Section 4

Hough Transform and Harris Operator

Presentation by *Asem Alaa*

Hough Transform

Proposed by Paul V.C Hough 1962

- Got USA Patent
- Originally for line detection
- Extended to detect other shapes like, circle, ellipse etc.

Hough Transform: Line Detection (Cartesian Coordinates)

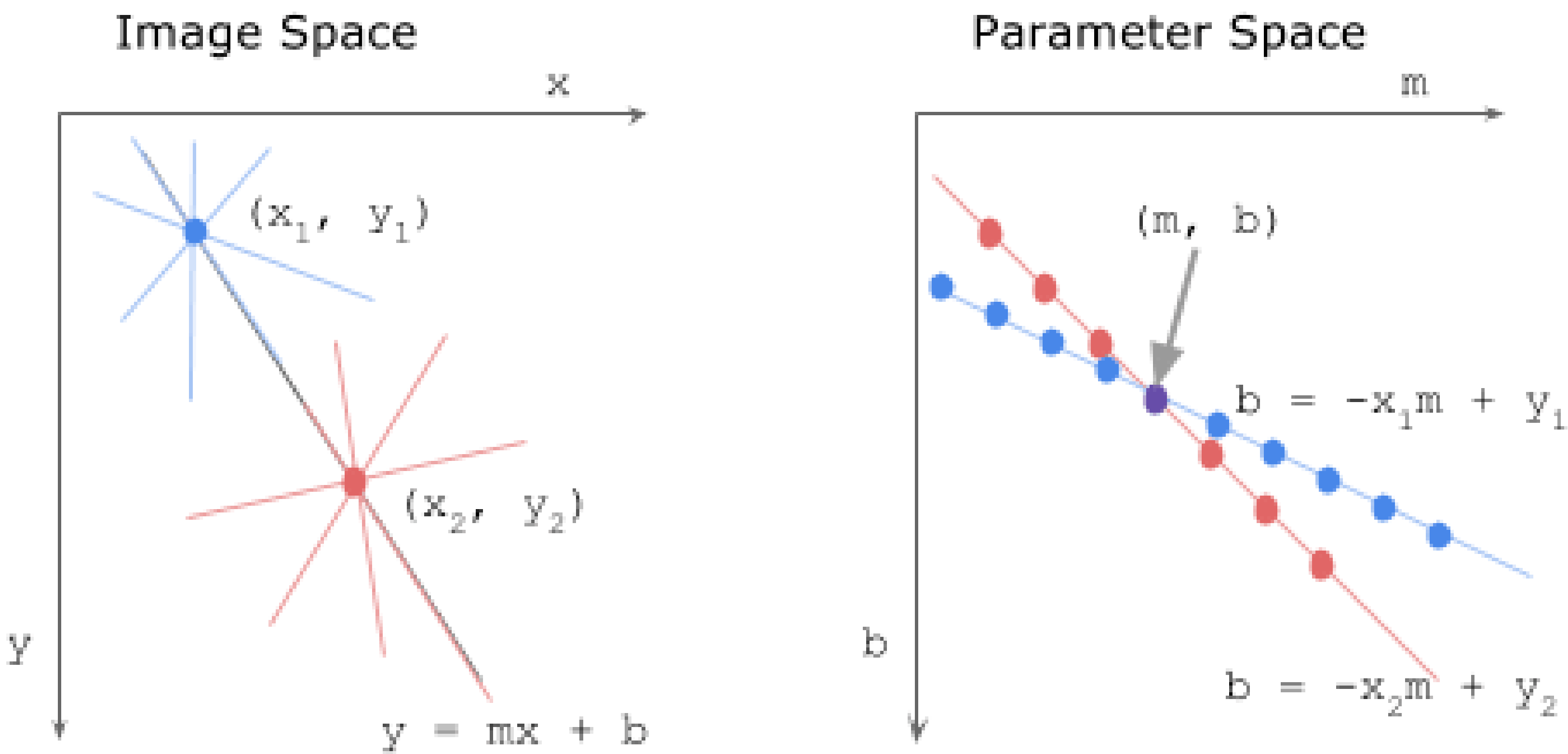
In image space line is defined by the slope m and the y-intercept b :

$$y \quad mx \quad b$$

Hough Transform: Line Detection (Cartesian Coordinates)

In image space line is defined by the slope m and the y-intercept b :

$$y = mx + b$$



Hough Transform: Line Detection (Cartesian Coordinates)

Hough Transform: Line Detection (Cartesian Coordinates)

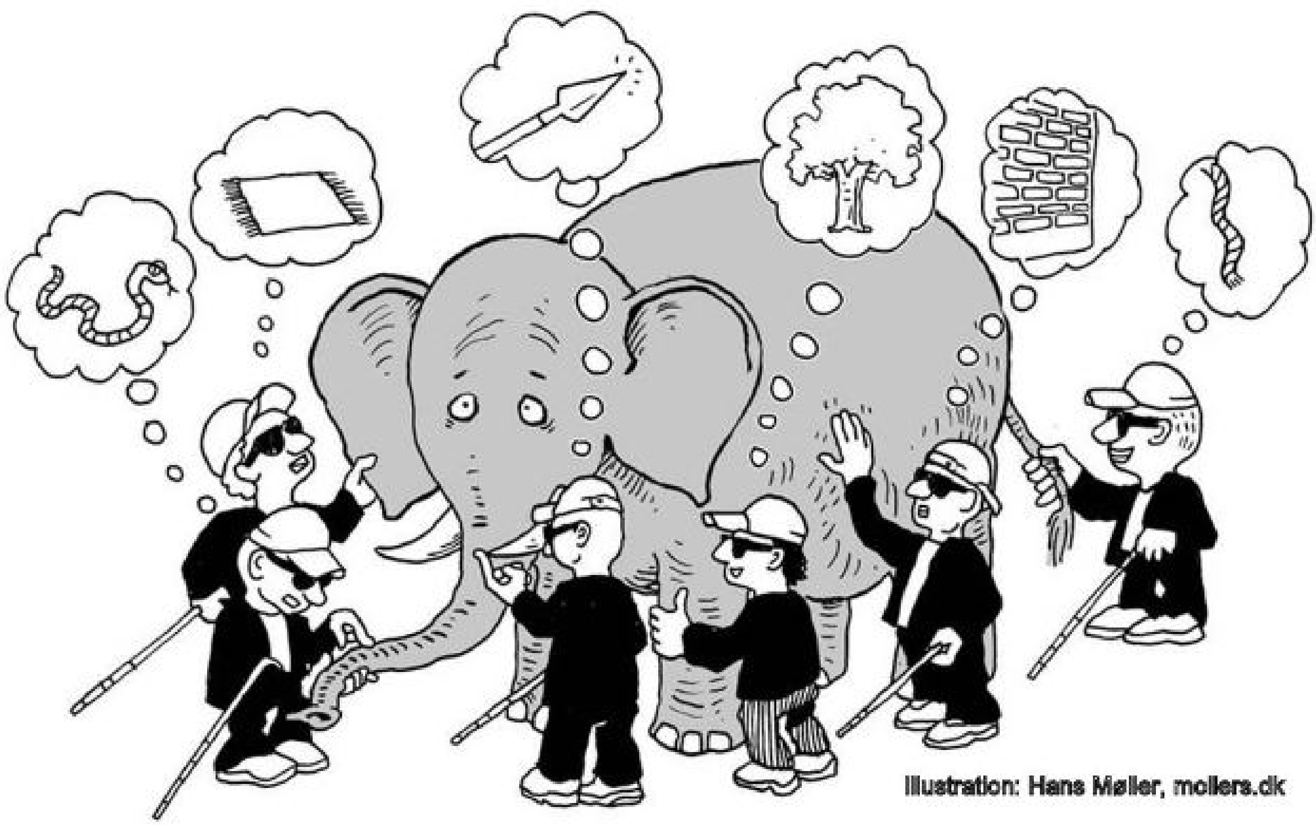
- Each point proposes list of candidate lines

Hough Transform: Line Detection (Cartesian Coordinates)

- Each point proposes list of candidate lines
- Overall, how to find the true lines?

Hough Transform: Line Detection (Cartesian Coordinates)

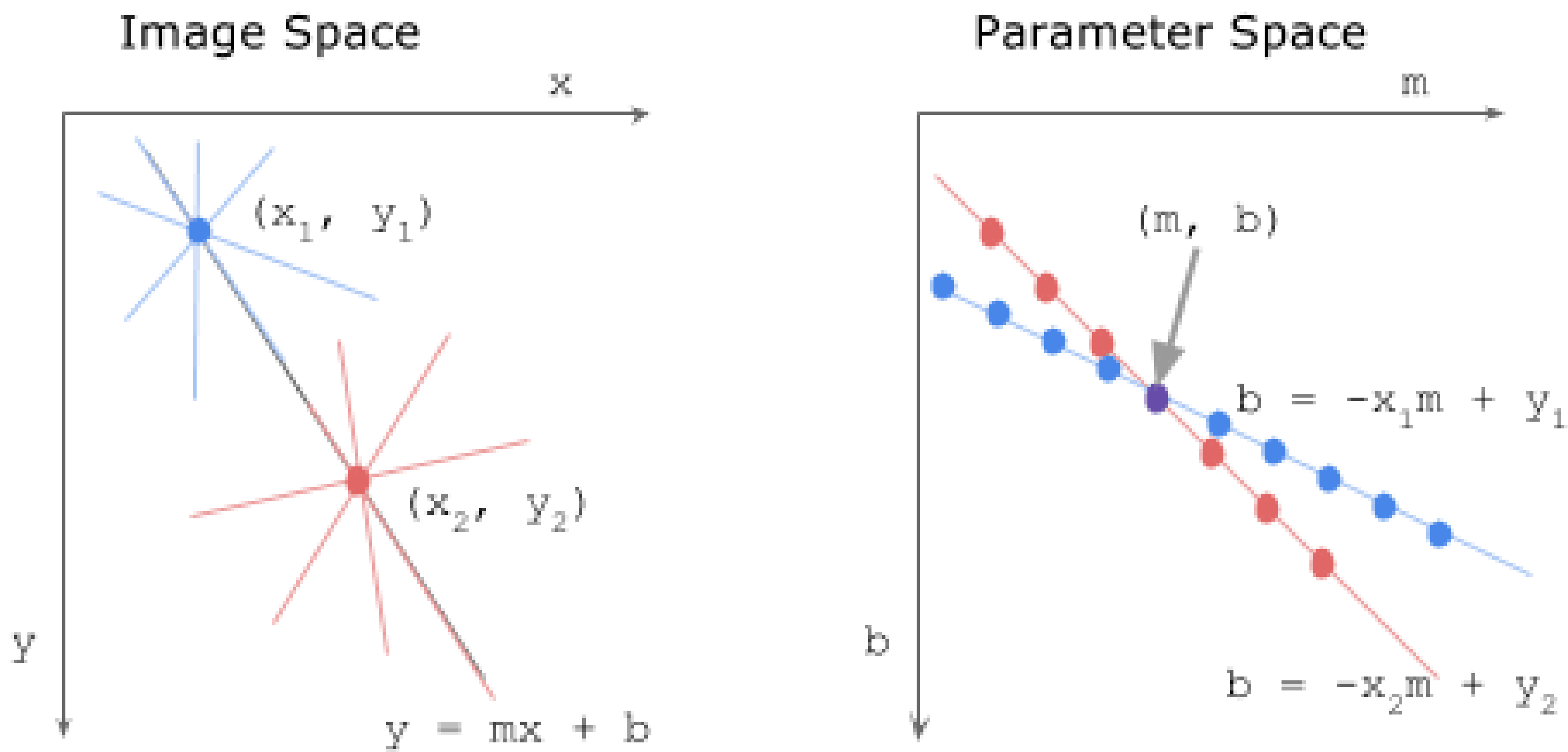
- Each point proposes list of candidate lines
- Overall, how to find the true lines?



Hough Transform: Line Detection (Cartesian Coordinates)

In image space line is defined by the slope m and the y-intercept b :

$$y = mx + b$$



Hough Transform: Line Detection (Polar Coordinates)

Hough Transform: Line Detection (Polar Coordinates)

- Some lines cannot be be defined in Cartesian

Hough Transform: Line Detection (Polar Coordinates)

- Some lines cannot be be defined in Cartesian
- So we have to move to polar coordinates.

Hough Transform: Line Detection (Polar Coordinates)

- Some lines cannot be be defined in Cartesian
- So we have to move to polar coordinates.
- In polar coordinates line is define by ρ and θ

Hough Transform: Line Detection (Polar Coordinates)

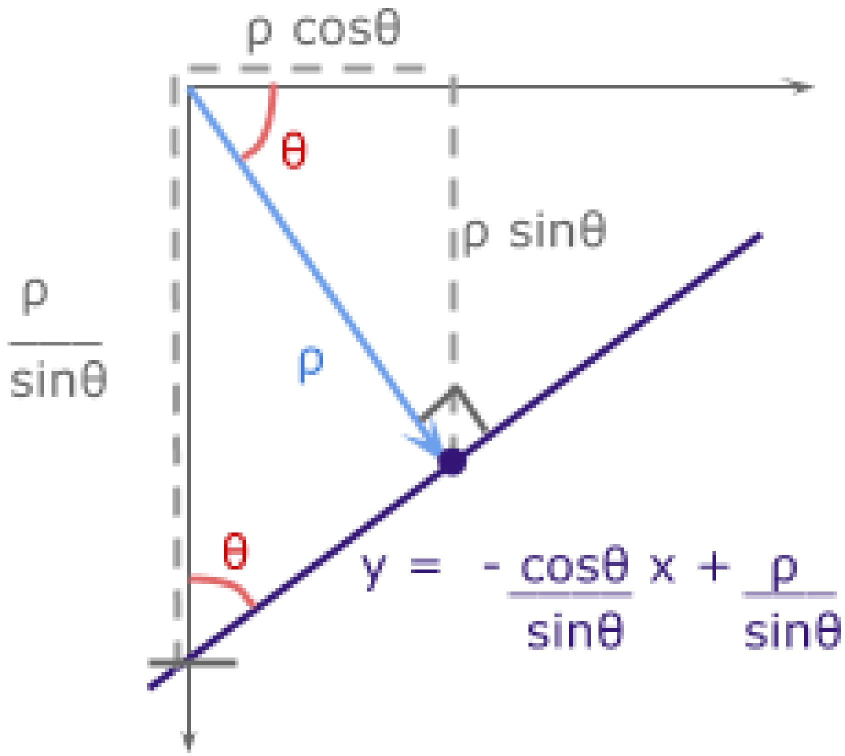
- Some lines cannot be be defined in Cartesian
- So we have to move to polar coordinates.
- In polar coordinates line is define by ρ and θ
- ρ is the norm distance of the line from origin.
- θ is the angle between the norm and the horizontal x axis.
- The equation of line in terms of ρ and θ now is

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

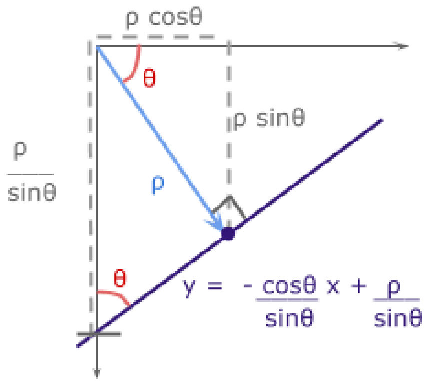
and

$$\rho = xcos(\theta) + ysin(\theta)$$

Hough Transform: Line Detection (Polar Coordinates)

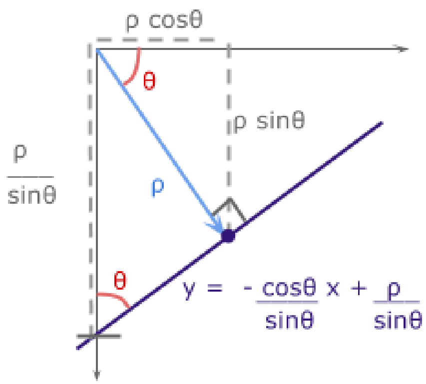


Hough Transform: Line Detection (Polar Coordinates)



The Range of values of ρ and θ

Hough Transform: Line Detection (Polar Coordinates)



The Range of values of ρ and θ

- θ: in polar coordinate takes value in range of -90 to 90
- The maximum norm distance is given by diagonal distance which is $\rho \max = \sqrt{x^2 + y^2}$
- So ρ has values in range from $-\rho\max$ to $\rho\max$

Hough Transform: Line Detection (Polar Coordinates)

Algorithm

Basic Algorithm steps for Hough transform is :

```
# Extract edges of the image (For example, 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
```

Hough Transform: Line Detection (Polar Coordinates)

Basic Implementation

At first import used libraries

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

Hough Transform: Line Detection (Polar Coordinates)

Basic Implementation

```
def hough_line(image):
    Ny = image.shape[0]
    Nx = image.shape[1]
    Maxdist = int(np.round(np.sqrt(Nx**2 + Ny ** 2)))
    thetas = np.deg2rad(np.arange(-90, 90))
    rs = np.linspace(-Maxdist, Maxdist, 2*Maxdist)
    accumulator = np.zeros((2 * Maxdist, len(thetas)))

    for y in range(Ny):
        for x in range(Nx):
            if image[y,x] > 0:
                for k in range(len(thetas)):
                    r = x*np.cos(thetas[k]) + y * np.sin(thetas[k])
                    accumulator[int(r) + Maxdist,k] += 1
    return accumulator, thetas, rs
```

Useful links

- {Understanding Hough transform in python}
- {OpenCV Hough Line Transform}
- {Scikit-image Hough Line}
- {OpenCV Hough Circle}
- {Survey of Hough transform}

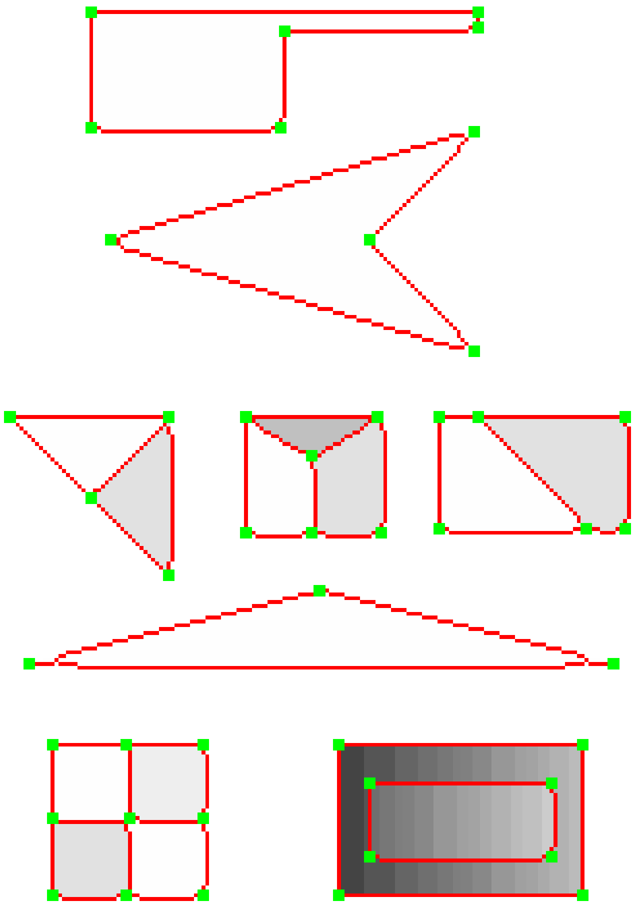
Hough Transform: Line Detection (Polar Coordinates)

jupyter

{hough_transform.ipnyb}

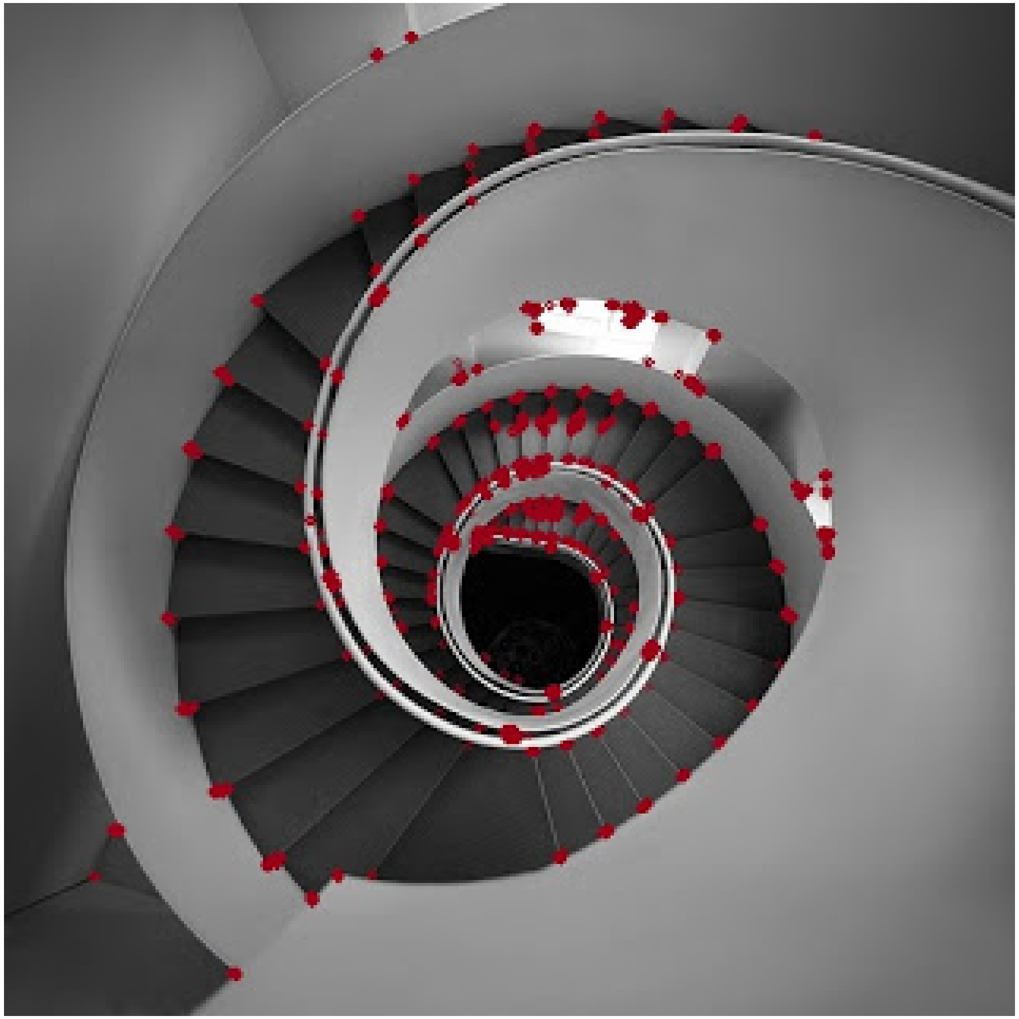
Corner Detection

Feature Detection



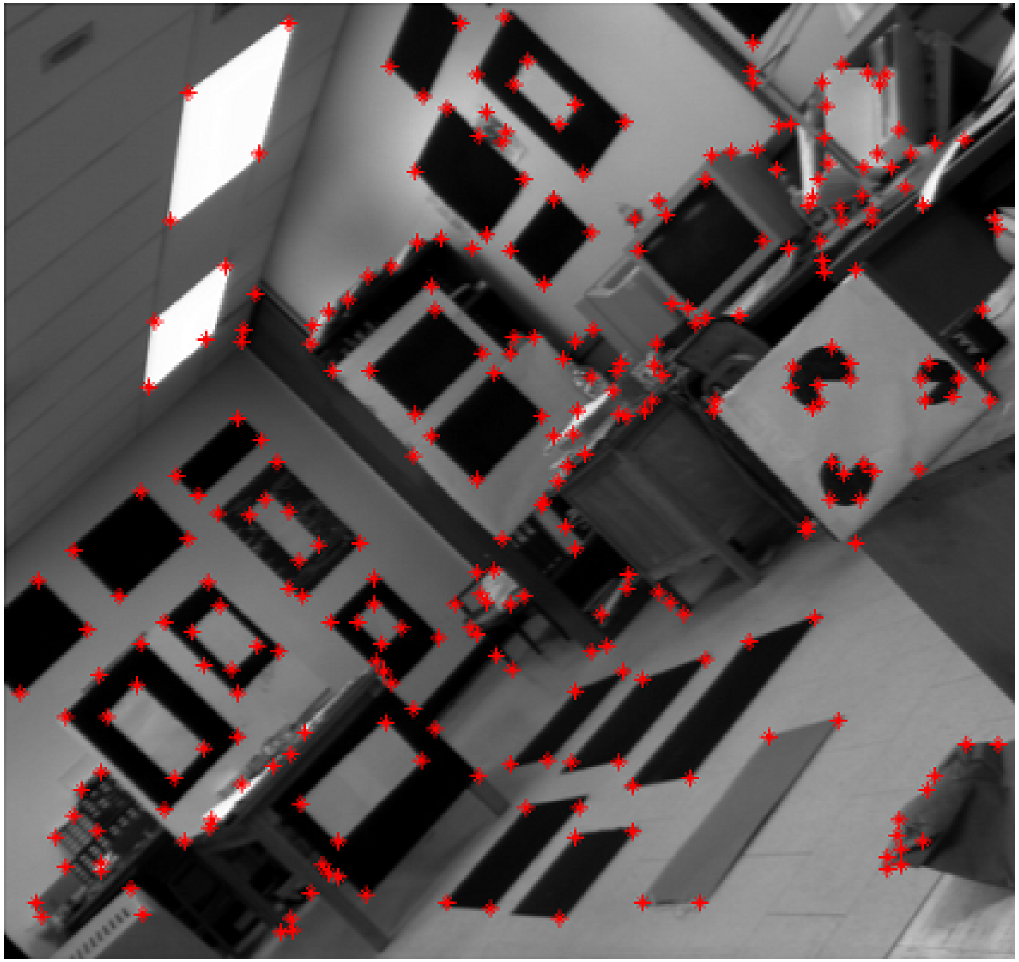
Corner Detection

Feature Detection



Corner Detection

Feature Detection



Corner Detection

Challenges

- Patch (image) matching

Corner Detection

Challenges

- Patch (image) matching
    - Distinctive features

Corner Detection

Challenges

- Patch (image) matching
  - Distinctive features
- Geometric transformations (translation, rotation, scale)

Corner Detection

Challenges

- Patch (image) matching
  - Distinctive features
- Geometric transformations (translation, rotation, scale)
  - Robust and efficient

Corner Detection

Challenges

- Patch (image) matching
  - Distinctive features
- Geometric transformations (translation, rotation, scale)
  - Robust and efficient
- Photometric (brightness, exposure)

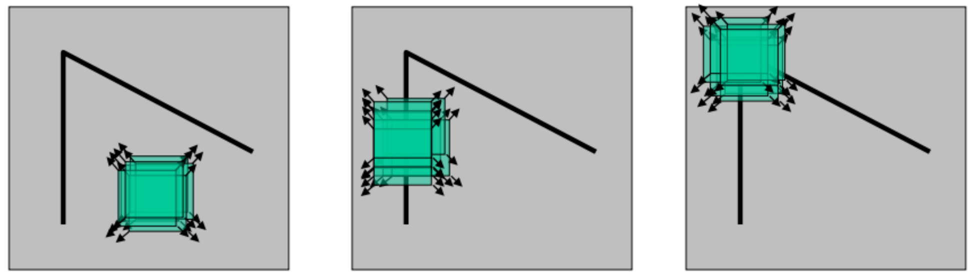
Corner Detection

Challenges

- Patch (image) matching
  - Distinctive features
- Geometric transformations (translation, rotation, scale)
  - Robust and efficient
- Photometric (brightness, exposure)
  - Many preprocessing options can be applied

Corner Detection

Harris operator: corner detector



“flat” region:
no change in
all directions

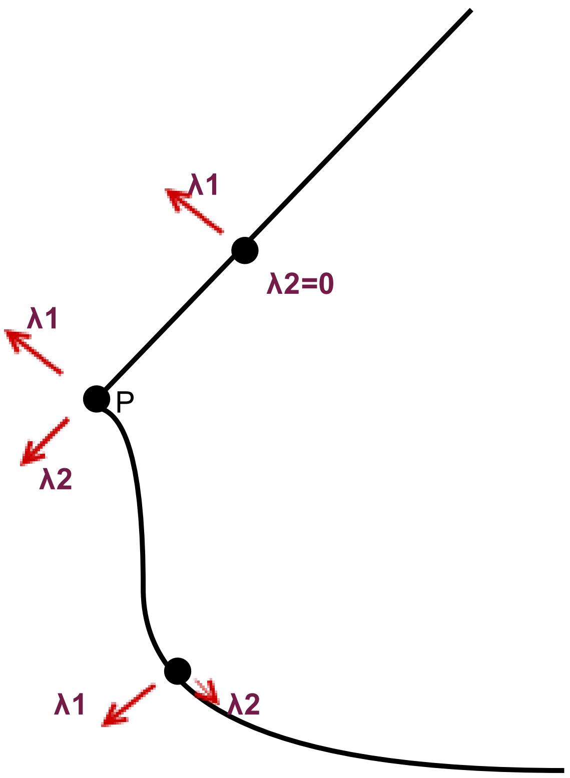
“edge”:
no change along
the edge direction

“corner”:
significant change
in all directions

Corner Detection

Harris operator: corner detector

Compute the principal vectors of variation at location p



Corner Detection: Harris operator

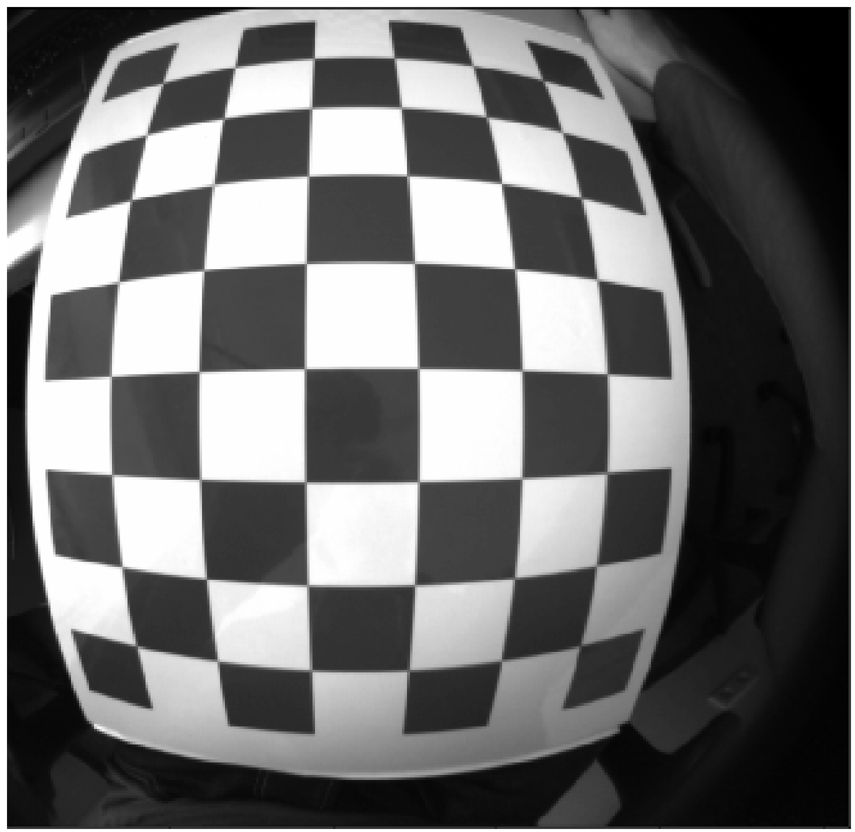
Step 1: image smoothing (optional)

Corner Detection: Harris operator

Step 1: image smoothing (optional)

$$L(p, \sigma) = [I * G_\sigma](p)$$

```
signal.convolve2d(img, gaussian_kernel(7,1.0) ,'same')
```



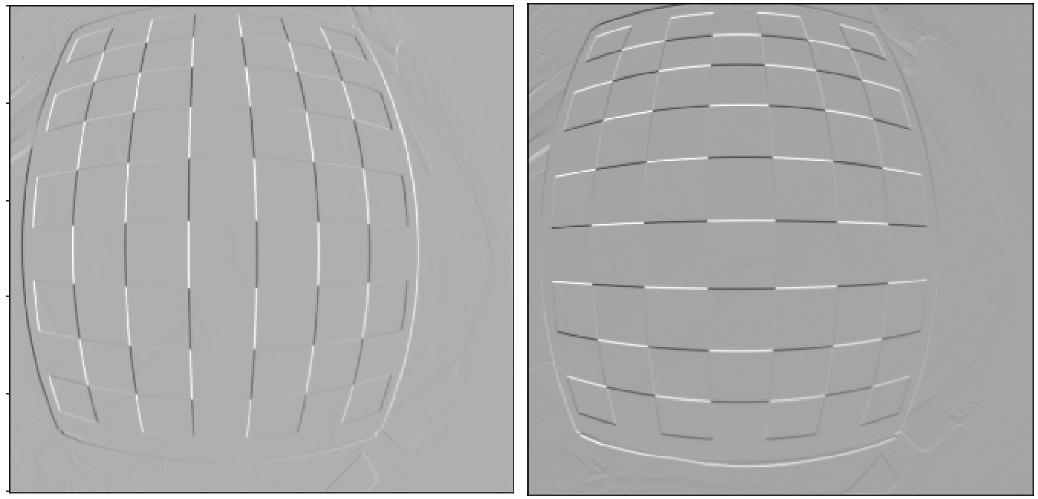
Corner Detection: Harris operator

Step 2: compute I_x and I_y

Many options to compute the I_x and I_y exist:

1. First order difference.
2. Prewitt kernel
3. Sobel kernel

```
Ix = signal.convolve2d( img , sobel_h ,'same')
Iy = signal.convolve2d( img , sobel_v ,'same')
```



Corner Detection: Harris operator

Step 3: construct the Hessian (Hesh'n) matrix M

We will construct the Hessian matrix so we are able to compute the principal vectors of variation.

Corner Detection: Harris operator

Step 3: construct the Hessian (Hesh'n) matrix M

We will construct the Hessian matrix so we are able to compute the principal vectors of variation.

$$M(p) = \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

Corner Detection: Harris operator

Step 3: construct the Hessian (Hesh'n) matrix M

We will construct the Hessian matrix so we are able to compute the principal vectors of variation.

$$M(p) = \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

```
Ixx =  np.multiply( Ix, Ix)
Iyy =  np.multiply( Iy, Iy)
Ixy =  np.multiply( Ix, Iy)
```

Corner Detection: Harris operator

Step 3 (Alternative): construct the Hessian (Hesh'n) matrix M **over a window**

- If we need more robust detection

Corner Detection: Harris operator

Step 3 (Alternative): construct the Hessian (Hesh'n) matrix M over a window

- If we need more robust detection
- Compute M over a window (e.g 3×3)

Corner Detection: Harris operator

Step 3 (Alternative): construct the Hessian (Hesh'n) matrix M over a window

- If we need more robust detection
- Compute M over a window (e.g 3×3)
- Now can detect larger corner that lives inside a window of pixels, instead of a single pixel.

Corner Detection: Harris operator

Step 3 (Alternative): construct the Hessian (Hesh'n) matrix M over a window

- If we need more robust detection
- Compute M over a window (e.g 3×3)
- Now can detect larger corner that lives inside a window of pixels, instead of a single pixel.

$$\hat{M}(p) = \sum_{i,j} w(i,j) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

Corner Detection: Harris operator

Step 3 (Alternative): construct the Hessian (Hesh'n) matrix M over a window

- If we need more robust detection
- Compute M over a window (e.g 3×3)
- Now can detect larger corner that lives inside a window of pixels, instead of a single pixel.

$$\hat{M}(p) = \sum_{i,j} w(i,j) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

$$\hat{M}(p) = \begin{bmatrix} \sum w(i,j) I_x^2(i,j) & \sum w(i,j) I_x I_y(i,j) \\ \sum w(i,j) I_x I_y(i,j) & \sum w(i,j) I_y^2(i,j) \end{bmatrix}$$

Corner Detection: Harris operator

Step 3 (Alternative): construct the Hessian (Hesh'n) matrix M over a window

$$\hat{M}(p) = \begin{bmatrix} \widehat{I_x^2} & \widehat{I_x I_y} \\ \widehat{I_x I_y} & \widehat{I_y^2} \end{bmatrix}$$

Corner Detection: Harris operator

Step 3 (Alternative): construct the Hessian (Hesh'n) matrix M over a window

$$\hat{M}(p) = \begin{bmatrix} \hat{I_x^2} & \widehat{I_x I_y} \\ \widehat{I_x I_y} & \hat{I_y^2} \end{bmatrix}$$

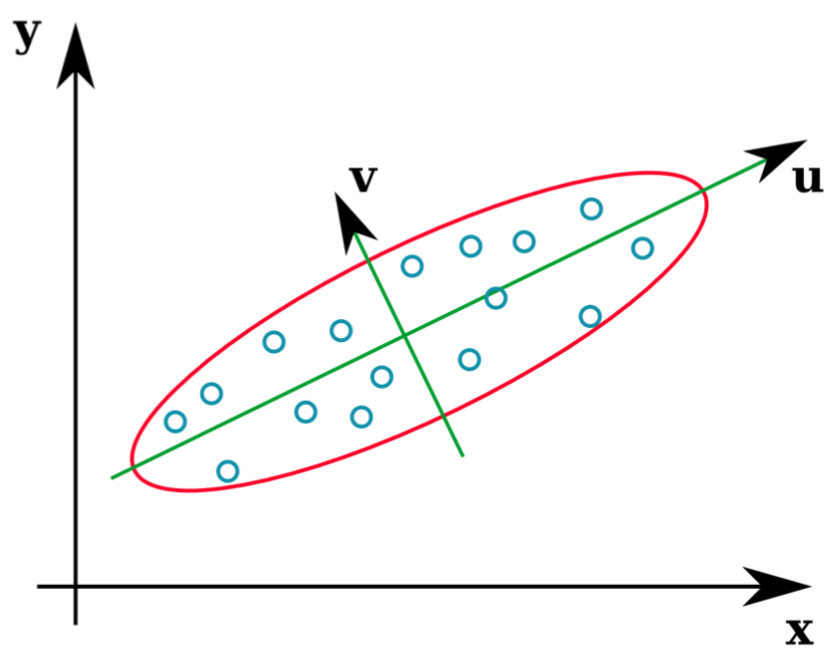
```
Ixx =  np.multiply( Ix, Ix)
Iyy =  np.multiply( Iy, Iy)
Ixy =  np.multiply( Ix, Iy)

Ixx_hat = signal.convolve2d( Ixx , box_filter(3) ,'same')
Iyy_hat = signal.convolve2d( Iyy , box_filter(3) ,'same')
Ixy_hat = signal.convolve2d( Ixy , box_filter(3) ,'same')
```

Corner Detection: Harris operator

Step 4: compute λ_1 and λ_2 of $\hat{M}$

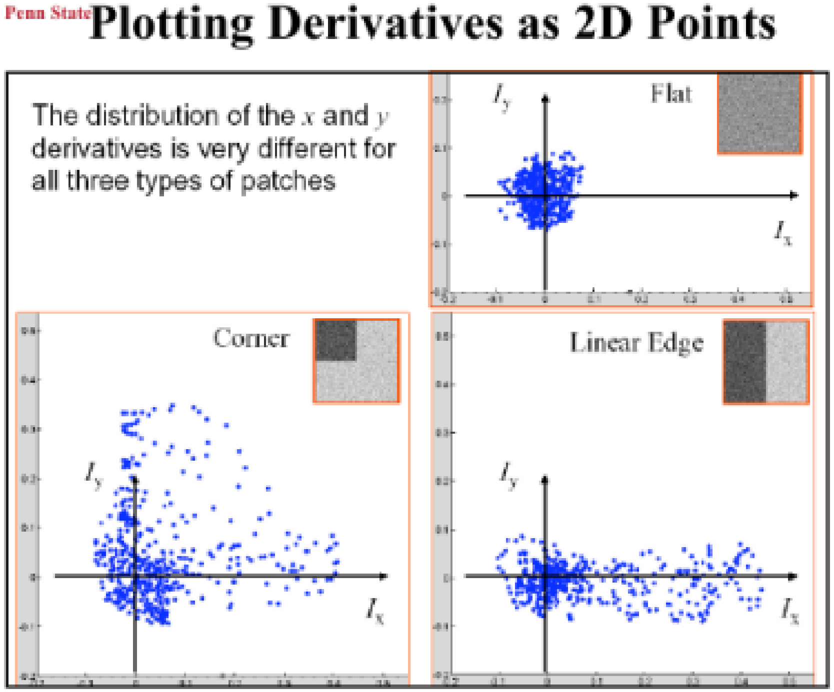
- Hessian matrix $$\mathbf{H}(p) = \begin{bmatrix} I_{xx}(p) & I_{xy}(p) \\ I_{xy}(p) & I_{yy}(p) \end{bmatrix}$$

- Eigen vectors and Eigen values
    - values (amount of variation)
    - vector (variation direction)



Corner Detection: Harris operator

Step 4: compute λ_1 and λ_2 of $\hat{M}$



Corner Detection: Harris operator

Step 4: compute λ_1 and λ_2 of $\hat{M}$

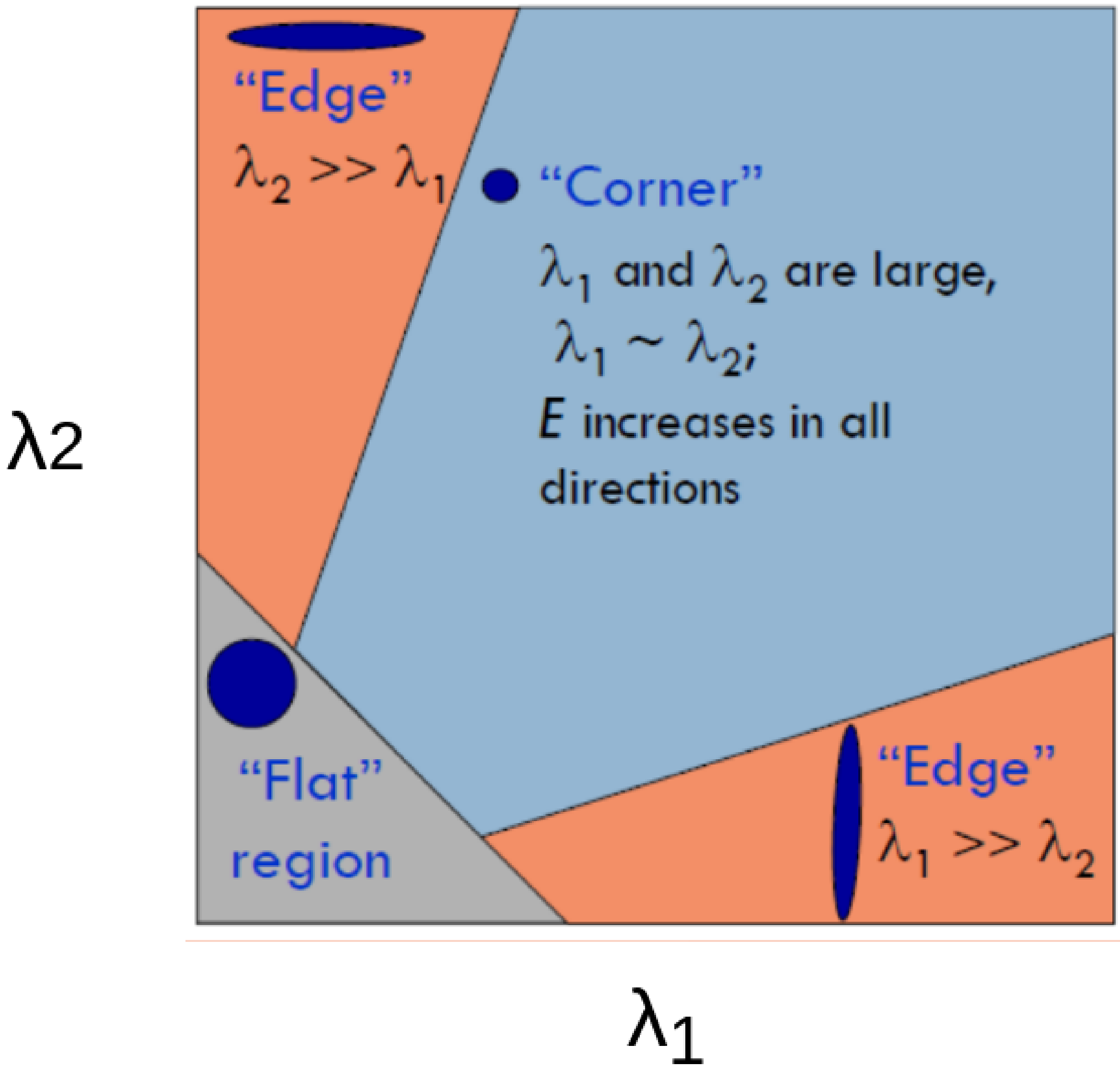
Corner Detection: Harris operator

Step 4: compute λ_1 and λ_2 of $\hat{M}$

$$|H - \lambda I| = 0$$

Corner Detection: Harris operator

Step 4: compute λ_1 and λ_2 of $\hat{M}$



Corner Detection: Harris operator

Step 5: evaluate corners using R as a measure

Corner Detection: Harris operator

Step 5: evaluate corners using R as a measure

$$R = (\lambda_1 \times \lambda_2) - k(\lambda_1 + \lambda_2)$$

Corner Detection: Harris operator

Step 4 (Alternative): evaluate R directly without λ_1 and λ_2

Indirect solution

Corner Detection: Harris operator

Step 4 (Alternative): evaluate R directly without λ_1 and λ_2

Indirect solution

$$det(M) = \lambda_1 \times \lambda_2$$

Corner Detection: Harris operator

Step 4 (Alternative): evaluate R directly without λ_1 and λ_2

Indirect solution

$$det(M) = \lambda_1 \times \lambda_2$$

$$trace(M) = \lambda_1 + \lambda_2$$

Corner Detection: Harris operator

Step 4 (Alternative): evaluate R directly without λ_1 and λ_2

Indirect solution

$$det(M) = \lambda_1 \times \lambda_2$$

$$trace(M) = \lambda_1 + \lambda_2$$

Instead of calculating λ_1, λ_2

Corner Detection: Harris operator

Step 4 (Alternative): evaluate R directly without λ_1 and λ_2

Indirect solution

$$det(M) = \lambda_1 \times \lambda_2$$

$$trace(M) = \lambda_1 + \lambda_2$$

Instead of calculating λ_1, λ_2

- $R = det(\hat{M}) - k * trace(\hat{M})$

Corner Detection: Harris operator

Step 4 (Alternative): evaluate R directly without λ_1 and λ_2

Indirect solution

$$det(M) = \lambda_1 \times \lambda_2$$

$$trace(M) = \lambda_1 + \lambda_2$$

Instead of calculating λ_1, λ_2

- $R = det(\hat{M}) - k * trace(\hat{M})$
- Trace is sum of diagonal elements

Corner Detection: Harris operator

Step 4 (Alternative): evaluate R directly without λ_1 and λ_2

$$\hat{M}(p) = \begin{bmatrix} \hat{I_x^2} & \hat{I_x I_y} \\ \hat{I_x I_y} & \hat{I_y^2} \end{bmatrix}$$

$$R = det(\hat{M}) - k * trace(\hat{M})$$

```
K = 0.05

detM = np.multiply(Ixx_hat,Iyy_hat) - np.multiply(Ixy_hat,Ixy_hat)
trM = Ixx_hat + Iyy_hat
R = detM - K * trM
```

Corner Detection: Harris operator

Finally

```
corners = ???
```

Select large values of R, using whatever thresholding heuristic in mind.

Thresholding options:

- constant absolute value
    - (e.g `corners = np.abs(R) > 2.5`)

Corner Detection: Harris operator

Finally

```
corners = ???
```

Select large values of R, using whatever thresholding heuristic in mind.

Thresholding options:

- constant absolute value
  - (e.g `corners = np.abs(R) > 2.5`)
- relative to maximum value
  - (e.g `corners = np.abs(R) > 0.2 * np.max(R)`)

Corner Detection: Harris operator

Finally

```
corners = ???
```

Select large values of R, using whatever thresholding heuristic in mind.

Thresholding options:

- constant absolute value
  - (e.g `corners = np.abs(R) > 2.5`)
- relative to maximum value
  - (e.g `corners = np.abs(R) > 0.2 * np.max(R)`)
- relative to quantile value
  - (e.g `corners = np.abs(R) > np.quantile(np.abs(R),0.9)`)

Corner Detection: Harris operator

Results

Corner Detection: Harris operator
Results