



Image Processing



Edge Detection





Sobel Filter

- ◆ The Sobel filter uses the following filter to get the gradient in each direction

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I$$

SYNTAX:

`dst=cv2.Sobel(img,ddepth,dx,dy,ksize)`

Dx and dy: specify which direction you want to take the gradient in

Ksize: filter size which can be 3,5,7 (if filter size is 3 then Scharr filter is used)

Ddepth: taken as CV_64F





Canny Edge Detection

- ◇ This involves 4 steps of processing
 1. Noise Reduction (5X5 Gaussian)
 2. Finding Intensity Gradient of Image (Sobel)
 3. Non-Maximum Suppression
 4. Hysteresis Thresholding

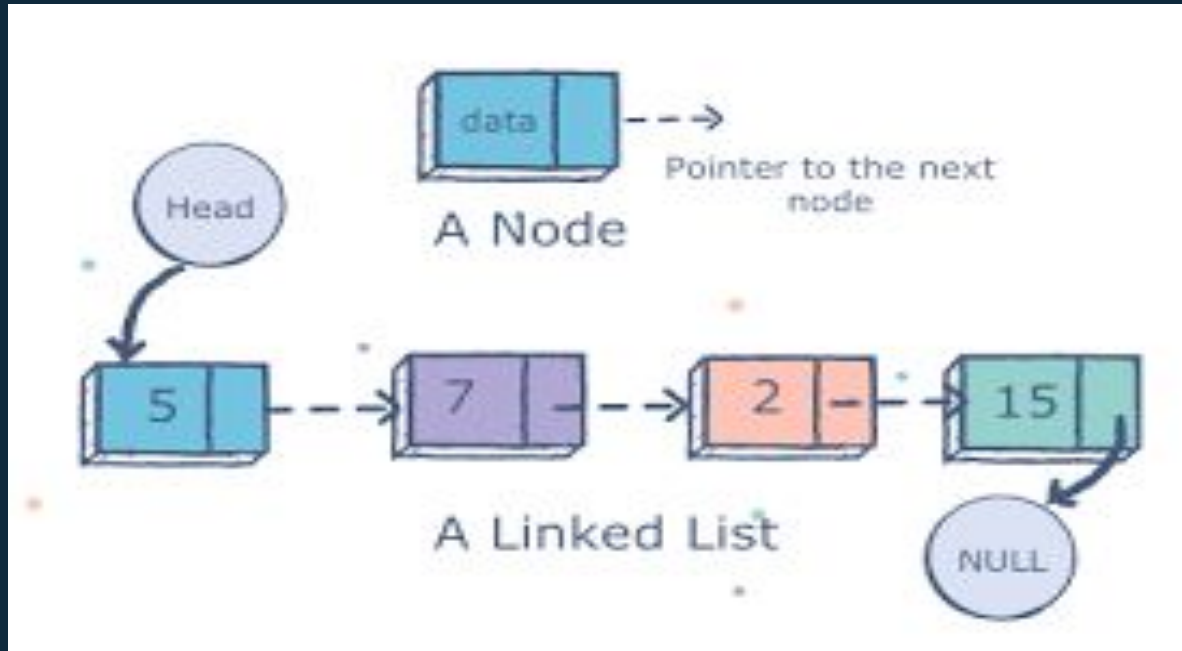
SYNTAX:

```
result=cv2.Canny(img,maxVal,minVal)
```

maxVal and **minVal** are the values used in hysteresis thresholding. Generally taken in the ratio 2:1 or 3:1.



Linked List



Stacks

TOP = -1



**empty
stack**

**TOP = 0
stack[0] = 1**



push

**TOP = 1
stack[1] = 2**



push

**TOP = 2
stack[2] = 3**



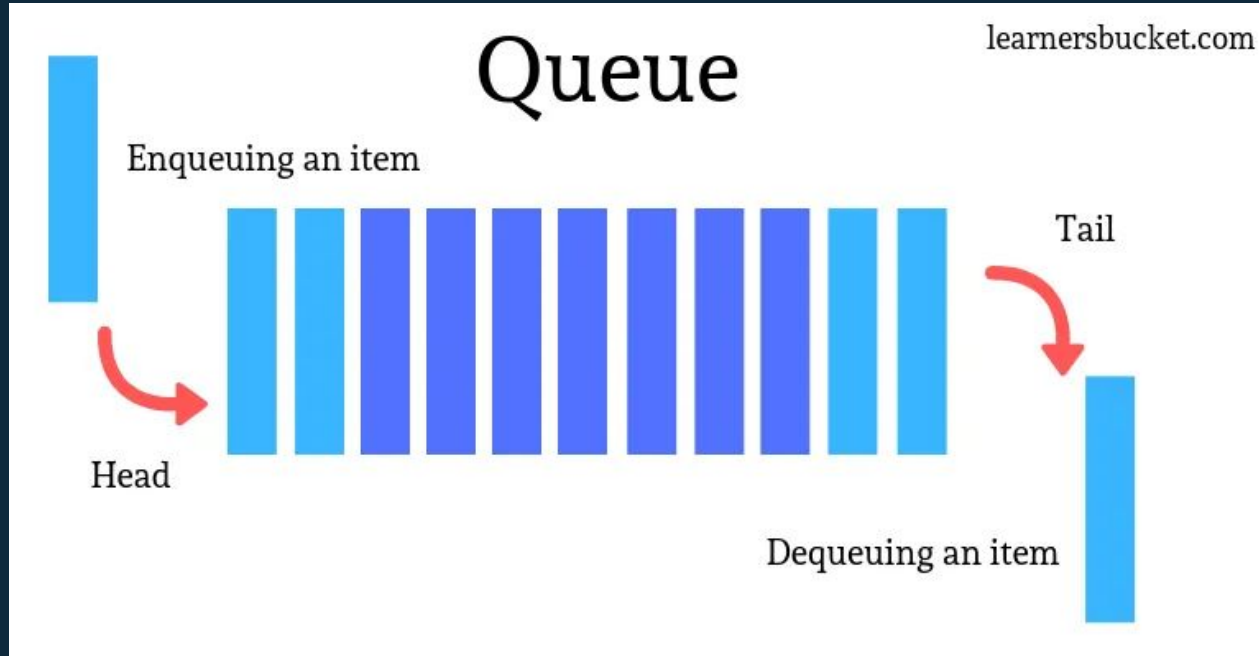
push

**TOP = 1
return stack[2]**

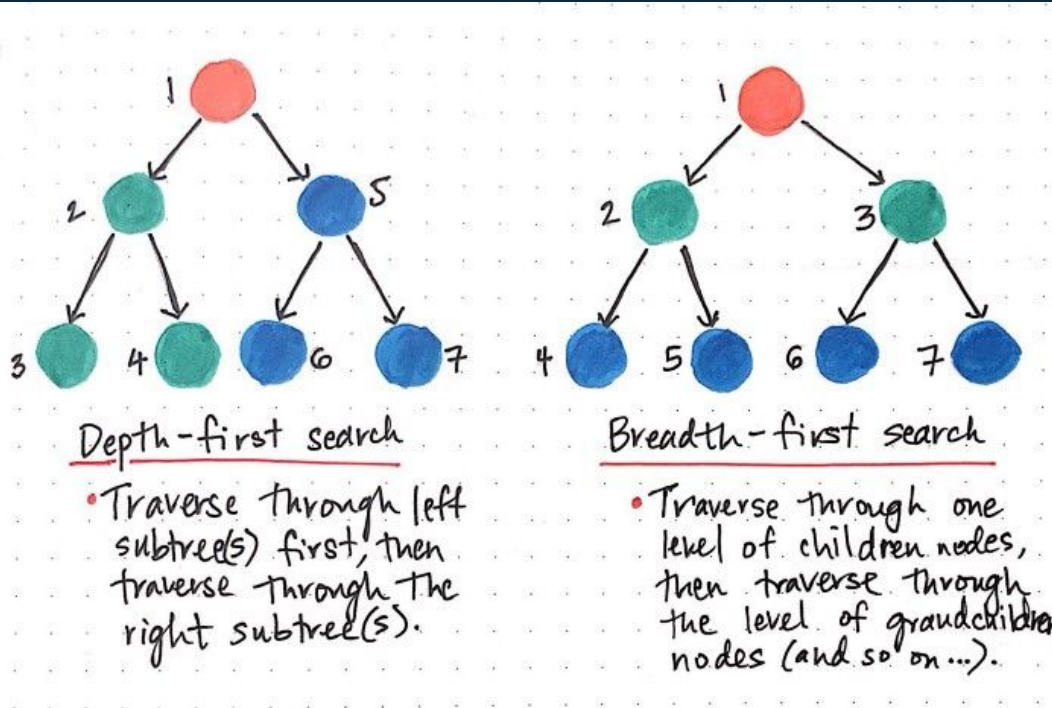


pop

Queue



BFS and DFS





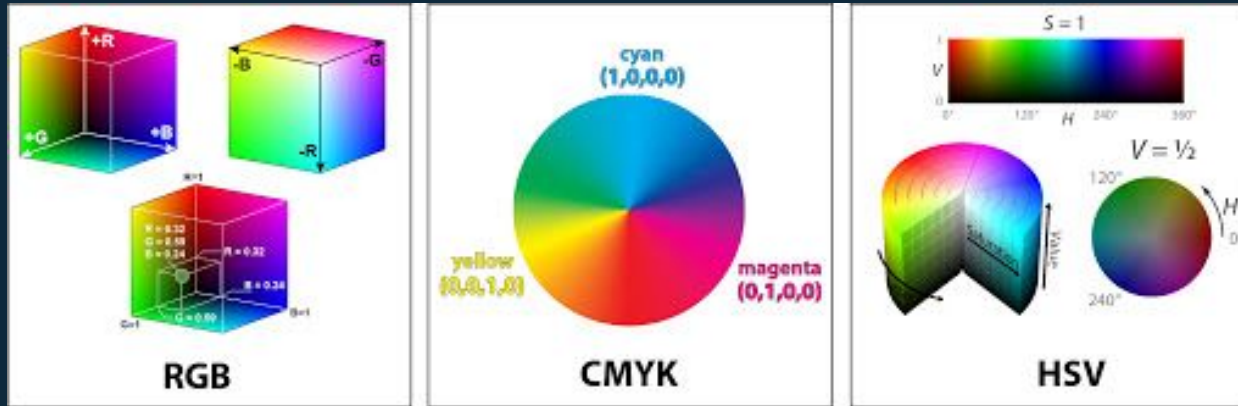
Video Capture

IP on videos works exactly like that on images. Using OpenCV “video capture” feature, we take the input video in the form of multiple “frames”, where each frame can be thought of as representing an image, and multiple such images, when played successively with a very short interval of time between them, give us the videos having motion!

See the sample snippet...

```
cap = cv.VideoCapture(0)
if not cap.isOpened():
    print("Cannot open camera")
    exit()
while True:
    # Capture frame-by-frame
    ret, frame = cap.read()
    # if frame is read correctly ret is True
    if not ret:
        print("Can't receive frame (stream
        end?). Exiting ...")
        break
    # Our operations on the frame come here
    processed_frame = ....
    # Display the resulting frame
    cv.imshow('frame', processed_frame)
    if cv.waitKey(1) == ord('q'):
        break
```

Color Spaces



A Color Space acts as a reference for what colors we can “actually see”. Screens having different Color Spaces will have different range of colors that can be reproduced by them/perceived by us. Remember the Chromaticity Diagram and the Gamuts different color spaces had?



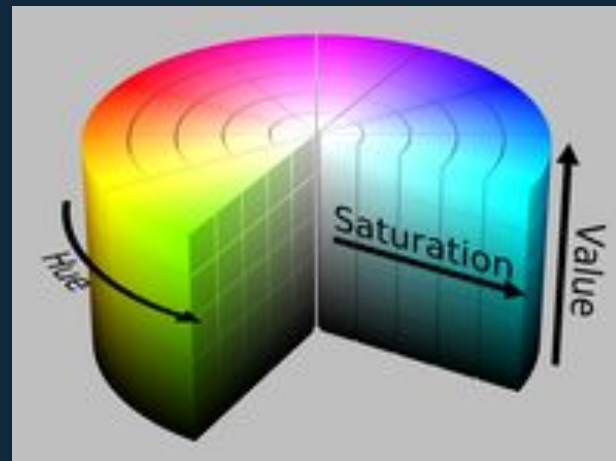
HSV Color Space

What is HSV?

HSV means Hue-Saturation-Value, where the Hue is the color, Saturation is the greyness (a saturation value near 0 means it is dull or grey looking), and Value is the brightness of the pixel.

Why use HSV?

HSV color space is more often used in Computer Vision owing to its superior performance compared to RGB color space in varying illumination levels. Often thresholding and masking is done in HSV color space. It helps us do better segmentation and filtering on objects even under varying resolution and lighting conditions.





Conversion b/w HSV & BGR

There are more than 150 color-space conversion methods available in OpenCV. The most common ones are among BGR, Gray and HSV.

For BGR \rightarrow HSV, we use the flag `cv.COLOR_BGR2HSV`.


Eg, `hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)`

For HSV, hue range is [0,179], saturation range is [0,255], and value range is [0,255] in OpenCV.

`cv.cvtColor()`, on passing BGR values instead of passing an image, returns the color in hsv scale.

Eg, `cv.cvtColor(green,cv.COLOR_BGR2HSV)` returns `[[60 255 255]]`

On taking [H-10, 100,100] and [H+10, 255, 255] as the lower bound and upper bound respectively, we get the desired hsv ranges of green.





Hough Line

Transformation

Hough Transformation is used to detect curves in binary images after they have undergone edge detection (edges depicted as white).

Hough Line Transformation works by analyzing each pixel in the image and plots them in a (θ, r) space (Hough plane), if they are white. The number of intersection points gives us the number of unique lines and the coordinates of each such point give us the parameters of each of those lines.

`cv.HoughLines`(img, rho, theta, threshold)

Parameters:

img: 8-bit, single-channel binary source image. The image may be modified by the function.

rho: Resolution of the parameter r

theta: Resolution of the parameter θ (in radians, $1 \text{ deg} = \text{CV_PI}/180$)

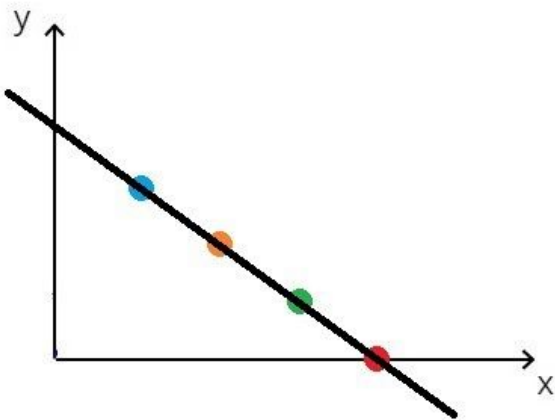
threshold: The minimum number of intersections to "detect" a line

There are other optional parameters. Check OpenCV docs to know more.

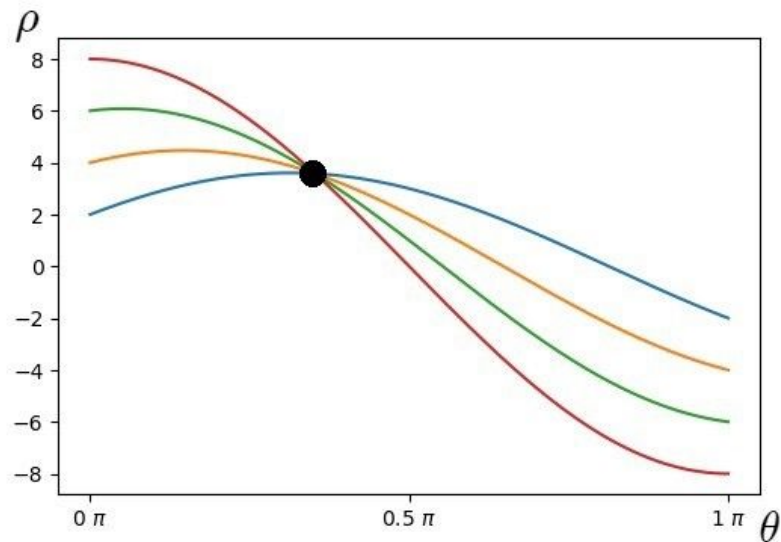




Hough Lines in Action



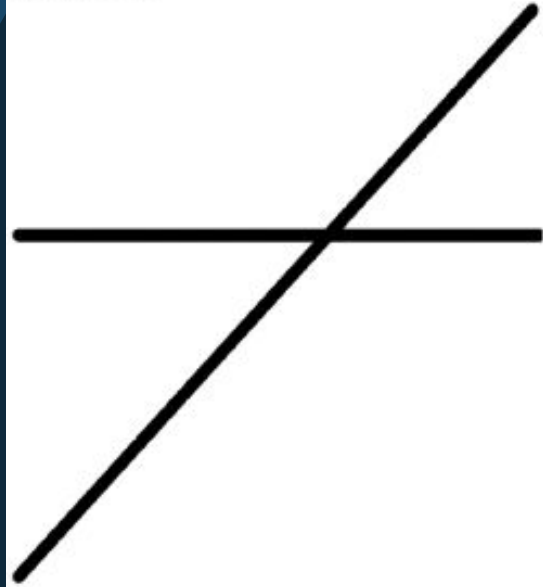
Points which form a line



Bunch of sinusoids intersecting at one point

Hough Lines in Action

Input Image



Rendering of Transform Results





Contours

- ◇ Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity.
- ◇ Found on binary images
- ◇ https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_contours/py_contours_begin/py_contours_begin.html#contours-getting-started





Find Contours

Syntax:


```
contours,hierarchy=cv2.findContours(image,contour_retrieval_mode,contour_approximation_method)
```

Contour Retrieval Methods

This defines how the hierarchy between contours is dealt with. Is generally **cv2.RETR_TREE** or **cv2.RETR_LIST**. More on the next slide.

Contour Approximation Methods

This defines how the points in contour is stored. Are all points needed to store the contour? **cv2.CHAIN_APPROX_SIMPLE** approximates the contour to store less points. To store all points use **cv2.CHAIN_APPROX_NONE**



Contour Hierarchy

TERMS:

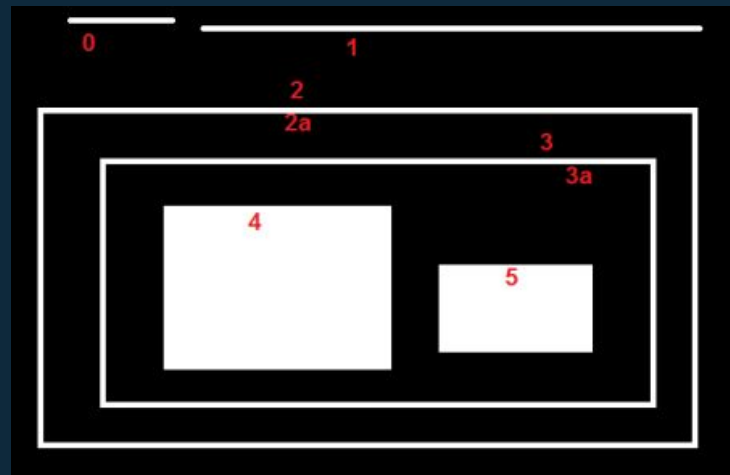
- ◇ Same Hierarchy Level
- ◇ External Contour
- ◇ Child and Parent Relationship
- ◇ First Child

REPRESENTATION:

[Next, Previous, First_Child, Parent]

CONTOUR RETRIEVAL MODES

- ◇ **RETR_LIST**: returns all with no parent child relationship
- ◇ **RETR_EXTERNAL**: return only external contours
- ◇ **RETR_CCOMP**: Classify into 2 level hierarchy
- ◇ **RETR_TREE**: returns complete hierarchy





Draw Contours

SYNTAX:

```
img=cv2.drawContours(img,contours,contour_index,colour)
```

Has a few more optional parameters like thickness,lineType,hierarchy,maxLevel and offset.

PARAMETERS:

contour_index: which contour should be drawn. -1 if you want to draw all contours.





Contour Features

AREA: `area=cv2.contourArea(contour)`

PERIMETER: `peri=cv2.arcLength(contour)`

CONTOUR APPROXIMATION:

`approxCurve=cv2.approxPolyDP(curve,epsilon,closed)`

Epsilon is the error parameter: generally 10% of perimeter

Closed: is a boolean to make the approximate curve closed

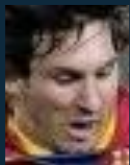
Many more [here](#)





Template Matching

- ◇ Template Matching is a method for searching and finding the location of a template image in a larger image.
- ◇ OpenCV has a function called `cv2.matchTemplate()` for this purpose.



Template



Image



Result





cv2.matchTemplate


SYNTAX:

result=cv2.matchTemplate(image,template,method)

METHODS:

- ◇ **TM_SQDIFF**: squared difference between template and image
- ◇ **TM_SQDIFF_NORMED**: Normalise the output of squared difference
- ◇ **TM_CCORR**: Multiply the two images together (Cross Correlation). If they line up exactly, they will give highest value. Used generally for signal processing (resonance). Not used here much
- ◇ **TM_CCORR_NORMED**: Normalised CCORR
- ◇ **TM_CCOEFF**: Correlation Coefficient. It subtracts mean of template and image from themselves before comparing to give better results.
- ◇ **TM_CCOEFF_NORMED**: Normalised CCOEFF

RETURNS: Grayscale image, where each pixel denotes how much does the neighbourhood of that pixel match with template.





cv2.minMaxLoc

- ◇ cv2.matchTemplate() returns a grayscale image in which there are scores for each point.
- ◇ We need to find the maximum score. (for TM_SQDIFF min gives best match)
- ◇ OpenCV provides a function called cv2.minMaxLoc that does this

SYNTAX:

`min_val,max_val,min_loc,max_loc=cv2.minMaxLoc(image)`

RETURNS: maximum and minimum value and the location of both (x,y) coordinate. Use this point as the top left corner of the detected rectangle.





Detecting Multiple Objects

- ◇ Use `matchTemplate` to give the `grayScale` image with the scores.
- ◇ Use thresholding to get the best outputs.





Drawbacks

- ◇ Pattern occurrences have to preserve the orientation of the reference pattern image(template)
- ◇ As a result, it does not work for rotated or scaled versions of the template as a change in shape/size/shear etc. of object w.r.t. template will give a false match.
- ◇ The method is inefficient when calculating the pattern correlation image for medium to large images as the process is time consuming.



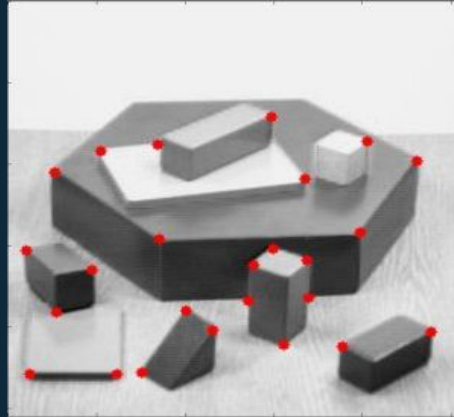
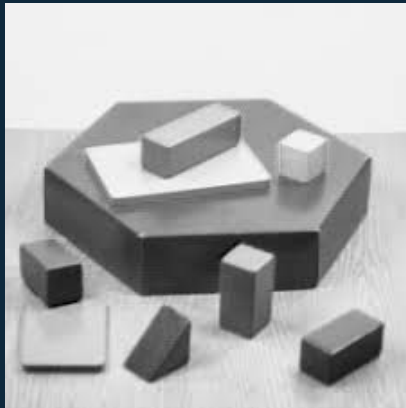


Better Methods

- ◇ Multiscaling
- ◇ Feature Based Template Matching
- ◇ Machine Learning or Deep Learning frameworks.



Corner Detection



Why Corners?



Text



How to Detect Corners?

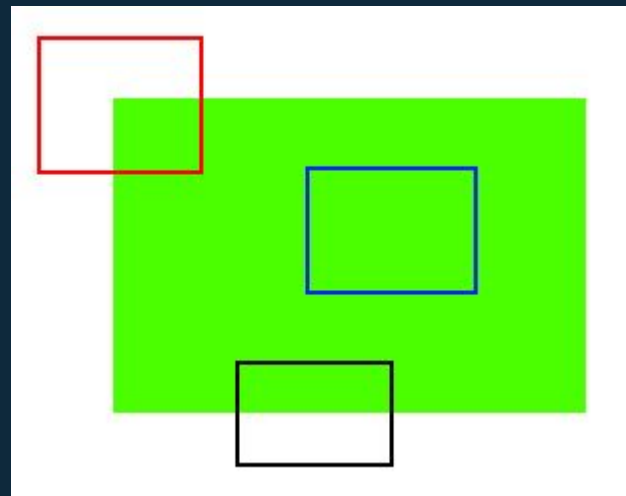
- ◆ Corners are regions in the image with large variation in intensity in all the directions.

$$E(u, v) = \sum_{x, y} \underbrace{w(x, y)}_{\text{window function}} \underbrace{[I(x + u, y + v) - I(x, y)]}_{\text{shifted intensity}}^2 \underbrace{I(x, y)}_{\text{intensity}}$$

$$M = \sum_{x, y} w(x, y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

$$R = \det(M) - k(\text{trace}(M))^2$$

- ◆ When this value is large it corresponds to a corner





Harris Corner Detection

- ◇ We have a function to perform Harris corner detection in OpenCV
- ◇ `dst=cv2.cornerHarris(src, blocksize, ksize, k)`

PARAMETERS:

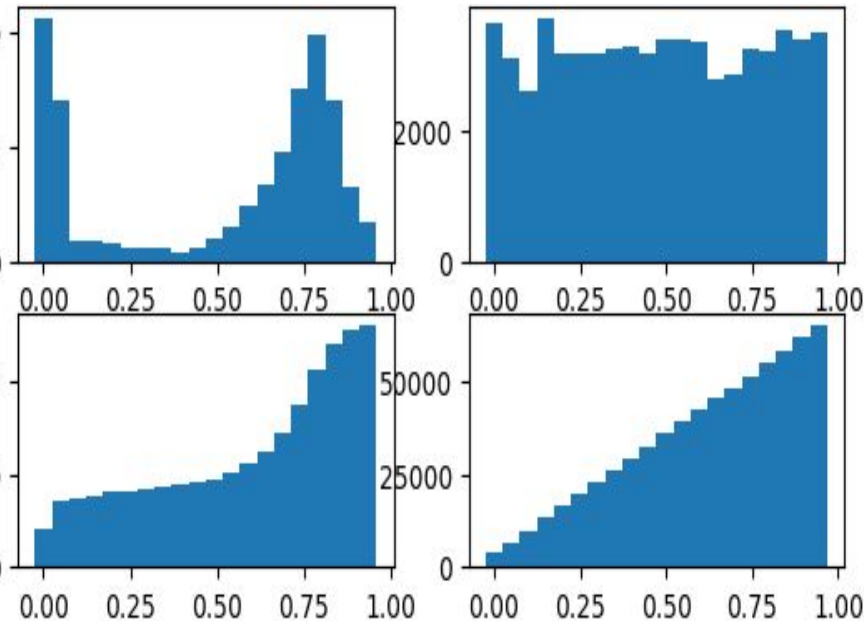
- ◇ `blockSize`: Neighbourhood Size
- ◇ `kSize`: Size of aperture of sobel operator
- ◇ `K`: Harris detector free parameter (the `k` in the `R` formula)
- ◇ It is just taken as `cv2.cornerharris(img,2,3,0.04)`

RETURNS:

A grayscale image with `R` values at each point. Threshold this ($0.01 * \max$) to detect corner points.



Histogram Equalization



$$p_n = \frac{\text{number of pixel of intensity } n}{\text{total number of pixels}}$$

0,1,2,3
0.25, 0.25, 0.25
0.25

X : pixel intensity in original image

New Image $Y = T(x)$

$$cdf(X) : F(x) = \sum (X \leq x)$$


$$\text{New cdf} : F'(y) = \sum (Y \leq y)$$

$Y = T(X)$ Such That $Y: U(0, 1)$

$$F'(y) = \sum (T(X) \leq y) = y$$

Clearly $F(X)$ fits the role of $T(X)$

$F(n)$ = Sum of All probabilities such that intensity $\leq n$
 $F(0) = 0.25$
 $F(1) = 0.50$
 $F(2) = 0.75$
 $F(3) = 1.0$
 $(0 \rightarrow (L-1))$
 $1/(L)$
 $F(n) = n/(L-1)$



2. All pixels having earlier intensity x now have y .

In case of Histogram equalization the Function is the CDF of the Histogram.

2. Probability for all pixel intensities. $P[0-----255]$

3. Find the CDF : $cdf[0---255]$

4. For all pixels in the original image:

 Their current intensity : $img[i,j] : 0-255$

 Probability : $p[img[i,j]] :$

 Cdf : $cdf[img[i,j]] - (0-1)*(255)$

 Assign value to pixel





Dijkstra's Algorithm

TBD

