COMPUTER VISION LAB 4

OpenCV Inbuilt Functions

MUNADI SIAL

SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

Overview

This lab will be centered on the following:

- Gaussian Blur
- Canny Edge Detection
- Bitwise Operations
- Color Spaces
- InRange Function
- Centroids
- Perspective Transform

Review

To load an image:

```
img = cv2.imread(“file.jpg”, 1)
```

To display an image:

```
cv2.imshow('image1',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

To save an image:

```
cv2.imwrite(‘myImageRotated.jpg',imgA)
```

To get image width and height:

```
rows = img.shape[0]
cols = img.shape[1]
```

Review

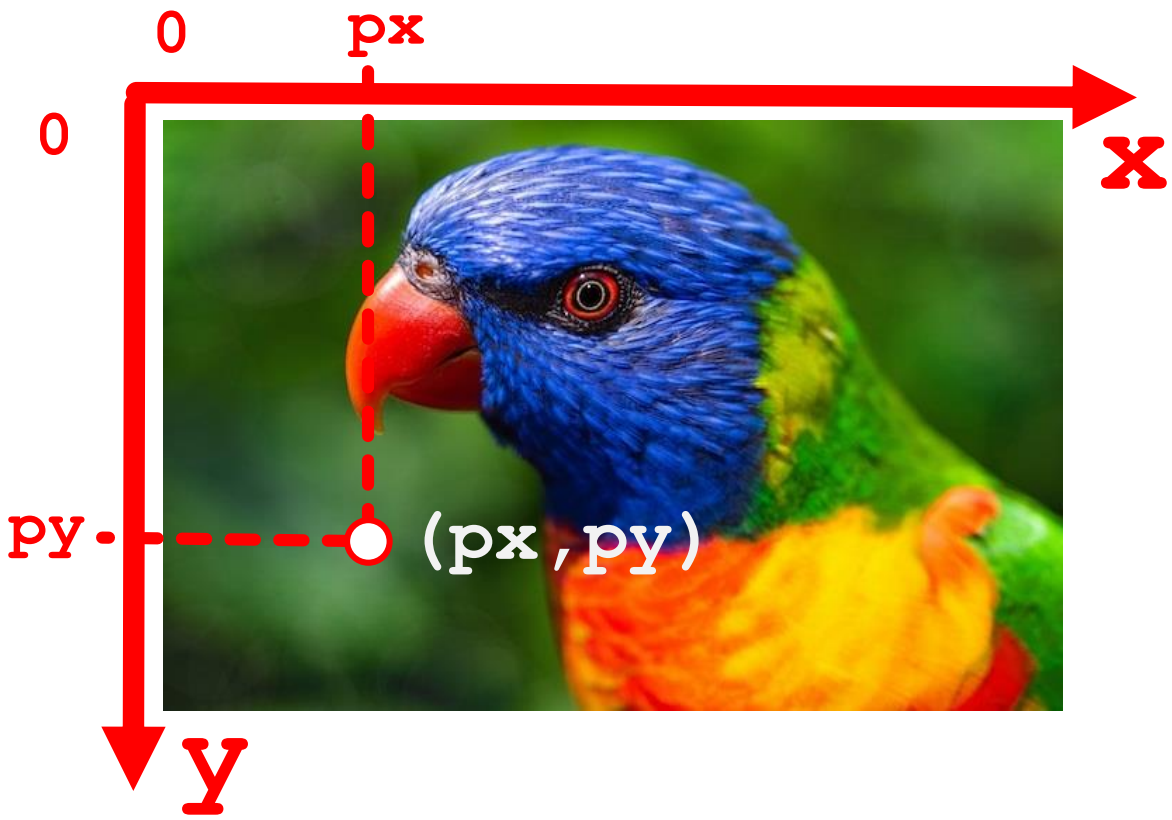
To get the BGR values of a specific pixel at location (px, py), we use:

```
img = cv2.imread('bird.jpg', 1)

val = img[py, px, :]
```

To change pixel colors:

```
img[24,120,:] = (255,0,0)
img[320,84,:] = (255,255,255)
img[71,120,:] = (0,0,255)
img[56,153,:] = (0,0,0)
img[200,:,:] = (0,255,0)
img[:,300,:] = (255,0,255)
img[5:80,300,:] = (88,34,14)
```



Gaussian Blur

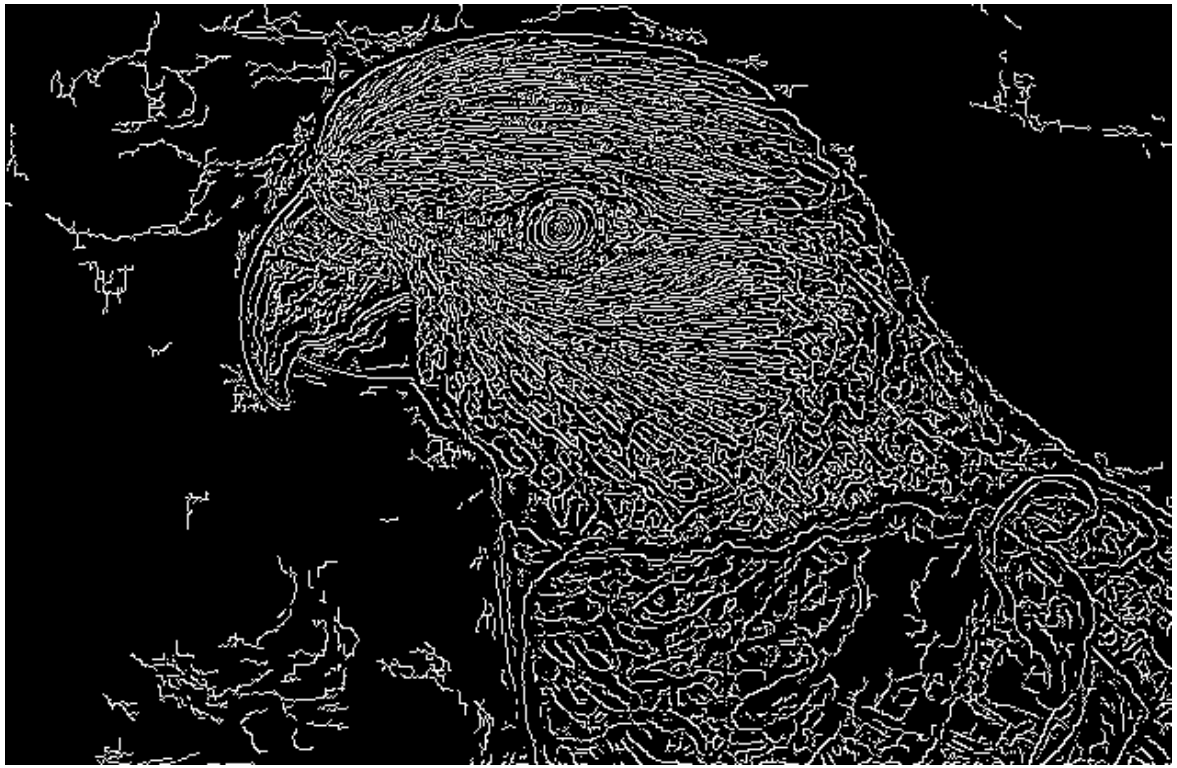
To blur an image:

```
blur = cv2.GaussianBlur(img,(5,5),0)
```

- The first argument is the image
- The second argument is the kernel size. A larger kernel has more blurring. Kernel sizes can be 3x3, 5x5, 7x7 and so on. The size dimension must be a positive odd number
- The third argument is the standard deviation of the Gaussian function. It it is set to 0, the standard deviation is calculated from the kernel size.

Gaussian Blur

Blurring is a preprocessing step to remove noise in the image data



Edge detection

Edge detection after blurring

Edge Detection

To use the Canny edge detector:

```
edged = cv2.Canny(img,10,200)
```

- The first argument is the image
- The second and third arguments are the lower and upper thresholds respectively. The thresholds are for the edge gradients. A gradient above the upper is definitely an edge. A gradient below the lower threshold is definitely not an edge. Gradient values between the thresholds are either edges or non-edges depending on their connectivity of nearby edges

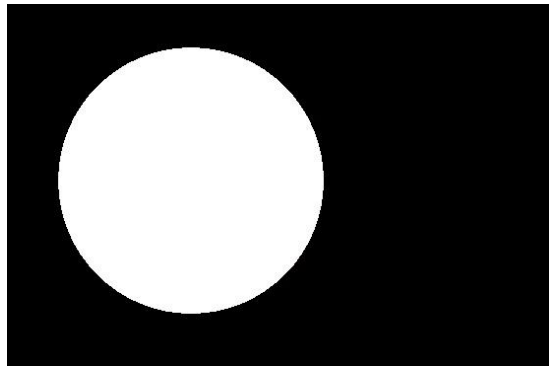
Low gradient

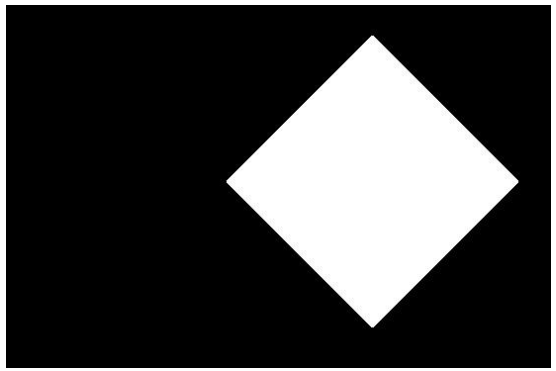
High gradient

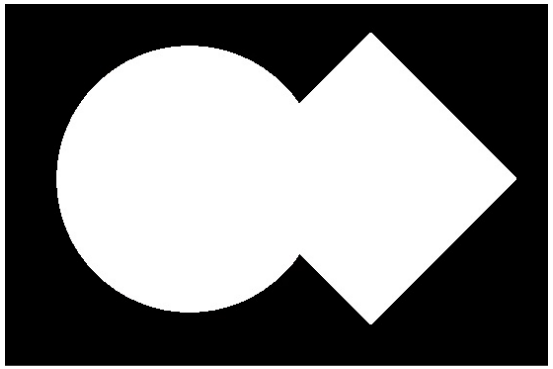
:

Bitwise Operations

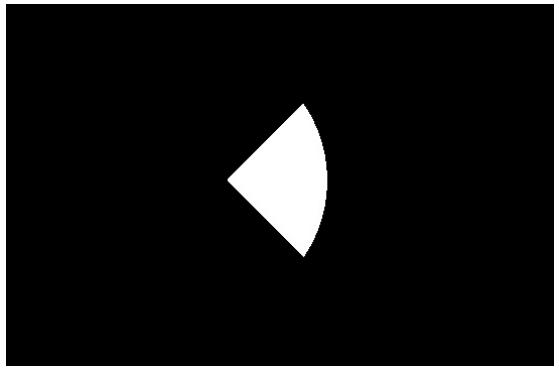
The following bitwise operations can be used



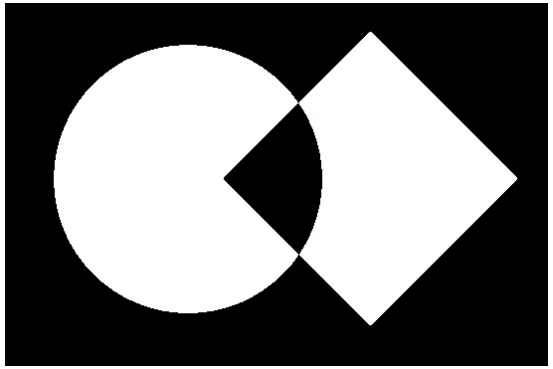




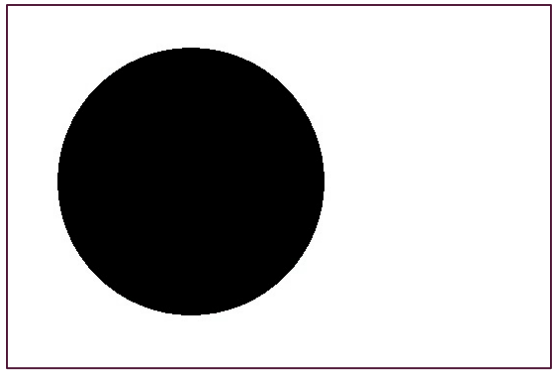
OR



AND



XOR



NOT

Bitwise Operations

The following bitwise operations can be used

```
img3 = cv2.bitwise_and(img1,img2)
img4 = cv2.bitwise_or(img1,img2)
img5 = cv2.bitwise_xor(img1,img2)
img6 = cv2.bitwise_not(img1)
```

In the first three functions, both images must be the same size.
The bitwise operations are done between corresponding pixels.

Each pixel can be considered to be 0 or 1.
The 0 pixel has an intensity of 0.
The 1 pixel has an intensity greater than 0.

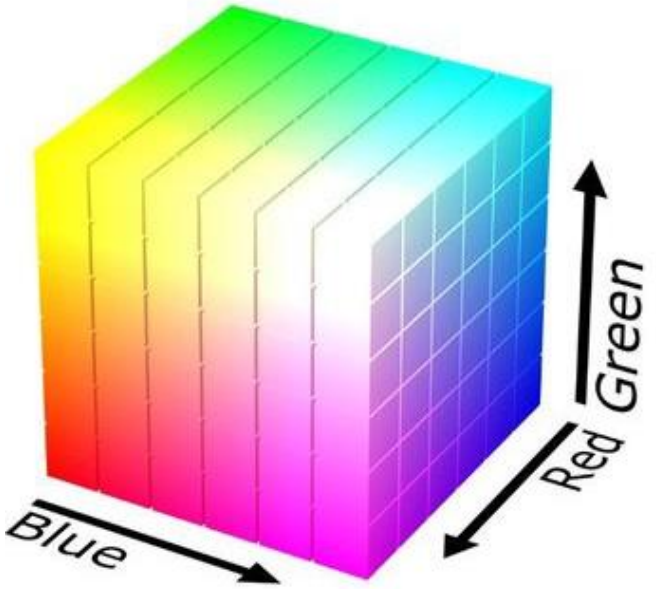
Color Spaces

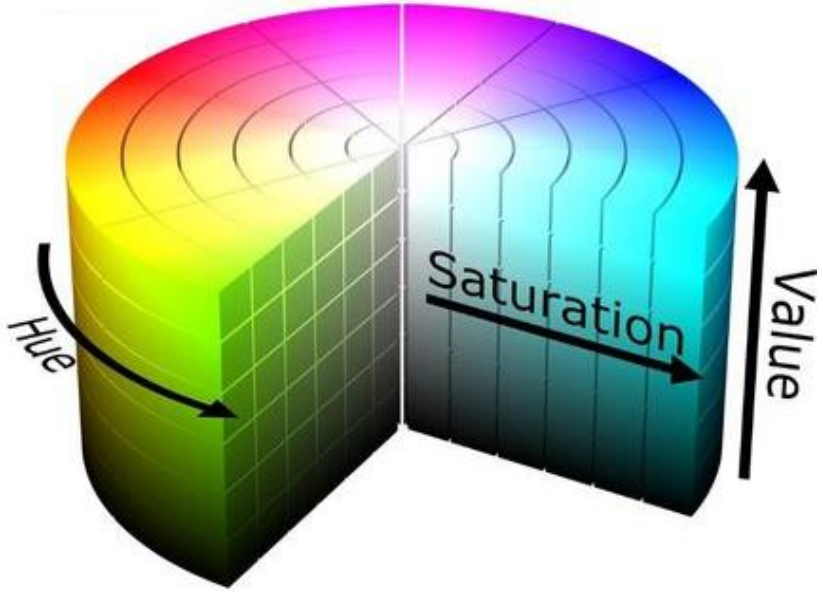
To change the color space of image from BGR to HSV:

```
img_hsv = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
```

img_hsv will now have (H,S,V) values instead of (B,G,R)

:





https://medium.com/neurosapiens/segmentation-and-classification-with-hsv-8f2406c62b39

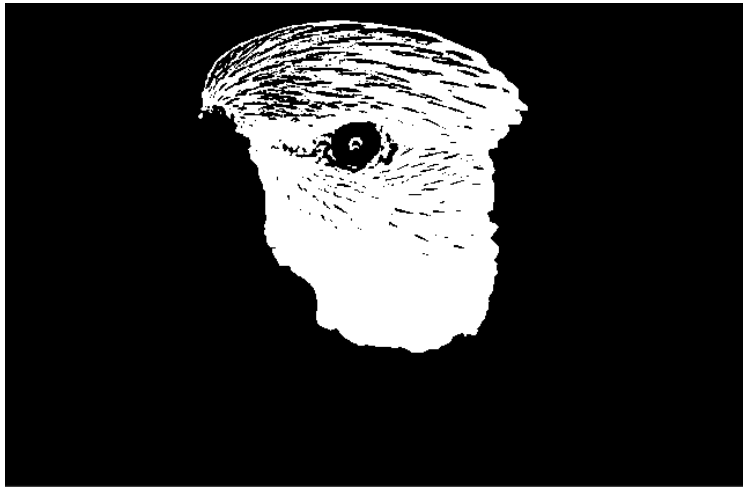
InRange Function

We can use the **inRange** function to get pixels that have certain values.

```
img2 = cv2.inRange(hsv, np.array([hmin,smin,vmin]),
np.array([hmax,smax,vmax]))
```

Values of H, S,V range as follows:
Hue: 0-179
Saturation: 0-255
Value : 0-255

:

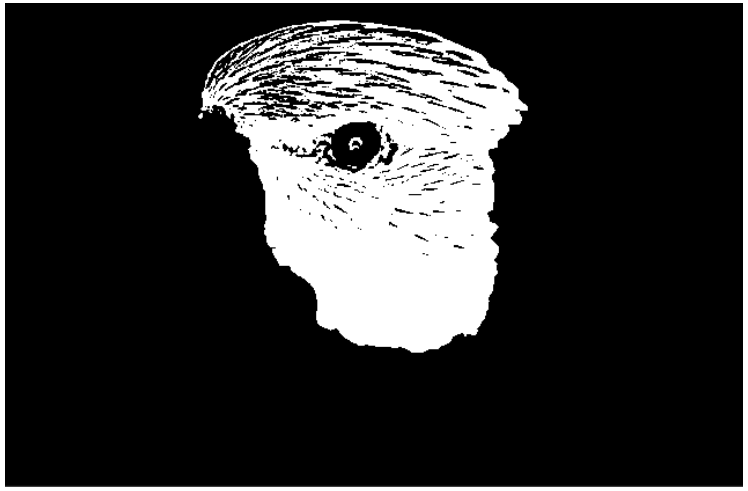


InRange Function

It is generally preferable to use HSV color space instead of BGR in order to get pixels that have a certain color

```
hsv = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
img2 = cv2.inRange(hsv, np.array([90,0,0]),
np.array([150,255,230]))
```

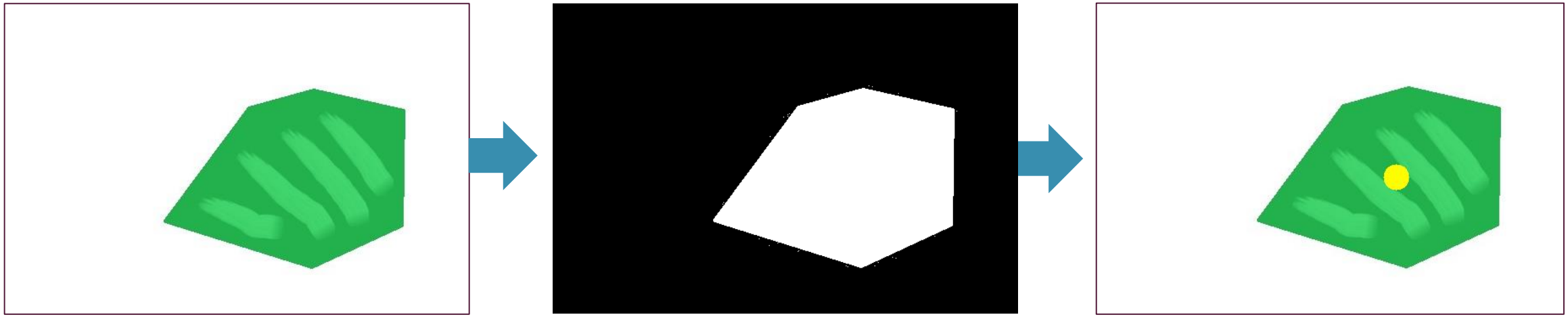
:



Centroids

To get the centroid point (cx, cy), we need a binary image

```
M = cv2.moments(img)
if M['m00'] > 0:
    cx = int(M['m10']/M['m00'])
    cy = int(M['m01']/M['m00'])
```



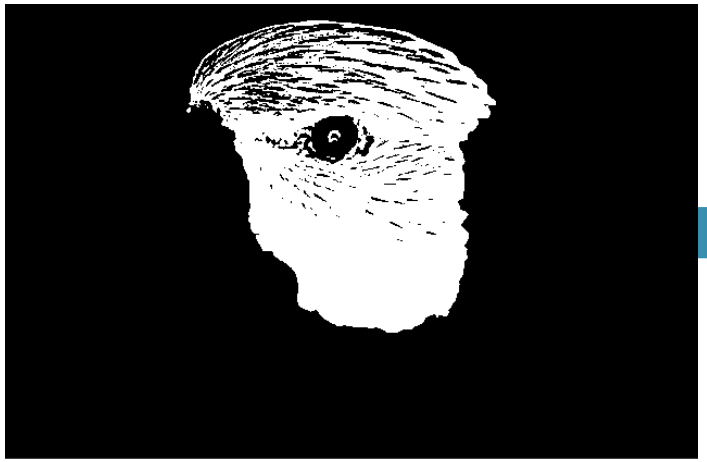
Original Image

Binary Image

Centroid in Image

Centroids

Original Image



Binary Image

Centroid in Image

Perspective Transformation

To do perspective transformation:

```
pts1 = np.float32([[x1,y1],[x2,y2],[x3,y3],[x4,y4]])
pts2 = np.float32([[x5,y5],[x6,y6],[x7,y7],[x8,y8]])
M = cv2.getPerspectiveTransform(pts1,pts2)
imgP = cv2.warpPerspective(img,M,(cols,rows))
```

Lab Tasks

- Download the materials from LMS
- Perform the Lab Tasks given in the manual
- Convert the completed manual into .pdf and submit on LMS