

Accessing and Modifying pixel values

Pixel value `img[100,100]`

Accessing only blue pixel `img[100,100,0]`

Modifying A Pixel `img[100,100] = [255,255,255]`

Better pixel accessing `img.item(10,10,2)`

Better pixel modifying `img.itemset((10,10,2),100)`

Access image properties `img.shape`

Total number of pixels `img.size`

Image datatype `img.dtype`

Getting ROI `ball = img[280:340, 330:390]`

Setting ROI `img[273:333, 100:160] = ball`

Split Channels `b,g,r = cv2.split(img)`
`b = img[:, :, 0]`

Making Borders for Images `cv2.copyMakeBorder(img1, 10,10,10,10,cv2.BORDER_REPLICATE)`

borderType `cv2.BORDER_CONSTANT`
`cv2.BORDER_REFLECT`
`cv2.BORDER_REFLECT_101`
`cv2.BORDER_REPLICATE`
`cv2.BORDER_WRAP`

Arithmetic Operations on Images

Image Addition (OPENCV) `print cv2.add(x,y) # 250+10 = 260 => 255`

Image Addition (Numpy) `print x+y # 250+10 = 260 % 256 = 4`

Image Alpha Blending `dst = cv2.addWeighted(img1,0.7,img2,0.3,0)`

Bitwise AND `img1_bg = cv2.bitwise_and(roi,roi,mask = mask_inv)`

Bitwise NOT `mask_inv = cv2.bitwise_not(mask)`

Morphological Transformations

Erosion `erosion = cv2.erode(img,kernel,iterations = 1)`

Dilation `dilation = cv2.dilate(img,kernel,iterations = 1)`

Opening `opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)`

Closing `closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)`

Morphological Gradient `gradient = cv2.morphologyEx(img, cv2.MORPH_GRADIENT, kernel)`

Top Hat `tophat = cv2.morphologyEx(img, cv2.MORPH_TOPHAT, kernel)`

Black Hat `blackhat = cv2.morphologyEx(img, cv2.MORPH_BLACKHAT, kernel)`

Morphological Transformations (cont)

Create Structuring Elements `cv2.getStructuringElement(cv2.MORPH_RECT, (5,5))`
`cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5,5))`
`cv2.getStructuringElement(cv2.MORPH_CROSS, (5,5))`

Performance Measurement and Improvement Techniques

Find # of clock-cycles `e1 = cv2.getTickCount()`
`# your code execution`
`e2 = cv2.getTickCount()`
`time = (e2 - e1) / cv2.getTickFrequency()`

Find clock cycles per second `cv2.getTickFrequency()`

Enable Optimizations `cv2.setUseOptimized(True)`

Measure Performance (IPython) `%timeit y=x**2`



By [thatguyandy27](#)

Not published yet.
Last updated 17th September, 2017.
Page 1 of 3.

Sponsored by [ApolloPad.com](#)
Everyone has a novel in them. Finish Yours!
<https://apollopad.com>

Performance Measurement and Improvement Techniques (cont)

Performance Optimization Techniques

1. Avoid using loops in Python as far as possible, especially double/triple loops etc. They are inherently slow.
2. Vectorize the algorithm/code to the maximum possible extent because Numpy and OpenCV are optimized for vector operations.
3. Exploit the cache coherence.
4. Never make copies of array unless it is needed. Try to use views instead. Array copying is a costly operation.

Geometric Transformations of Images

Scaling Types

```
cv2.INTER_AREA
cv2.INTER_CUBIC
cv2.INTER_LINEAR
```

Scaling

```
res =
cv2.resize(img, (2width, 2height), interpolation =
cv2.INTER_CUBIC)
```

Geometric Transformations of Images (cont)

Shifting (100 x 50)

```
M =
np.float32([[1, 0, 100],
[0, 1, 50]])
dst =
cv2.warpAffine(img, M, (cols, rows))
```

Rotation

```
M =
cv2.getRotationMatrix2D((cols/2, rows/2), 90, 1)
dst =
cv2.warpAffine(img, M, (cols, rows))
```

Affine Transformation

```
pts1 =
np.float32([[50, 50],
[200, 50], [50, 200]])
pts2 =
np.float32([[10, 100], [200, 50], [100, 250]])
M =
cv2.getAffineTransform(pts1, pts2)
dst =
cv2.warpAffine(img, M, (cols, rows))
```

Perspective Transformation

```
pts1 =
np.float32([[56, 65],
[368, 52], [28, 387],
[389, 390]])
pts2 =
np.float32([[0, 0], [300, 0], [0, 300], [300, 300]])
M =
cv2.getPerspectiveTransform(pts1, pts2)
dst =
cv2.warpPerspective(img, M, (300, 300))
```

Canny Edge Detection

```
Canny Detection edges =
cv2.Canny(img, 100, 200)
```

Image Pyramids

Lower Gaussian Pyramid

```
lower_reso =
cv2.pyrDown(higher_reso)
```

Higher Gaussian Pyramid

```
higher_reso2 =
cv2.pyrUp(lower_reso)
```

Pyramid Blending

1. Load the two images
2. Find the Gaussian Pyramids
3. From Gaussian Pyramids, find their Laplacian Pyramids
4. Now each levels of Laplacian Pyramids
5. Finally from this joint image pyramids, reconstruct the original image

Changing Colorspaces

List Colorspace Flags (150+)

```
flags = [i for i in
dir(cv2) if
i.startswith('COLOR_')]
```

Convert to Gray

```
img_gray =
cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Convert to hsv

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



By [thatguyandy27](#)

cheatography.com/thatguyandy27/

Not published yet.

Last updated 17th September, 2017.

Page 2 of 3.

Sponsored by **ApolloPad.com**

Everyone has a novel in them. Finish Yours!

<https://apollopad.com>

Changing Colorspaces (cont)

```
Track Blue (color) Object lower_blue = np.array([110,50,50]) upper_blue = np.array([130,255,255]) mask = cv2.inRange(hsv, lower_blue, upper_blue) res = cv2.bitwise_and(frame, frame, mask= mask)
```

```
Find HSV Color green = np.uint8([[0,255,0]]) hsv_green = cv2.cvtColor(green,cv2.COLOR_BGR2HSV)
```

Image Thresholding

```
Thresholding Types cv2.THRESH_BINARY cv2.THRESH_BINARY_INV cv2.THRESH_TRUNC cv2.THRESH_TOZERO cv2.THRESH_TOZERO_INV
```

```
Getting Threshold ret, thresh4 = cv2.threshold(img,127,255, cv2.THRESH_TOZERO)
```

```
Adaptive Method Types cv2.ADAPTIVE_THRESH_MEAN_C cv2.ADAPTIVE_THRESH_GAUSSIAN_C
```

```
Adaptive Threshold th3 = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)
```

Image Thresholding (cont)

```
Otsu's Binarization ret3, th3 = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)
```

Smoothing Images

```
Convolve an Image dst = cv2.filter2D(img, -1, kernel)
```

```
Box (averaging) blur = cv2.blur(img, (5,5)) Filtering cv2.boxFilter()
```

```
Create Gaussian Kernel cv2.getGaussianKernel(size, sigma, type)
```

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

```
Median Blur median = cv2.medianBlur(img, 5)
```

```
Bilateral Blur blur = cv2.bilateralFilter(img, 9, 75, 75)
```

Image Gradients

```
Sobel sobelx = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=5)
```

```
Laplacian laplacian = cv2.Laplacian(img, cv2.CV_64F)
```

*Output datatype cv2.CV_8U or np.uint8. So when you convert data to np.uint8, all negative slopes are made zero. In simple words, you miss that edge. If you want to detect both edges, better option is to keep the output datatype to some higher forms, like cv2.CV_16S, cv2.CV_64F etc, take its absolute value and then convert back to cv2.CV_8U



By [thatguyandy27](#)

cheatography.com/thatguyandy27/

Not published yet.

Last updated 17th September, 2017.

Page 3 of 3.

Sponsored by [ApolloPad.com](#)

Everyone has a novel in them. Finish Yours!

<https://apollopad.com>