

Drawing on Images:

```
cv2.line(img, (0,0), (200,300), (255,0,0), 5) #Draw a blue line with thickness of 5
cv2.rectangle(img, (50, 50), (200, 300), (0, 255, 0), 3) #Draw a green rectangle with thickness of 3
cv2.circle(img, (200, 200), 50, (0,0,255), -1) #Draw a filled red circle
cv2.putText(img, 'Open CV', (100, 500), cv2.FONT_HERSHEY_SIMPLEX, 4, (255, 255, 255), 2, cv2.LINE_AA) #Add text
```

Feature Detection:

```
# identify distinct regions of an image that contain important information. Used for object
# recognition, image registration, and 3D reconstruction. Common methods: Harris corner
# detection, Shi-Tomasi corner detection, FAST, SURF, SIFT.
img = cv2.imread('image.jpg', 0) #Read image in grayscale
fast = cv2.FastFeatureDetector_create() ##FAST feature detection
keypoints_fast = fast.detect(img)
orb = cv2.ORB_create() ##ORB feature detection
keypoints_orb, descriptors = orb.detectAndCompute(img, None)
sift = cv2.xfeatures2d.SIFT_create() ##SIFT feature detection
keypoints_sift, descriptors = sift.detectAndCompute(img, None)
surf = cv2.xfeatures2d.SURF_create() ##SURF feature detection
keypoints_surf, descriptors = surf.detectAndCompute(img, None)
cv2.drawKeypoints(img, keypoints_fast, img, color= (255, 0, 0))
cv2.drawKeypoints(img, keypoints_orb, img, color=(0, 255, 0))
cv2.drawKeypoints(img, keypoints_sift, img, color=(0, 0, 255))
cv2.drawKeypoints(img, keypoints_surf, img, color= (255, 255, 0))
```

Perspective Transformation:

```
img = cv2.imread('image.jpg', cv2.IMREAD_COLOR)
pts1 = np.float32([ [56, 65], [368, 52], [28, 387], [389, 390] ]) #Four points on original image
pts2 = np.float32([ [0, 0], [300, 0], [0, 300], [300, 300] ]) #Four points on destination image
M = cv2.getPerspectiveTransform(pts1, pts2) #Perspective transformation matrix
dst = cv2.warpPerspective(img, M, (300, 300)) #Apply perspective transformation
cv2.imshow('image', dst)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Feature Description:

```
# extract numerical representation of detected feature in image. Important for object
# recognition and image matching. Common methods: BRIEF, ORB, SIFT, SURF.
img = cv2.imread('image.jpg', 0) # Read image in grayscale
fast = cv2.FastFeatureDetector_create() #FAST feature detection
brief = cv2.xfeatures2d.BriefDescriptorExtractor_create() #BRIEF descriptor
keypoints_fast = fast.detect(img)
keypoints_fast, descriptors_fast = brief.compute(img, keypoints_fast)
orb = cv2.ORB_create() #ORB feature detection and descriptor
keypoints_orb, descriptors_orb = orb.detectAndCompute(img, None)
```



Feature Description: (cont)

```
> sift = cv2.xfeatures2d.SIFT_create() #SIFT feature detection and descriptor
keypoints_sift, descriptors_sift = sift.detectAndCompute(img, None)
surf = cv2.xfeatures2d.SURF_create() #SURF feature detection and descriptor
keypoints_surf, descriptors_surf = surf.detectAndCompute(img, None)
```

Haar Cascade Classifiers (Object Detection):

```
# object detection using image patterns. Trains on features of positive and negative samples.
# Identifies object in new images by matching pattern. Can detect faces, eyes, and other objects.
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml') #Load face cascade classifier
img = cv2.imread('image.jpg', cv2.IMREAD_COLOR)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #Convert to grayscale
faces = face_cascade.detectMultiScale(gray, 1.3, 5) #Detect faces
for (x,y,w,h) in faces:
    cv2.rectangle(img, (x, y), (x+w,y+h), (255, 0,0),2) #Draw rectangle around face
```

Contour Detection:

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #Convert to grayscale
ret, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY) #Threshold
contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE) #Find contours
cv2.drawContours(img, contours, -1, (0,255,0), 3) #Draw contours
```

Loading and Saving Images:

```
img = cv2.imread('image.jpg', cv2.IMREAD_COLOR) #Load image
cv2.imshow('image',img) #Show image
cv2.imwrite('output.jpg',img) #Save image
cv2.waitKey(0) #Wait for a keyboard event
cv2.destroyAllWindows() #Close all windows
```

Image Properties:

```
img = cv2.imread('image.jpg', cv2.IMREAD_COLOR)
print('Image Shape:', img.shape) #Prints (height, width, channels)
print('Image Size:', img.size) #Prints the total number of pixels
print('Image Datatype:', img.dtype) #Prints the datatype of the pixels
```

Accessing a Camera/ Video Capture:

```
cap = cv2.VideoCapture(0) #Open default camera
while( True):
    ret, frame = cap.read() #Read frame from camera
    cv2.imshow('frame',frame) #Show frame
    if cv2.waitKey(1) & 0xFF == ord('q'): #Quit when 'q' is pressed
        break
```

Accessing a Video File:

```
cap = cv2.VideoCapture('video.mp4') #Open video file
while( cap.isOpened()):
    ret, frame = cap.read() #Read frame from video
    cv2.imshow('frame',frame) #Show frame
    if cv2.waitKey(25) & 0xFF == ord('q'): #Quit when 'q' is pressed
        break
```

Video Writer:

```
cap = cv2.VideoCapture(0)
fourcc = cv2.VideoWriter_fourcc(* 'XVID') #Codec
out = cv2.VideoWriter('output.avi',fourcc, 20.0, (640,480)) #Create video writer
while( cap.isOpened()):
    ret, frame = cap.read() #Capture frame- by- frame
    if ret==True:
        out.write(frame) #Write frame to output video
        cv2.imshow('frame',frame) #Display frame
        if cv2.waitKey(1) & 0xFF == ord('q'): #Exit loop on 'q' key press
            break
    else:
        break
cap.release() #Release camera
out.release() #Release video writer
```



Video Writer: (cont)

```
> cv2.destroyAllWindows()
```

Colorspace Conversion:

```
img = cv2.imread('image.jpg', cv2.IMREAD_COLOR)
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #Convert to grayscale
hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV) #Convert to HSV colorspace
```

Blurring:

```
# reduces image noise and sharpness by averaging pixel values in a neighborhood.
gaussian_blur = cv2.GaussianBlur(img, (5, 5), 0)
median_blur = cv2.medianBlur(img, 5)
bilateral_filter = cv2.bilateralFilter(img, 9, 75, 75)
```

Image Filtering:

```
# modifies pixel values to improve image quality. Common methods include 2D convolution,
#blurring, and edge detection.
img = cv2.imread('image.jpg')
kernel = np.ones((5,5), np.float32)/25
convolution = cv2.filter2D(img, -1, kernel) #2D Convolution
blur = cv2.blur(img, (5,5)) # Image Blurring (averaging)
gaussian_blur = cv2.GaussianBlur(img, (5,5), 0) # Gaussian Blurring
median_blur = cv2.medianBlur(img, 5) # Median Blurring
bilateral_filter = cv2.bilateralFilter(img, 9, 75, 75) #Bilateral Filtering
```

Edge Detection:

```
img = cv2.imread('image.jpg', 0) # Read image in grayscale
# Canny edge detection
edges_canny = cv2.Canny(img, 100, 200)
# Sobel operator
edges_sobel = cv2.Sobel(img, cv2.CV_64F, 1, 0) + cv2.Sobel(img, cv2.CV_64F, 0, 1)
# Laplacian of Gaussian (LoG)
edges_log = cv2.Laplacian(cv2.GaussianBlur(img, (3, 3), 0), cv2.CV_64F)
cv2.imshow('Original', img)
cv2.imshow('Canny Edges', edges_canny)
cv2.imshow('Sobel Edges', edges_sobel)
cv2.imshow('LoG Edges', edges_log)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Morphological Operations:

```
kernel = np.ones((5,5),np.uint8) #5x5 kernel
erosion = cv2.erode(img, kernel, iterations = 1) #Erosion
dilation = cv2.dilate(img, kernel, iterations = 1) #Dilation
opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel) #Opening
closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel) #Closing
```

Histogram Equalization:

```
# improves contrast by redistributing pixel values in an image.
# It's useful for image preprocessing and analysis
eq_img = cv2.equalizeHist(img) #Apply histogram equalization
```

Hough Transform:

```
# detect simple shapes such as lines, circles, and ellipses in an image
img = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)
edges = cv2.Canny(img, 50, 150, apertureSize = 3) #Canny edge detection
lines = cv2.HoughLines(edges, 1, np.pi/180, 200) #Hough transform
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
```

Thresholding:

```
# Thresholding separates object from background by setting pixel values above or below a
# threshold. It creates a binary image. Helps with object detection, segmentation, and feature
# extraction. Methods: global, adaptive, and Otsu's.
img = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)
ret, thresh1 = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY) #Binary thresholding
ret, thresh2 = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY_INV) #Inverse binary thresholding
ret, thresh3 = cv2.threshold(img, 127, 255, cv2.THRESH_TRUNC) #Truncated thresholding
ret, thresh4 = cv2.threshold(img, 127, 255, cv2.THRESH_TOZERO) #Threshold to zero
ret, thresh5 = cv2.threshold(img, 127, 255, cv2.THRESH_TOZERO_INV) #Inverse threshold to zero
adaptive_thresh = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2) #Adaptive thresholding
```

Image Pyramids:

```
img = cv2.imread('image.jpg', cv2.IMREAD_COLOR)
lower_reso = cv2.pyrDown(img) #Reduce resolution
higher_reso = cv2.pyrUp(img) #Increase resolution
```

