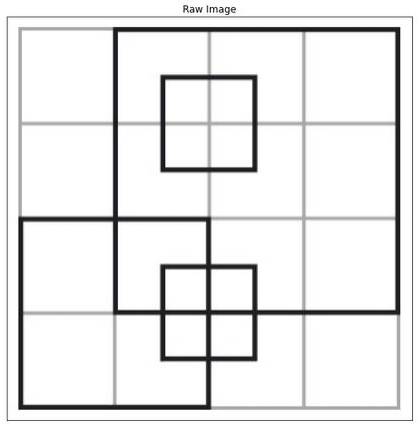
Raw Image and Sobel-Gauss Filtered Images Visualization

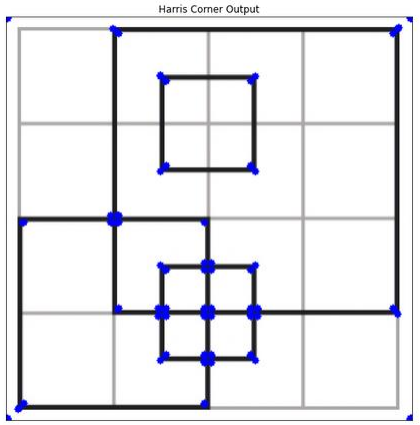






Raw Image and Harris Corner Output Visualization ✓



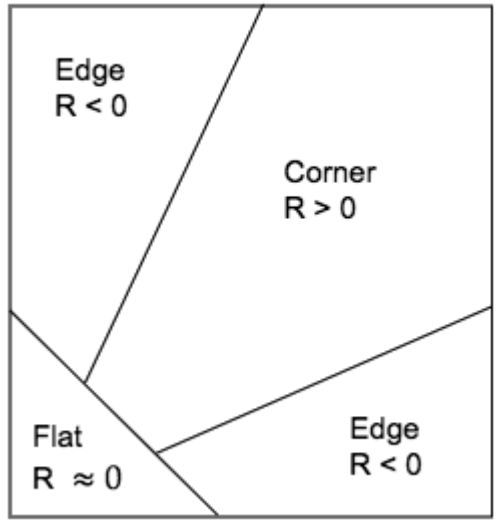


۱.وقتی |R| کوچک است، که زمانی اتفاق می افتد که λ1و λ2کوچک هستند، منطقه مسطح است.
۲.وقتی R<0، که در λ1>>λ2یا بالعکس اتفاق می افتد، منطقه یک یال است.
۳.وقتی Rبزرگ باشد، زمانی که λ1و λ2بزرگ و λ1~λ2 باشند، این ناحیه یک گوشه است.

$$R = \det M - k(trace\, M)^2$$

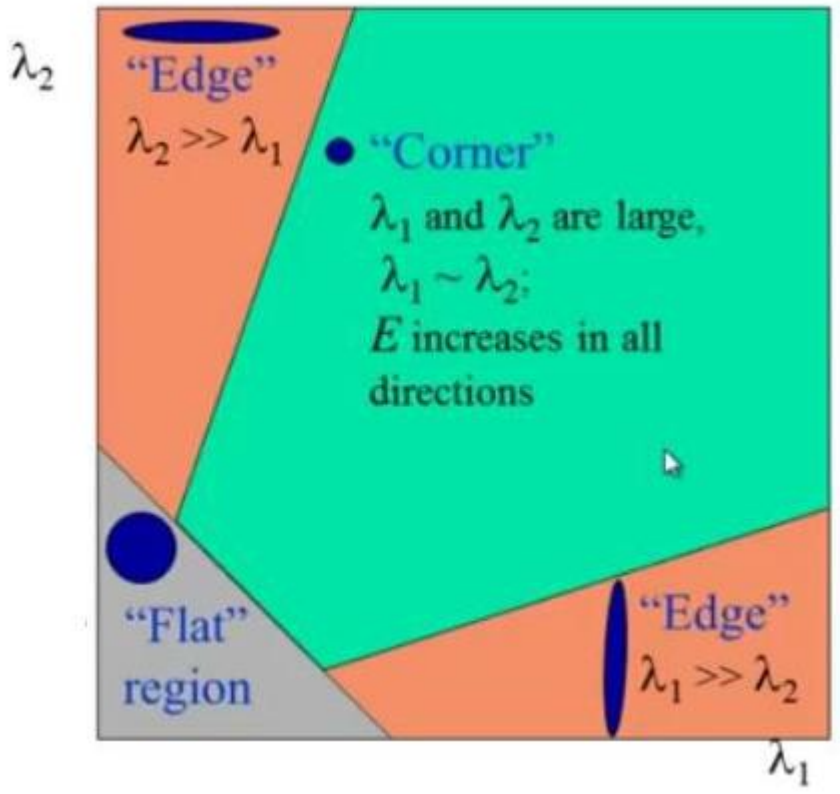
$$\det M = \lambda_1 \lambda_2$$

$$trace\, M = \lambda_1 + \lambda_2$$



$$A = I_x I_x \oplus w$$
$$B = I_y I_y \oplus w$$
$$C = I_x I_y \oplus w$$

$$det(M) = AB - C^2 \; or \; \lambda_1 \lambda_2$$
$$trace(M) = A + B \; or \; \lambda_1 + \lambda_2$$

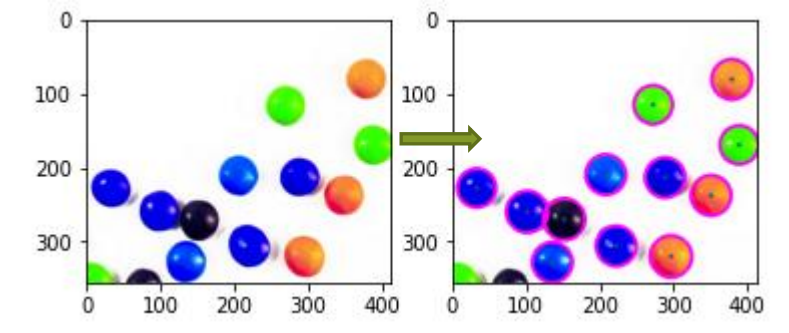


تبدیل هاف(Hough transform)

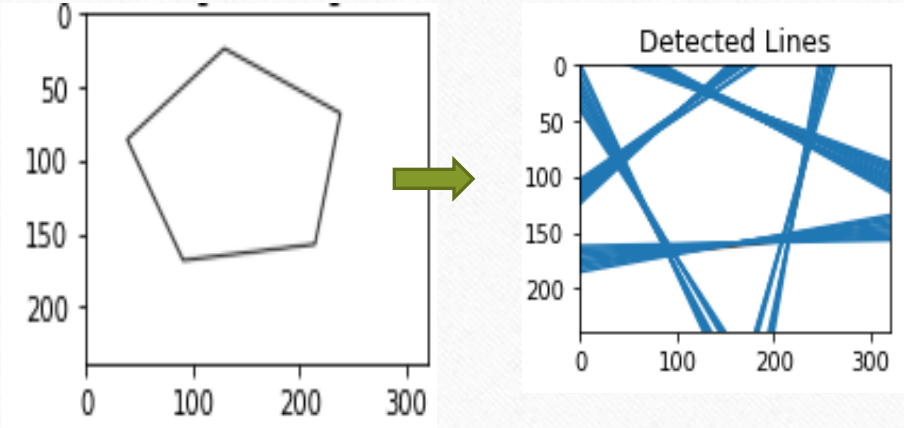
- روشی برای استخراج ویژگی‌ها در آنالیز تصاویر، بینایی رایانه‌ای و پردازش تصویر دیجیتال است.
- این روش در یک تصویر به دنبال نمونه‌هایی از یک الگو می‌گردد. این نمونه‌ها ممکن است کامل نباشند و همچنین تا حدی دچار اعوجاج شده باشند.
- به عنوان نمونه از کاربردهای این روش می‌توان به تشخیص وجود خط مستقیم در یک تصویر و اشکال دایره‌ای و... اشاره کرد.

مثال تشخیص خط مستقیم و دایره در تصویر با تبدیل هاف

- Circle Hough Transform



- straight line Hough Transform



Lines Detection with Hough Transform:

Extract edges of the image How ? using **Canny ,…**

1- initialize parameter space **rs, thetas**

2- Create **accumulator** array and initialize to **zero**

3- for each edge pixel

4- for each theta

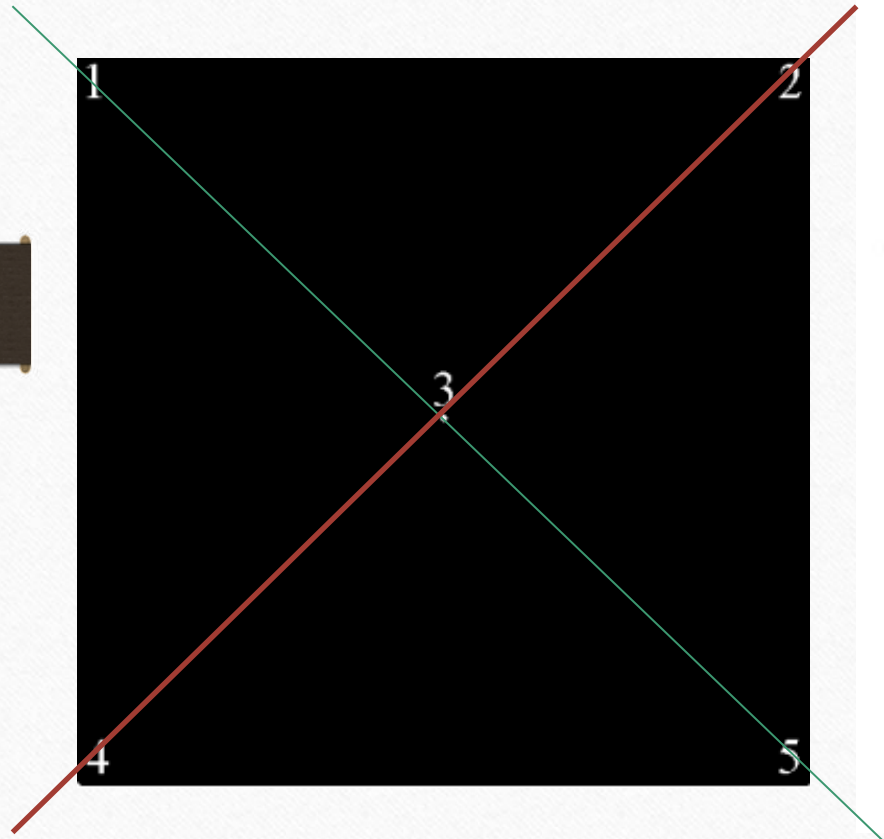
5- calculate r = x cos(theta) + y sin(theta)

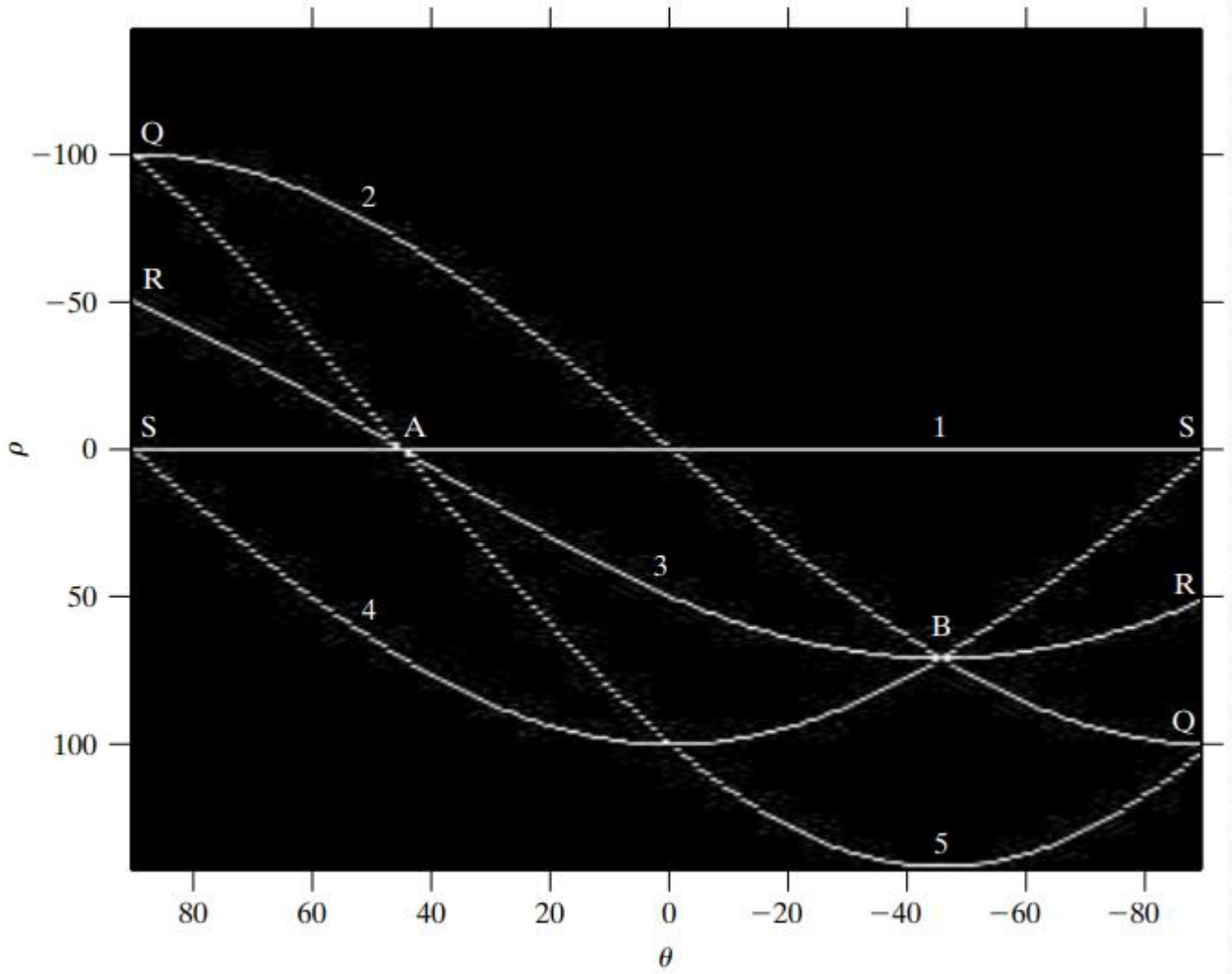
6- Increment accumulator at r, theta

7- Find **Maximum values** in **accumulator** (lines)

Extract **related r, theta**

دقت هم راستایی

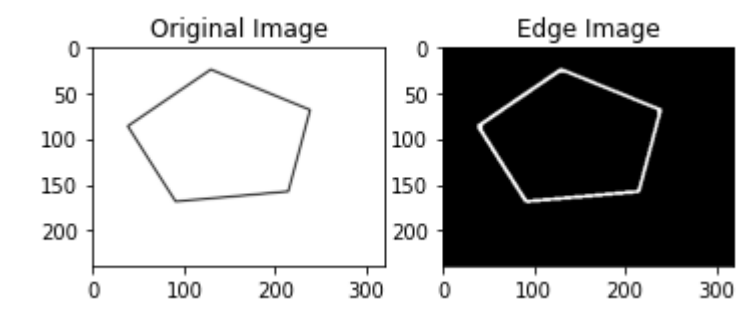




Lines Detection with Hough Transform

Non-Vectorized Solution

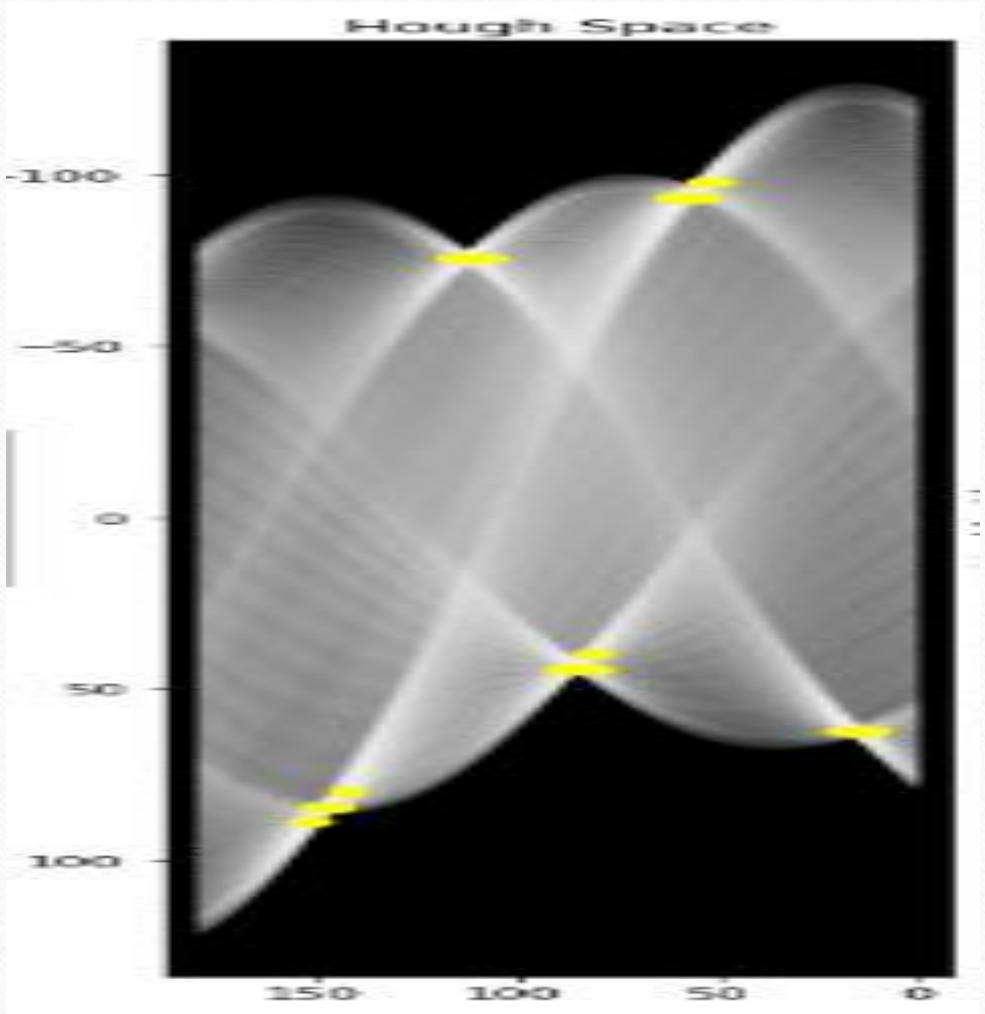
```
image = cv2.imread('/content/drive/MyDrive/pentagon.PNG')
edge_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
edge_image = cv2.GaussianBlur(edge_image, (3, 3), 1)
edge_image = cv2.Canny(edge_image, 100, 200)
edge_image = cv2.dilate(
    edge_image,
    cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5)),
)
edge_image = cv2.erode(
    edge_image,
    cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5)),
)
line_detection_non_vectorized(image, edge_image)
```

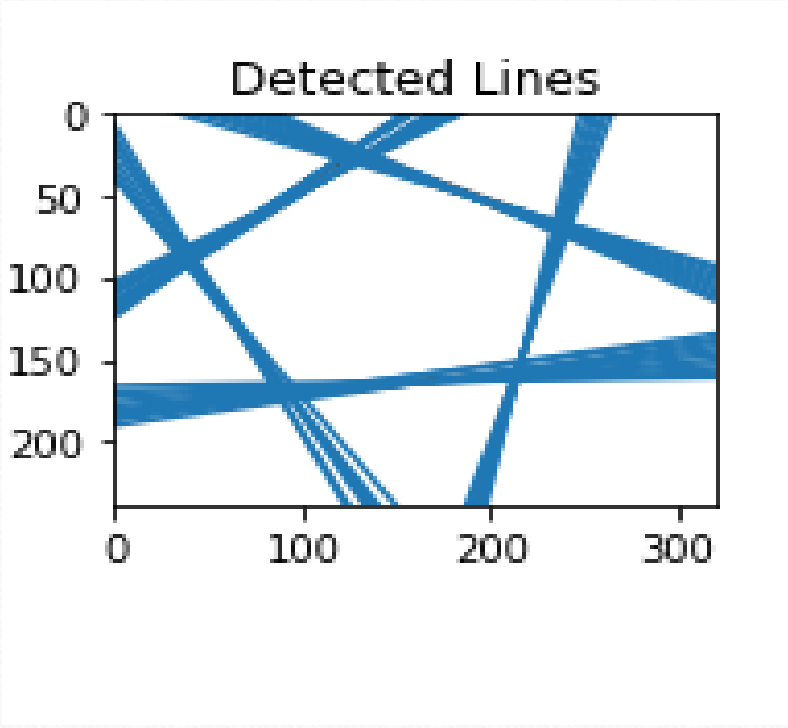


```
def line_detection_non_vectorized(image, edge_image, num_rhos=180, num
_thetas=180, t_count=220):

    edge_height, edge_width = edge_image.shape[:2]
    edge_height_half, edge_width_half = edge_height / 2, edge_width / 2
    #
    d = np.sqrt(np.square(edge_height) + np.square(edge_width))
    dtheta = 180 / num_thetas
    drho = (2 * d) / num_rhos
    #
    thetas = np.arange(0, 180, step=dtheta)
    rhos = np.arange(-d, d, step=drho)
    #
    cos_thetas = np.cos(np.deg2rad(thetas))
    sin_thetas = np.sin(np.deg2rad(thetas))
    #
    accumulator = np.zeros((len(rhos), len(rhos)))
```

```
  for y in range(edge_height):
     for x in range(edge_width):
       if edge_image[y][x] != 0:
         edge_point = [y - edge_height_half, x - edge_width_half]
         ys, xs = [], []
         for theta_idx in range(len(thetas)):
           rho = (edge_point[1] * cos_thetas[theta_idx]) + (edge_point[0] * si
n_thetas[theta_idx])
           theta = thetas[theta_idx]
           rho_idx = np.argmin(np.abs(rhos - rho))
           accumulator[rho_idx][theta_idx] += 1
           ys.append(rho)
           xs.append(theta)
         subplot3.plot(xs, ys, color="white", alpha=0.05)
```

```
for y in range(accumulator.shape[0]):
  for x in range(accumulator.shape[1]):
    if accumulator[y][x] > t_count:
      rho = rhos[y]
      theta = thetas[x]
      a = np.cos(np.deg2rad(theta))
      b = np.sin(np.deg2rad(theta))
      x0 = (a * rho) + edge_width_half
      y0 = (b * rho) + edge_height_half
      x1 = int(x0 + 1000 * (-b))
      y1 = int(y0 + 1000 * (a))
      x2 = int(x0 - 1000 * (-b))
      y2 = int(y0 - 1000 * (a))
      subplot3.plot([theta], [rho], marker='o', color="yellow")
      subplot4.add_line(mlines.Line2D([x1, x2], [y1, y2]))
```


Hough Space
-100
-50
0
50
100
150
100
50
0


Detected Lines
0
50
100
150
200
0
100
200
300

دقت تشخیص خط

- تمام سلول‌های انباشتگر که بالاتر از یک حد آستانه هستند را انتخاب کنیم تا خطوط تصویر را پیدا کنیم.
- میزان آستانه بالاتر باشد، تعداد خطوط قوی کمتری
- هر چه آستانه را مقداری پایین‌تر قرار دهیم، در این صورت تعداد خطوط بیشتری را به دست می‌آوریم که حتی شامل خطوط ضعیف نیز هستند.

تشخیص دایره با تبدیل هاف

برای تشخیص دایره با تبدیل هاف به **انباشتگرسه بعدی** نیاز داریم که هر بعد انباشتگر متعلق به یکی از پارامترهای x0 وy0 و r است. معادله یک دایره را می‌توان به صورت زیر بیان کرد:

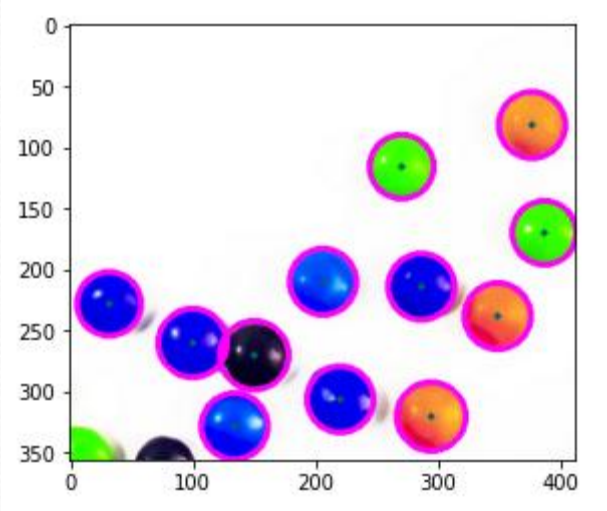
$$(x–x0)2+(y–y0)2=r2$$

برای تشخیص دایره در یک تصویر گام‌های زیر لازم هستند:

1. در تصویر مورد نظر، **لبه** را با استفاده از تشخیص‌گرهای لبه مانند **Canny** شناسایی کنید.
2. برای تشخیص دایره در یک تصویر، باید یک **حد آستانه را برای مقادیر کمینه و بیشینه شعاع** در نظر بگیرید.
3. **دایره‌های با مرکز و شعاع مختلف**، در یک آرایه انباشتگر سه بعدی جمع‌آوری می‌شوند.

- کیفیت نهایی خروجی و تشخیص، تا حد زیادی به **کیفیت تشخیص لبه** بستگی دارد. همچنین عامل مهم دیگر در کیفیت خروجی این است که **چه مقدار دانش پیشین درباره اندازه دایره‌ای در اختیار است که قصد تشخیص آن در تصویر را داریم**.

```
import sys
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

# Loads an image
src = cv.imread('/content/drive/MyDrive/smarties.png')
gray = cv.cvtColor(src, cv.COLOR_BGR2GRAY)
gray = cv.medianBlur(gray, 5)

rows = gray.shape[0]
circles = cv.HoughCircles(gray, cv.HOUGH_GRADIENT, 1, rows / 8, param1=100, param2=30,
                                                                    minRadius=1, maxRadius=30)

if circles is not None:
    circles = np.uint16(np.around(circles))
    for i in circles[0, :]:
      center = (i[0], i[1])
      # circle center
      cv.circle(src, center, 1, (0, 100, 100), 3)
      # circle outline
      radius = i[2]
      cv.circle(src, center, radius, (255, 0, 255), 3)

    plt.imshow( src)
    plt.show()
```



دانشگاه شهید چمران اهواز

سپاس از توجه شما دوستان