Master OpenCV 1

Created By	FREDDIE MAR MAÑEGOS
Last Edited	@Feb 10, 2020 9:31 AM
Tags	Artificial Intelligence Computer Vision

2.4 - Reading, writing and displaying images

cv2.imread → load an image using 'imread' specifying the path to image
cv2.imshow → display image variable (title of the image window, image variable)
cv2.waitkey → allows us to input information when a image window is open.
cv2.destroyAllWindows() → closes all open windows
cv2.imwrite() → specifying the filename and the image to be saved

```
# We don't need to do this again, but it's a good habit
import cv2

# Load an image using 'imread' specifying the path to image
input = cv2.imread('./images/input.jpg')

# Our file 'input.jpg' is now loaded and stored in python
# as a varaible we named 'image'

# To display our image variable, we use 'imshow'
# The first parameter will be title shown on image window
# The second parameter is the image varialbe
cv2.imshow('Hello World', input)

# 'waitKey' allows us to input information when a image window is open
# By leaving it blank it just waits for anykey to be pressed before
# continuing. By placing numbers (except 0), we can specify a delay for
# how long you keep the window open (time is in milliseconds here)
cv2.waitKey()

# This closes all open windows
# Failure to place this will cause your program to hang
cv2.destroyAllWindows()
```

2.5 - Grayscaling

Grayscaling is process by which an image is converted from a full color to shades of grey (black & white)

In OpenCV, many functions grayscale images before processing. This is done because it simplifies the image, acting almost as a noise reduction and increasing processing time as there is less information in the image.

gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

```
import cv2

# Load our input image
image = cv2.imread('./images/input.jpg')
cv2.imshow('Original', image)
cv2.waitKey()

# We use cvtColor, to convert to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

cv2.imshow('Grayscale', gray_image)
cv2.waitKey()
cv2.destroyAllWindows()
```

```
import cv2
#Another method faster method
img = cv2.imread('./images/input.jpg',0)

cv2.imshow('Grayscale', img)
cv2.waitKey()
cv2.destroyAllWindows()
```

2.6 - Color Spaces

RGB, HSV or CMYK are color spaces, which simply a way to represent color.

OpenCV default color is RGB. Stores color in BGR format.

- we use BGR order on computers due to how unsigned 32-bit integers are stored in memory.

HSV (Hue, Saturation & Value/Brightness) is a color space that attempts to represent colors the way humans perceive it.

Hue - Color Value (0-179)

It is mapped differently than standard.

Color Range Filters:

- Red - 165 to 15
- Green - 45 to 75
- Blue - 90 to 120

Saturation - Vibrancy of color (0-255)

Value - Brightness or intensity (0-255)

2.7 Histograms

Histograms are a great way to visualize individual color components

NOTE:

The first time you run this block the diagrams won't be displayed, you will need to run this a second time!

```
import cv2
import numpy as np

# We need to import matplotlib to create our histogram plots
from matplotlib import pyplot as plt

image = cv2.imread('images/input.jpg')

histogram = cv2.calcHist([image], [0], None, [256], [0, 256])

# We plot a histogram, ravel() flatens our image array
plt.hist(image.ravel(), 256, [0, 256]); plt.show()

# Viewing Separate Color Channels
color = ('b', 'g', 'r')

# We now separate the colors and plot each in the Histogram
for i, col in enumerate(color):
    histogram2 = cv2.calcHist([image], [i], None, [256], [0, 256])
    plt.plot(histogram2, color = col)
    plt.xlim([0,256])

plt.show()
```

cv2.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]])

- **images**: it is the source image of type uint8 or float32. it should be given in square brackets, ie, "[img]".
- **channels**: it is also given in square brackets. It is the index of channel for which we calculate histogram. For example, if input is grayscale image, its value is [0]. For color image, you can pass [0], [1] or [2] to calculate histogram of blue, green or red channel respectively.
- **mask**: mask image. To find histogram of full image, it is given as "None". But if you want to find histogram of particular region of image, you have to create a mask image for that and give it as mask. (I will show an example later.)
- **histSize**: this represents our BIN count. Need to be given in square brackets. For full scale, we pass [256].
- **ranges** : this is our RANGE. Normally, it is [0,256].

2.8 Drawing Images

Read more::

```
https://docs.opencv.org/master/dc/da5/tutorial_py_drawing_functions.html
```

3 - Image Manipulation

Transformations, affine and non affine
Translations
Rotations
Scaling, re-sizing and interpolations
Image Pyramids
Cropping
Arithmetic Operations
Bitwise Operations and Masking
Convolutions & Blurring
Sharpening
Thresholding and Binarization
Dilation, erosion, opening and closing
Edge Detection & Image Gradients
Perspective & Affine Transforms

3.1 Transformations

Are geometric distortions enacted upon an image.

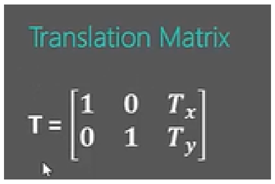
We use transformations to correct distortions or perspective issues from arising from the point of view an image was captured.

Affine VS Non-Affine

Affine - Scaling, Rotation, Translation

Non-Affine or Projective Transform and also called Homography. Does not preserve parallelism, length and angle.

3.2 Translation


Translation Matrix

$$T = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \end{bmatrix}$$

Tx - Represents the shift along the x-axis (horizontal)

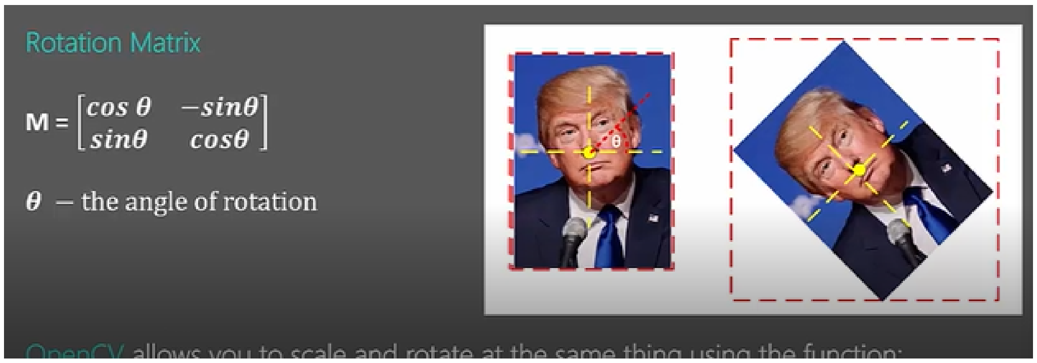
Ty - Represents the shift along the y-axis (vertical)

In OpenCV function cv2.warpAffine to implement these relations.

```
import cv2
import numpy as np

image = cv2.imread('images/input.jpg')

# Store height and width of the image
height, width = image.shape[:2]

quarter_height, quarter_width = height/4, width/4

#       | 1 0 Tx |
#  T  = | 0 1 Ty |

# T is our translation matrix
T = np.float32([[1, 0, quarter_width], [0, 1,quarter_height]])

# We use warpAffine to transform the image using the matrix, T
img_translation = cv2.warpAffine(image, T, (width, height))
cv2.imshow('Translation', img_translation)
cv2.waitKey()
cv2.destroyAllWindows()
```

3.3 Rotations

Rotation Matrix


Rotation Matrix

$$M = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

θ – the angle of rotation

OpenCV allows you to scale and rotate at the same thing using the function:

OpenCV allows you to scale and rotate at the same thing using the function.

```
cv2.getRotationMatrix2D(rotation_center_x, rotation_center_y, angle of rotation, scale)
```

```
import cv2
import numpy as np

image = cv2.imread('images/input.jpg')
height, width = image.shape[:2]

# Divide by two to rototate the image around its centre
rotation_matrix = cv2.getRotationMatrix2D((width/2, height/2), 90, .5)

rotated_image = cv2.warpAffine(image, rotation_matrix, (width, height))

cv2.imshow('Rotated Image', rotated_image)
cv2.waitKey()
cv2.destroyAllWindows()
```

### Notice all the black space surrounding the image.

We could now crop the image as we can calculate it's new size (we haven't learned cropping yet!).

But here's another method for simple rotations that uses the cv2.transpose function

3.4 Re-sizing, Scaling and Interpolation

What is interpolation?
Interpolation is a method of constructing new data points within the range of a discrete set of known data points
v2.INTER_AREA - Good for shrinking or down sampling
v2.INTER_NEAREST - Fastest
v2.INTER_LINEAR - Good for zooming or up sampling (default)
v2.INTER_CUBIC - Better
v2.INTER_LANCZOS4 - Best
Good comparison of interpolation methods:
http://tanbakuchi.com/posts/comparison-of-openv-interpolation-algorithms

Sample code:

```
import cv2
import numpy as np

# load our input image
image = cv2.imread('images/input.jpg')

# Let's make our image 3/4 of it's original size
image_scaled = cv2.resize(image, None, fx=0.75, fy=0.75)
cv2.imshow('Scaling - Linear Interpolation', image_scaled)
cv2.waitKey()

# Let's double the size of our image
img_scaled = cv2.resize(image, None, fx=2, fy=2, interpolation = cv2.INTER_CUBIC)
cv2.imshow('Scaling - Cubic Interpolation', img_scaled)
```

```
cv2.waitKey()

# Let's skew the re-sizing by setting exact dimensions
img_scaled = cv2.resize(image, (900, 400), interpolation = cv2.INTER_AREA)
cv2.imshow('Scaling - Skewed Size', img_scaled)
cv2.waitKey()

cv2.destroyAllWindows()
```

3.5 Image Pyramids

Pyramiding image refers to either upscaling (enlarging) and downscaling (shrinking images).

It’s simply a different way of re-sizing that allows us to easily and quickly scale images. Scaling do reduces the height and width of the new image by half.

This comes in useful when making object detectors that scales images each time it looks for an.

```
import cv2

image = cv2.imread('images/input.jpg')

smaller = cv2.pyrDown(image)
larger = cv2.pyrUp(smaller)

cv2.imshow('Original', image )

cv2.imshow('Smaller ', smaller )
cv2.imshow('Larger ', larger )
cv2.waitKey(0)
cv2.destroyAllWindows()
```

3.6 Cropping Images

Cropping images refers to extracting a segment a of that image.

```
import cv2
import numpy as np

image = cv2.imread('images/input.jpg')
height, width = image.shape[:2]

# Let's get the starting pixel coordiantes (top  left of cropping rectangle)
start_row, start_col = int(height * .25), int(width * .25)

# Let's get the ending pixel coordinates (bottom right)
end_row, end_col = int(height * .75), int(width * .75)

# Simply use indexing to crop out the rectangle we desire
cropped = image[start_row:end_row , start_col:end_col]

cv2.imshow("Original Image", image)
cv2.waitKey(0)
cv2.imshow("Cropped Image", cropped)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

3.7 Convolutions & Blurring

A **Convolution** is a mathematical operation performed on two functions producing a third function which is typically a modified version of one of the original functions.

In Computer Vision we use kernel's to specify the size over which
we run our manipulating function over our image

Blurring is an operation where we average the pixels within a region
(kernel).

Below is a 5×5 kernel. We multiply by 1/25 to normalize i.e. sum to 1, otherwise we'd be increasing intensity.

$$Kernel\_ = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

cv2.filter2D(image, -1, kernel)

cv2.blur – Averages values over a specified window
cv2.GaussianBlur – Similar, but uses a Gaussian window (more
emphasis or weighting on points around the center)
cv2.medianBlur – Uses median of all elements in the window
cv2.bilateralFilter – Blur while keeping edges sharp (slower). It also takes a Gaussian filter in space, but one more Gaussian filter which is a function of pixel difference. The pixel difference function makes sure only those pixels with similar intensity to central pixel is considered for blurring. So it preserves the edges since pixels at edges will have large intensity variation.

```
import cv2
import numpy as np

image = cv2.imread('images/elephant.jpg')
cv2.imshow('Original Image', image)
cv2.waitKey(0)

# Creating our 3 x 3 kernel
kernel_3x3 = np.ones((3, 3), np.float32) / 9

# We use the cv2.fitler2D to conovlve the kernal with an image
blurred = cv2.filter2D(image, -1, kernel_3x3)
cv2.imshow('3x3 Kernel Blurring', blurred)
cv2.waitKey(0)

# Creating our 7 x 7 kernel
kernel_7x7 = np.ones((7, 7), np.float32) / 49

blurred2 = cv2.filter2D(image, -1, kernel_7x7)
cv2.imshow('7x7 Kernel Blurring', blurred2)
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```

```
import cv2
import numpy as np

image = cv2.imread('images/elephant.jpg')

# Averaging done by convolving the image with a normalized box filter.
# This takes the pixels under the box and replaces the central element
# Box size needs to odd and positive
blur = cv2.blur(image, (3,3))
cv2.imshow('Averaging', blur)
cv2.waitKey(0)

# Instead of box filter, gaussian kernel
Gaussian = cv2.GaussianBlur(image, (7,7), 0)
cv2.imshow('Gaussian Blurring', Gaussian)
cv2.waitKey(0)

# Takes median of all the pixels under kernel area and central
# element is replaced with this median value
median = cv2.medianBlur(image, 5)
cv2.imshow('Median Blurring', median)
cv2.waitKey(0)

# Bilateral is very effective in noise removal while keeping edges sharp
bilateral = cv2.bilateralFilter(image, 9, 75, 75)
cv2.imshow('Bilateral Blurring', bilateral)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Image De-noising - Non-Local Means Denoising

```
import numpy as np
import cv2

image = cv2.imread('images/elephant.jpg')

# Parameters, after None are - the filter strength 'h' (5-10 is a good range)
# Next is hForColorComponents, set as same value as h again
#
dst = cv2.fastNlMeansDenoisingColored(image, None, 6, 6, 7, 21)

cv2.imshow('Fast Means Denoising', dst)
cv2.waitKey(0)

cv2.destroyAllWindows()
```

There are 4 variations of Non-Local Means Denoising:

- cv2.fastNlMeansDenoising() - works with a single grayscale images.
- cv2.fastNlMeansDenoisingColored() - works with a color image.
- cv2.fastNlMeansDenoisingMulti() - works with image sequence captured in short period of time (grayscale images).

- cv2.fastNlMeansDenoisingColoredMulti() - same as above, but for color images.

3.8 Sharpening

Sharpening is the opposite of blurring, it strengthens or emphasizing edges in an image

$$Kernel\_ = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Our kernel matrix sums to one, so there is no need to normalize (i.e. multiply by a factor to retain the same brightness of the original)

By altering our kernels we can implement sharpening, which has the effects of in strengthening or emphasizing edges in an image.

```
import cv2
import numpy as np

image = cv2.imread('images/input.jpg')
cv2.imshow('Original', image)

# Create our shapening kernel, we don't normalize since the
# the values in the matrix sum to 1
kernel_sharpening = np.array([[-1,-1,-1],
                              [-1,9,-1],
                              [-1,-1,-1]])

# applying different kernels to the input image
sharpened = cv2.filter2D(image, -1, kernel_sharpening)

cv2.imshow('Image Sharpening', sharpened)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

3.9 Thresholding, Binarization & Adaptive Thresholding

In thresholding, we convert a grey scale image to it's binary form.

cv2.threshold(image, Threshold Value, Max Value, Threshold Type

Threshold Types:

cv2.THRESH_BINARY – Most common

cv2.THRESH_BINARY_INV – Most common

cv2.THRESH_TRUNC

cv2.THRESH_TOZERO

cv2.THRESH_TOZERO_INV

NOTE: Image need to be converted to greyscale before thresholding.

```
import cv2
import numpy as np

# Load our image as greyscale
image = cv2.imread('images/gradient.jpg',0)
cv2.imshow('Original', image)

# Values below 127 goes to 0 (black, everything above goes to 255 (white)
ret,thresh1 = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)
cv2.imshow('1 Threshold Binary', thresh1)

# Values below 127 go to 255 and values above 127 go to 0 (reverse of above)
ret,thresh2 = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY_INV)
cv2.imshow('2 Threshold Binary Inverse', thresh2)

# Values above 127 are truncated (held) at 127 (the 255 argument is unused)
ret,thresh3 = cv2.threshold(image, 127, 255, cv2.THRESH_TRUNC)
cv2.imshow('3 THRESH TRUNC', thresh3)

# Values below 127 go to 0, above 127 are unchanged
ret,thresh4 = cv2.threshold(image, 127, 255, cv2.THRESH_TOZERO)
cv2.imshow('4 THRESH TOZERO', thresh4)

# Resever of above, below 127 is unchanged, above 127 goes to 0
ret,thresh5 = cv2.threshold(image, 127, 255, cv2.THRESH_TOZERO_INV)
cv2.imshow('5 THRESH TOZERO INV', thresh5)
cv2.waitKey(0)

cv2.destroyAllWindows()
```

Adaptive Thresholding'

Simple threshold methods require us to provide the threshold value.

Adaptive threshold methods take that uncertainty away

c2.aptiveThreshold(image, Max Value, Adaptive type, Threshold Type, Block size, Constant that is subtracted from mean)

NOTE: Block sizes need to be odd numbers

Adaptive Threshold Types:

ADAPTIVE_THRESH_MEAN_C – based on mean of the neighborhood of pixels

ADAPTIVE_THRESH_GAUSSIAN_C – weighted sum of neighborhood pixels under the Gaussian window

THRESH_OTSU (uses cv2.threshold function) – Clever algorithm assumes there are two peaks in the gray scale histogram of the image and then tries to find an optimal value to separate these two peaks to find T

3.10 Dilation and Erosion

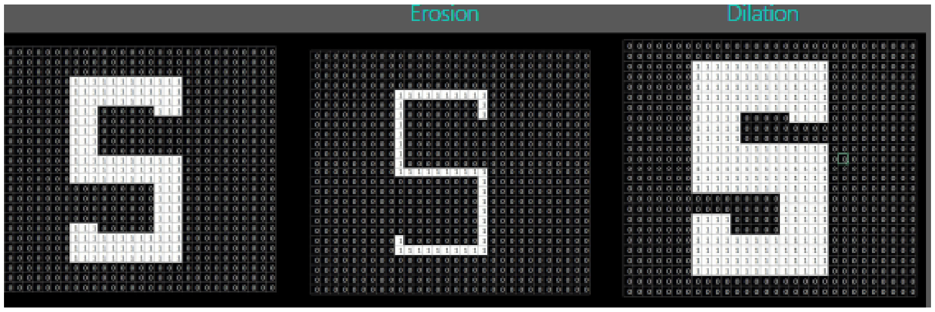
These are operations in the field of mathematical morphology:

Dilation – Adds pixels to the boundaries of objects in an image

Erosion – Removes pixels at the boundaries of objects in an image

Opening - Erosion followed by dilation

Closing - Dilation followed by erosion

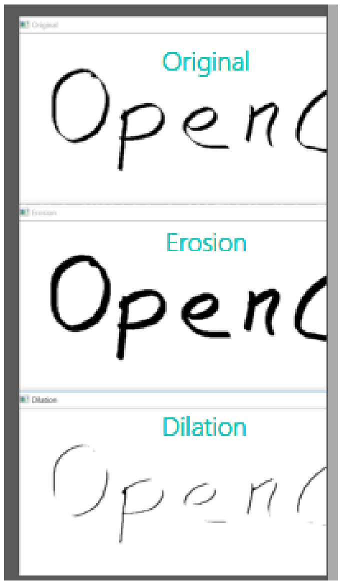


Confusion with Dilation and Erosion

Common StackOverflow question:"Why is dilation and erosion doing the reverse of what I expect?Remember:

Dilation – Adds pixels to the boundaries of objects in an image

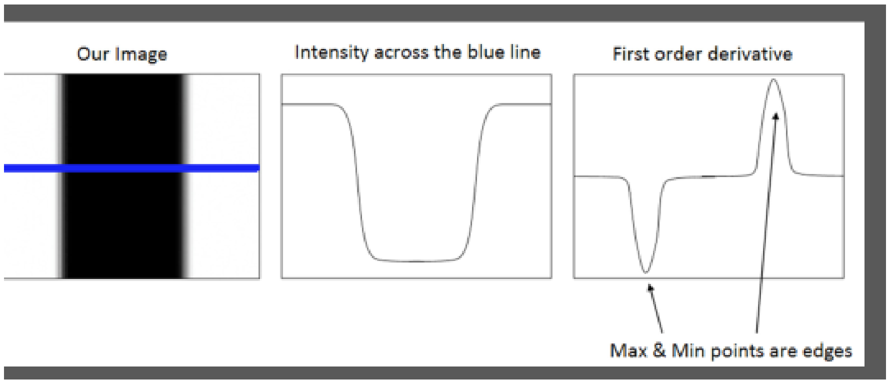
Erosion – Removes pixels at the boundaries of objects in an image

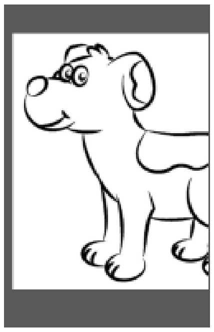


3.11 Edge Detection & Image Gradients

Edge Detection is a very important area in Computer Vision, especially when dealing with contour (you'll learn this later soon)

Edges can be defined as sudden changes (discontinuities) in an image and they can encode just much information as pixels.





There are three main types of Edge Detection:

- Sobel – to emphasize vertical or horizontal edges
- Laplacian – Gets all orientations
- Canny – Optimal due to low error rate, well defined edges and accurate detection

```
import cv2
import numpy as np

image = cv2.imread('images/input.jpg',0)

height, width = image.shape

# Extract Sobel Edges
sobel_x = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=5)
sobel_y = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=5)
```

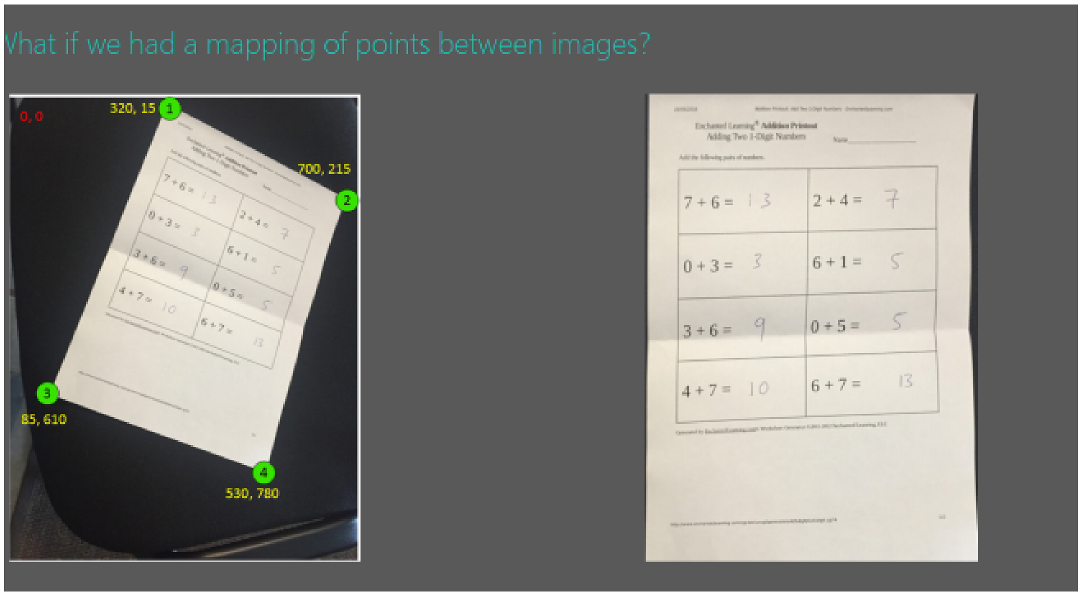
```
cv2.imshow('Original', image)
cv2.waitKey(0)
cv2.imshow('Sobel X', sobel_x)
cv2.waitKey(0)
cv2.imshow('Sobel Y', sobel_y)
cv2.waitKey(0)

sobel_OR = cv2.bitwise_or(sobel_x, sobel_y)
cv2.imshow('sobel_OR', sobel_OR)
cv2.waitKey(0)

laplacian = cv2.Laplacian(image, cv2.CV_64F)
cv2.imshow('Laplacian', laplacian)
cv2.waitKey(0)


##  Then, we need to provide two values: threshold1 and threshold2. Any gradient value larger than threshold2
# is considered to be an edge. Any value below threshold1 is considered not to be an edge.
#Values in between threshold1 and threshold2 are either classified as edges or non-edges based on how their
#intensities are “connected”. In this case, any gradient values below 60 are considered non-edges
#whereas any values above 120 are considered edges.


# Canny Edge Detection uses gradient values as thresholds
# The first threshold gradient
canny = cv2.Canny(image, 50, 120)
cv2.imshow('Canny', canny)
cv2.waitKey(0)

cv2.destroyAllWindows()
```

Canny Edge Detection Algorithm (developed by John F. Canny in 1986)

Applies Gaussian blurring.

- Finds intensity gradient of the image.
- Applied non-maximum suppression (i.e. removes pixels that are not edges).
- Hysteresis – Applies thresholds (i.e. if pixel is within the upper and lower thresholds, it is considered an edge)

3.12 Obtaining the Perspective of Affine Transformation



```
import cv2
import numpy as np
import matplotlib.pyplot as plt

image = cv2.imread('images/scan.jpg')

cv2.imshow('Original', image)
cv2.waitKey(0)

# Cordinates of the 4 corners of the original image
points_A = np.float32([[320,15], [700,215], [85,610], [530,780]])

# Cordinates of the 4 corners of the desired output
# We use a ratio of an A4 Paper 1 : 1.41
points_B = np.float32([[0,0], [420,0], [0,594], [420,594]])

# Use the two sets of four points to compute
# the Perspective Transformation matrix, M
M = cv2.getPerspectiveTransform(points_A, points_B)

warped = cv2.warpPerspective(image, M, (420,594))

cv2.imshow('warpPerspective', warped)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Mini-Project : Live Sketch Using Webcam

```
import cv2
import numpy as np

#-*- coding:utf-8 -*-
# Our sketch generating function
```

```
def sketch(image):
    # Convert image to grayscale
    img_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Clean up image using Guassian Blur
    img_gray_blur = cv2.GaussianBlur(img_gray, (5,5), 0)

    # Extract edges
    canny_edges = cv2.Canny(img_gray_blur, 10, 70)

    # Do an invert binarize the image
    ret, mask = cv2.threshold(canny_edges, 70, 255, cv2.THRESH_BINARY_INV)
    return mask


# Initialize webcam, cap is the object provided by VideoCapture
# It contains a boolean indicating if it was sucessful (ret)
# It also contains the images collected from the webcam (frame)
cap = cv2.VideoCapture(-1)

while True:
    ret, frame = cap.read()
    cv2.imshow('Our Live Sketcher', sketch(frame))
    if cv2.waitKey(1) == 13: #13 is the Enter Key
        break

# Release camera and close windows
cap.release()
cv2.destroyAllWindows()
```