

Module 1.3

OpenCV Basics

Basic Functions

Satya Mallick, Ph.D.

LearnOpenCV.com

Table of Contents

OpenCV Basic Functions	2
Read, write and display an image	2
Crop and Resize	5
Datatype conversion	18
Drawing over an image	20
C++ code for drawing over an image	21
Python code for drawing over an image	24
References and Further reading	27

OpenCV Basic Functions

In this section, we will go over some basic OpenCV functions that you will need in almost every tutorial. We assume you have already installed [OpenCV 3.2](#) with **opencv_contrib** or above for your platform. We will provide the most common usage of each function and highlight important parameters as and when required.

Read, write and display an image

In OpenCV, the function **imread** is used for reading an image, **imwrite** is used to write an image in memory to disk and **imshow** in conjunction with **namedWindow** and **waitKey** is used for displaying an image in memory.

The following C++ and Python codes provide a simple example. The code below is a tutorial with important comments highlighted.

C++ [Read, write and display image] [readWriteDisplay.cpp]

```
1 #include <opencv2/opencv.hpp>
2
3 using namespace cv;
4 using namespace std;
5
// This program reads an image using imread() from the images directory, converts
// it to gray scale using cvtColor() and displays it using imshow(). It also saves the
// image to the disk using the imwrite() function.
6 int main(void)
7 {
8
9 // Read image
10 Mat image = imread("../data/images/lena.jpg");
11
12 // Check for invalid input
13 if(image.empty())
14 {
15     cout << "Could not open or find the image" << endl;
16     return EXIT_FAILURE;
17 }
18
19 // Convert color image to gray
20 Mat grayImage;
21 cvtColor(image, grayImage, COLOR_BGR2GRAY);
22
23 // Save result
24 imwrite("imageGray.jpg", grayImage);
25
26 // Create a window for display
```

```

27 namedWindow("Image", WINDOW_AUTOSIZE);
28 namedWindow("Gray Image", WINDOW_NORMAL);
29
30 // Display image
31 imshow("Image", image);
32 imshow("Gray Image", grayImage);
33
34 // Wait for a keystroke in the window
35 waitKey(0);
36
37 return EXIT_SUCCESS;
38 }

```

Python [Read, write and display image] [readWriteDisplay.py]


```

1 import cv2
2
3 # This program reads an image using imread() from the images directory, converts
  it to gray scale using cvtColor() and displays it using imshow(). It also saves the
  image to the disk using the imwrite() function
4 image = cv2.imread("../data/images/lena.jpg")
5
6 # Check for invalid input
7 if image is None:
8     print("Could not open or find the image")
9
10 # Convert color image to gray
11 grayImage = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
12
13 # Save result
14 cv2.imwrite("imageGray.jpg", grayImage)
15
16 # Create a window for display.
17 cv2.namedWindow("Image", cv2.WINDOW_AUTOSIZE)
18 cv2.namedWindow("Gray Image", cv2.WINDOW_NORMAL)
19
20 # Display image
21 cv2.imshow("Image", image)
22 cv2.imshow("Gray Image", grayImage)
23
24 # Wait for a keystroke in the window
25 cv2.waitKey(0)

```

In C++ **imread** read the image into an instance of the Mat class, while in Python images are represented using a numpy array. While the usage of imread and imwrite is straightforward, the code for displaying an image requires some explanation.

In lines **27-28 (cpp)** and lines **17-18 (python)** we create two windows named “Image” and “Gray Image” to display the two images using **namedWindow**. Strictly speaking, this step is not necessary unless we want to specify the resizing property of the window using



WINDOW_AUTOSIZE and WINDOW_NORMAL. For larger images WINDOW_AUTOSIZE resizes the window to fit your screen.

Bug Alert : WINDOW_AUTOSIZE does not work as expected on OSX.

In lines **31-32 (cpp)** and lines **21-22 (python)** we display two images using **imshow** in windows “Image” and “Gray Image”. If a window with the specified name does not exist, it will be automatically created. Therefore, we will often skip using `namedWindow` in our code.

In line **35 (cpp)** and line **25 (python)**, the function **waitKey(n)** is used to wait for n seconds. When n=0, execution is paused until a key is pressed. This allows us to see the image before the program terminates.

Crop and Resize

In OpenCV, we can **resize** images. We can both shrink and enlarge images. Images are shrunk to perform certain operations on it and then once the operations are done they are enlarged back. Shrinking also helps to gain **computational speed**. We can also crop out images. This helps in selective computation as we do not need to process the entire image.

The following C++ and Python codes provide a simple example.

C++ [Crop and Resize] [cropAndResize.cpp]

//This program demonstrates how we can use the resize function of OpenCV to shrink and enlarge images. It then uses the range function to crop an image and extract a portion of the image.

```
1 #include <opencv2/opencv.hpp>
2
3 using namespace std;
4 using namespace cv;
5
6 int main(void)
7 {
8     Mat source, scaleDown, scaleUp;
9
10    // Read source image
11    source = imread("../data/images/lena.jpg",1);
12
13    // Scaling factors
14    double scaleX = 0.6;
15    double scaleY = 0.6;
16
```

The resize has the following prototype:

resize(sourceImage, destinationImage, Size(), scale factor in x direction, scale factor in y direction, interpolation method)

We can either specify the Size(,) of the output to determine the height and width of the output or we can add the scaling factors and the Size will be calculated automatically.

```
17 // Scaling down the image 0.6 times
18 cv::resize(source, scaleDown, Size(), scaleX, scaleY, INTER_LINEAR );
19 // Scaling up the image 1.8 times
20 cv::resize(source,scaleUp,Size(), scaleX*3, scaleY*3, INTER_LINEAR );
21
```

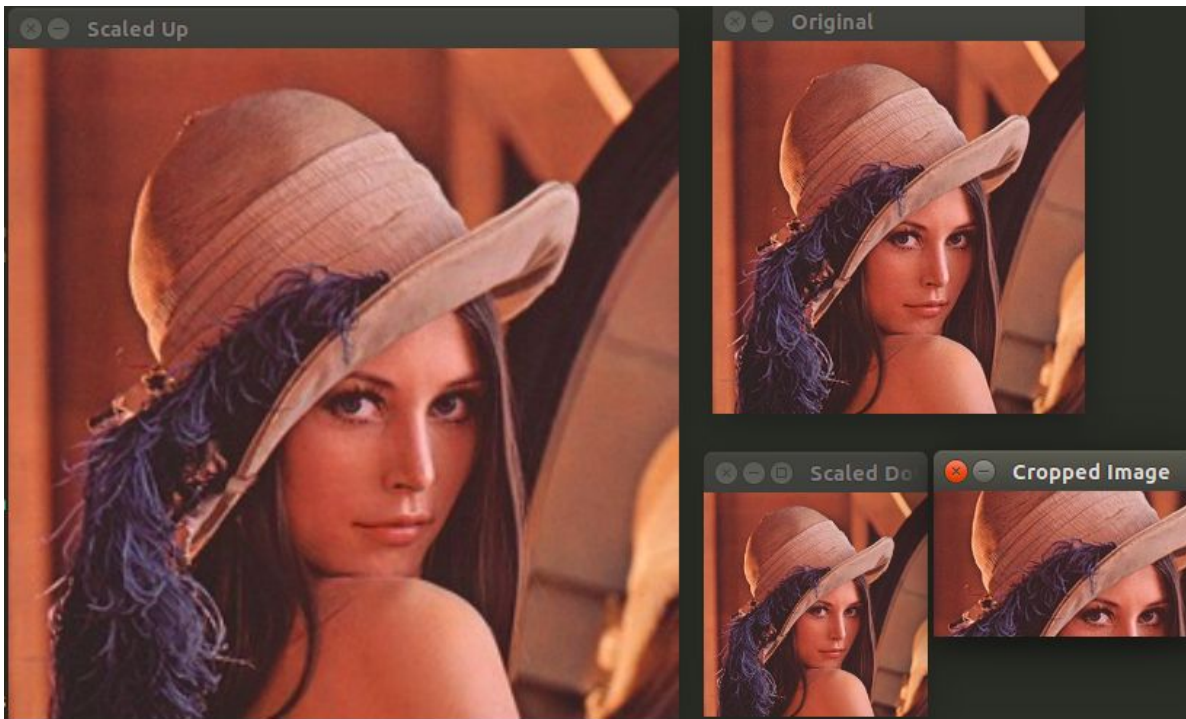
For cropping we use the Range function of OpenCV and then first specify the rows and then the columns of the cropping region.

```
22 //Cropped image
23 Mat crop = source(cv::Range(50,150), cv::Range(20, 200));
24
25 // Create Display windows for all three images
26 namedWindow("Original", WINDOW_AUTOSIZE);
```

```

27 namedWindow("Scaled Down", WINDOW_AUTOSIZE);
28 namedWindow("Scaled Up", WINDOW_AUTOSIZE);
29 namedWindow("Cropped Image", WINDOW_AUTOSIZE);
30
31 // Show images
32 imshow("Original", source);
33 imshow("Scaled Down", scaleDown);
34 imshow("Scaled Up", scaleUp);
35 imshow("Cropped Image", crop);
36
37 waitKey(0);
38
39 }
40

```



Sample Output of the Program

Python [Crop and Resize] [cropAndResize.py]

#This program demonstrates how we can use the resize function of OpenCV to shrink and enlarge images. It then uses the range function to crop an image and extract a portion of the image.

```

1 import cv2
2
3 # Reading the image
4 source = cv2.imread("../data/images/lena.jpg",1)
5
6 scaleX = 0.6

```

```
7 scaleY = 0.6
8
```

The resize has the following prototype:

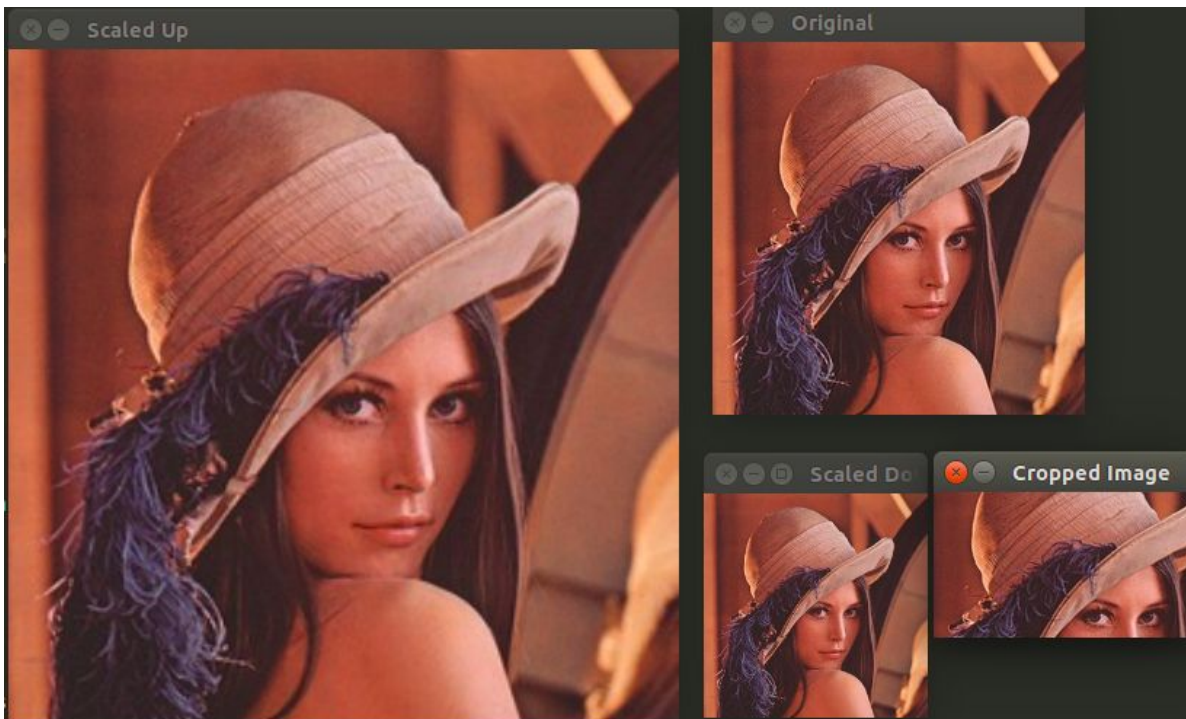
destinationImage = cv2.resize(sourceImage, (,), scale factor in x direction, scale factor in y direction, interpolation method)


We can either specify the size of the output to determine the height and width of the output or we can add the scaling factors and the Size will be calculated automatically.

```
9 # Scaling Down the image 0.6 times
10 scaleDown = cv2.resize(source, None, fx= scaleX,
11     fy= scaleY, interpolation= cv2.INTER_LINEAR)
12
13 # Scaling up the image 1.8 times
14 scaleUp = cv2.resize(source, None, fx= scaleX*3,
15     fy= scaleY*3, interpolation= cv2.INTER_LINEAR)
16
```

For cropping in python we can directly use the slicing facility of numpy arrays and slice it by first specifying the rows and then the columns of the cropping region.

```
17 #Cropped Image
18 crop = source[50:150,20:200]
19
20 # Displaying all the images
21 cv2.imshow("Original", source)
22 cv2.imshow("Scaled Down", scaleDown)
23 cv2.imshow("Scaled Up", scaleUp)
24 cv2.imshow("Cropped Image",crop)
25
26 cv2.waitKey(0)
```





Sample Output of the Program

Here we have defined the **scaling factors** for both x axis and y axis in lines **6** and **7** (python code) but this is not necessary. If we want we can also specify the exact size required in the second parameter of the resize function. Instead of None we can write **(640,480)** in case we want the size to be **640x480**. The x axis and y axis scaling factor will be automatically calculated. In case of C++ we can enter the size required in the Size function used, e.g. **Size(640,480)**. The **interpolation** method used here is **linear** which is the default and the most commonly used method.

Rotation

In OpenCV, we can rotate images about any point (let's call it center) and by an angle (let's simply call this angle).

In OpenCV, a rotation is represented as a 2x3 Matrix because rotations belong to the special class of transforms called the Affine Transform which is covered in the next section.

The `getRotationMatrix2D` function is used to get the matrix which will define the rotation.

The parameters for the function is

```
rotMat = getRotationMatrix2D(center, angle, scale)
```

Math Box : What is inside the rotation matrix?

If you are curious, `getRotationMatrix2D` returns the following matrix

$$\begin{bmatrix} \alpha & \beta & (1 - \alpha) \cdot \text{center.x} - \beta \cdot \text{center.y} \\ -\beta & \alpha & \beta \cdot \text{center.x} + (1 - \alpha) \cdot \text{center.y} \end{bmatrix}$$

α is given by **scale x cos (angle)**

β is given as **scale x sin (angle)**

The matrix `rotMat` can be used with `warpAffine` to get the final rotated image. Remember rotation is a special case of the affine transform and therefore the function is called `warpAffine`. We will see the affine transform in the next section.

The following C++ and Python codes provide a simple example. The code below is written in tutorial form to walk you step by step on how to rotate an image.

C++ [Rotation] [rotate.cpp]

// This program reads an image and rotates it. The rotation matrix is obtained using the `getRotationMatrix2D` function of OpenCV

```
1 #include <opencv2/opencv.hpp>
2
3 using namespace std;
4 using namespace cv;
5
6 int main(void)
7 {
8     Mat source, M, result;
9 }
```

```

10 // Read image
11 source = cv::imread("../data/images/lena.jpg",1);
12
13 Point2f center(source.cols/2, source.rows/2);
14 double rotationAngle=180;
15 double scale=1;
16

```

The `getRotationMatrix2D` function takes the following parameters:
Center: point about which rotation will occur
rotationAngle: angle by which rotation is occurring
Scale : an optional scaling factor

```

17 // Getting the matrix which will define the rotation
18 M = cv::getRotationMatrix2D(center,
19     rotationAngle, scale);

```

```

[-1, 1.224646799147353e-16, 256;
-1.224646799147353e-16, -1, 256]

```

Rotation matrix obtained here

```

20
21 // Rotate the source and store in result
22 cv::warpAffine(source, result, M,
23     Size(source.cols, source.rows));
24
25 // Create windows for display
26 namedWindow("Original Image", WINDOW_AUTOSIZE);
27 namedWindow("Rotated Image", WINDOW_AUTOSIZE);
28
29 // Display images
30 imshow("Original Image", source);
31 imshow("Rotated Image", result);
32
33 waitKey(0);
34
35 }

```

Python [Rotation] [rotate.py]

This program reads an image and rotates it. The rotation matrix is obtained using the `getRotationMatrix2D` function of OpenCV

```

1 import cv2
2
3 source = cv2.imread("../data/images/lena.jpg",1)
4
5 # Getting the dimensions of the image
6 dim = source.shape
7
8 rotationAngle = 180
9 scaleFactor = 1
10
11 # Rotating the image by 90 degrees about the center

```

```
12 # dim[0] stores the no of rows and dim[1] no of columns
```

The `getRotationMatrix2D` function takes the following parameters:
Center: point about which rotation will occur
rotationAngle: angle by which rotation is occurring
Scale : an optional scaling factor

```
13 rotationMatrix = cv2.getRotationMatrix2D((dim[1]/2,dim[0]/2),  
14                                         rotationAngle, scaleFactor)
```

```
[[ -1.00000000e+00  1.22464680e-16  2.56000000e+02]  
 [ -1.22464680e-16 -1.00000000e+00  2.56000000e+02]]
```

Rotation matrix obtained here

```
15  
16 result = cv2.warpAffine(source, rotationMatrix, (dim[1],dim[0]))  
17  
18 cv2.imshow("Original", source)  
19 cv2.imshow("Rotated Image", result)  
20  
21 cv2.waitKey(0)
```

The function `getRotationMatrix2D` does not modify the center so keep in mind whether that is the desired result or not. By scale, you can shrink or enlarge the image.

Here are the two examples of an image rotated by 30 and -30 degrees.



Original Image



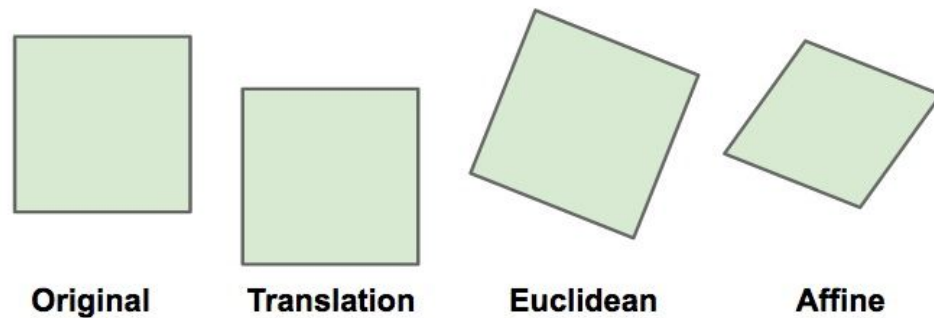
Image rotated by 30 degrees



Image rotated by -30 degrees

Affine Transform

An image can be warped in many different ways. The image below shows a few different kinds of image transformations.



A **translation** simply means moving the entire image in the x and y directions.

A **Euclidean Transform** is just a fancy word for a transform that is a combination of **translation** and **rotation**. Notice that a Euclidean transform preserves **orthogonality**; perpendicular lines in the original image are also perpendicular in the transformed image.

An **Affine Transform** is a combination of **translation**, **rotation**, **scaling** in x and y directions, and a parameter called **shear** because of which orthogonality is not preserved in the affine transform. Parallel lines in the original image remain parallel but lines that were perpendicular to each other are no longer at 90 degrees to each other in the transformed image.

Note: All the transforms listed above are linear transforms. Straight lines in original image remain straight in the transformed image. Later in the course, we will learn about non-linear transforms as well.

Affine Transform in OpenCV

In OpenCV, an Affine transform is stored in a 2 x 3 sized matrix. The first two columns of this matrix encode rotation, scale and shear, and the last column encodes translation (i.e. shift).

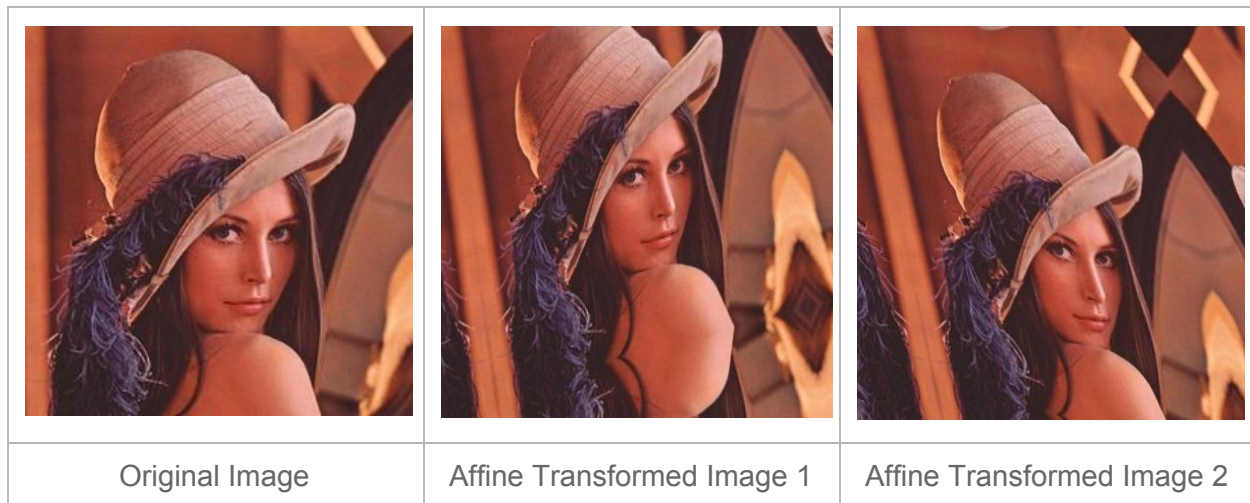
$$S = \begin{bmatrix} a & b & t_x \\ c & d & t_y \end{bmatrix}$$

Given a point (x, y) in the original image, the above affine transform, moves it to point (x_t, y_t) using the equation given below

$$\begin{bmatrix} x_t \\ y_t \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Translation, Rotation and Euclidean transforms are special cases of the Affine transform. In Translation, the rotation, scale and shear parameters are zero, while in a Euclidean transform the scale and shear parameters are zero.

In OpenCV, if you want to apply an affine transform to the entire image you can use the function **warpAffine**.



C++ [warpAffine example] [warpAffine.cpp]

```
1 #include <opencv2/core/core.hpp>
2 #include <opencv2/highgui/highgui.hpp>
3 #include <opencv2/imgproc/imgproc.hpp>
4 #include <iostream>
5 #include <cmath>
6 #include <vector>
7
8 using namespace cv;
9 using namespace std;
10
```

In this program, we will use OpenCV warpAffine function to transform a triangle from source image to another triangle as shown in the above figure.

```
11 int main(void)
12 {
13     // Read image
14     Mat source = imread("../data/images/lena.jpg",1);
15
16     // Create 2 warp matrices for different transformations
17     Mat warpMat = (Mat_<double>(2,3) << 1.2, 0.2, 2, -0.3, 1.3, 1 );
18     Mat warpMat2 = (Mat_<double>(2,3) << 1.2, 0.3, 2, 0.2, 1.3, 1);
19     Mat result,result2;
20
21     // Use warp affine
22     cv::warpAffine(source, result, warpMat, Size(1.5*source.rows,1.4*source.cols), INTER_LINEAR,
```

```

BORDER_REFLECT_101);
22 cv::warpAffine(source, result2, warpMat2, Size(1.5*source.rows, 1.4*source.cols), INTER_LINEAR,
BORDER_REFLECT_101);

23 // Display images
24 imshow("Original",source);
25 imshow("Result", result);
26 imshow("Result2",result2);
27 waitKey(0);
28 }

```

Python [warpAffine example] [warpAffine.py]

In this program, we will use OpenCV warpAffine function to transform a triangle in an image to another triangle as shown in the above figure.

```

1 import cv2
2 import numpy as np
3 # Read image
4 source = cv2.imread("../data/images/lena.jpg",1)

5 # Createmask/ warp matrix
6 warpMat = np.float32([[1.2, 0.2, 2],[-0.3, 1.3, 1]])

7 # Another mask/warp matrix
8 warpMat2 = np.float32([[1.2, 0.3, 2],[0.2, 1.3, 1]])

9 # Use warp affine
10 result = cv2.warpAffine(source, warpMat, (int(1.5*source.shape[0]),int(1.4*source.shape[1])), None,
flags=cv2.INTER_LINEAR, borderMode=cv2.BORDER_REFLECT_101 )
11 result2 = cv2.warpAffine(source, warpMat2, (int(1.5*source.shape[0]), int(1.4*source.shape[1])),
None, flags=cv2.INTER_LINEAR, borderMode=cv2.BORDER_REFLECT_101)

12 # Display images
13 cv2.imshow("Original",source)
14 cv2.imshow("Result", result)
15 cv2.imshow("Result2", result2)
16 cv2.waitKey(0);

```

The images must have given you an idea as to how would the transformed images look like. The code to obtain these images too is very simple. Here the transformation matrix is self-created. In **line 16 of the C++ code and line 6 of the Python code**, we create the first transformation matrix. The matrix here is a Mat class matrix and is has data entries of **double** type. In **line 18 of C++ and line 8 of Python** we have created another transformation matrix.

Note that the rotation is by **default counter clockwise**. We then **warpAffine the matrix in line 21 of C++ code and line 10 of Python code**. The Size parameter specifies the size of the output image. Note that the **size** has been kept **larger** than the **scaling factors** in order to view the entire image. The border mode : **BORDER_REFLECT_101** makes the border reflect the part of the image adjacent to it and hence you get a slightly modified image.

Sometimes we need to solve the inverse problem. Given the original image and the transformed image, we need to find the affine transform that relates the two images. In such cases, we need to find at least 3 point correspondences between the two images and use them to estimate the affine transform using `getAffineTransform`.

		
Three points on original image	Three points in transformed image 1	Three points in transformed image 2

C++ [`getAffineTransform` example] [`getAffine.cpp`]

```

1 #include <opencv2/core/core.hpp>
2 #include <opencv2/highgui/highgui.hpp>
3 #include <opencv2/imgproc/imgproc.hpp>
4 #include <iostream>
5 #include <cmath>
6 #include <vector>
7
8 using namespace cv;
9 using namespace std;
10
11 int main(void)
12 {

```

In this program, we will use OpenCV `getAffineTransform` function to obtain the warping matrix using two triangles, one being the input triangle and the other being the output triangle

```

13 // Input triangle
14 vector<Point2f> tri1;
15 tri1.push_back(Point2f(50, 50));
16 tri1.push_back(Point2f(180, 140));
17 tri1.push_back(Point2f(150, 200));
18
19 // Output triangle
20 vector<Point2f> tri2;
21 tri2.push_back(Point2f(72, 51));
22 tri2.push_back(Point2f(246, 129));

```



```

23 tri2.push_back(Point2f(222, 216));
24
25 // Another output triangle
26 vector<Point2f> tri3;
27 tri3.push_back(Point2f(77, 76));
28 tri3.push_back(Point2f(260, 219));
29 tri3.push_back(Point2f(242, 291));
30
31 // Get the transformation matrices
32 Mat warp = cv::getAffineTransform(tri1,tri2);
33 Mat warp2 = cv::getAffineTransform(tri1,tri3);
34 // Display the matrices
35 cout << warp<< "\n\n"<< warp2<< endl;
36 }

```

Python [getAffineTransform Example] [getAffine.py]

```

1 import cv2
2 import numpy as np
3

```

In this program, we will use OpenCV getAffineTransform function to obtain the warping matrix using two triangles, one being the input triangle and the other being the output triangle

```


4 # Input triangle
5 inp = np.float32([[50, 50], [180, 140], [150, 200]])
6 # Output triangle
7 output = np.float32([[72, 51], [246, 129], [222, 216]])

8 # Another output triangle
9 output2 = np.float32([[77, 76], [260, 219], [242, 291]])
10
11 # Get the transformation matrices
12 warpMat = cv2.getAffineTransform(inp, output)
13 warpMat2 = cv2.getAffineTransform(inp, output2)
14
15 # Display the matrices
16 print warpMat
17 print "\n\n"
18 print warpMat2

```

Output

<pre> [1.19, 0.20, 2; -0.30, 1.3, 1] [1.2, 0.30, 2; 0.19, 1.3, 1] </pre>	<pre> [[1.2 0.2 2.] [-0.3 1.3 1.]] [[1.2 0.3 2.] [0.2 1.3 1.]] </pre>
C++ output	Python Output



The code first makes a vector(C++ and numpy array in Python) of 3 points. The corresponding points in the transformed matrix is calculated using the formula:

$$\begin{bmatrix} x_t \\ y_t \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

The **getAffineTransform** function requires the **two** sets of points, to calculate the **transformation matrix**. As clear from the **output** the matrix is **same as the one** which we had **created in the previous code**, C++ gives a slightly

Note: Do not choose points which are collinear, they should form a valid triangle.

Datatype conversion

When we declare a **Mat** object, if we do not specify the default Mat object has the datatype of **unsigned integer(8 bits)**. Many a time this is not enough and we need more information about the pixel. To do this we generally convert the image to floating point datatype with each pixel having **32 bits** of data. This helps **prevent overflow** while working with the images.

The following C++ and Python codes provide a simple example.

C++ [convertTo] [datatypeConversion.cpp]

```
1 #include <opencv2/opencv.hpp>
2
3 using namespace std;
4 using namespace cv;
```

In this program, we will use OpenCV convertTo function to convert the datatype of the matrix used to store the image from integer to float and vice versa as well as change the range of the values from 0-255 to 0-1 and vice versa. This program will not give any different output but is crucial for other applications where floating point numbers are required.

```
5
6 int main(void)
7 {
8     Mat source;
9
10    // Scale will convert pixel values
11    double scale=1/255.0;
12    double shift=0;
13
14    // Read image
15    source= imread("../data/images/lena.jpg",1);
16
17    //Converting from unsigned char to float(32bit)
18    source.convertTo(source, CV_32FC3, scale, shift);
19
20    //Converting from float to unsigned char
21    source.convertTo(source, CV_8UC3, 1.0/scale, shift);
22 }
```

Python [convertTo] [datatypeConversion.py]

```
1 import cv2
2 import numpy as np
3
```

In this program, we will use OpenCV convertTo function to convert the datatype of the matrix used to store the image from integer to float and vice versa as well as change the range of the values from 0-255 to 0-1 and vice versa. This program will

not give any different output but is crucial for other applications where floating point numbers are required.

```
4 # Read image
5 source = cv2.imread("../data/images/lena.jpg",1)
6
7 scalingFactor = 1/255.0
8 # Convert unsigned int to float
9 source = np.float32(source)
10 source = source * scalingFactor
11
12 #Convert back to unsigned int
13 source = source * (1.0/scalingFactor)
14 source = np.uint8(source)
```

The scale or the scaling factor reduces the range of **0-255 to 0-1**, hence all the floating point numbers now range between 0-1. When we convert back to unsigned integer we use the **reciprocal** of the scaling factor and the range is converted from **0-1 to 0-255**.

Drawing over an image

OpenCV provides easy to use functions for drawing over an image. The following shapes and text can be drawn

1. Line
2. Circle
3. Ellipse
4. Rectangle
5. Polygon
6. Text

Below we have provided pseudo-code. We have also shared example code that draws different shapes on an image.

C++ [Usage]

cv::line (image, starting point , end point , color , line thickness, line type)

cv:: circle (image, center, radius, color of border, line thickness / fill type, line type)

cv:: ellipse (image, center, axes lengths, rotation degree of ellipse, starting angle , ending angle, color, line thickness / fill type, line type)

cv:: rectangle (image, upper left corner vertex, lower right corner vertex, line thickness / fill type, line type)

cv :: fillPoly (image, array of array of points, array containing the number of vertices for each polygon, number of contours, color, linetype)

cv :: putText (image, text, starting point of text, font type, font scale, color, linetype)

Python [Usage]

cv2.line (image, starting point , end point , color , line thickness, line type)

cv2.circle (image, center, radius, color of border, line thickness / fill type, line type)

cv2.ellipse (image, center, axes lengths, rotation degree of ellipse, starting angle , ending angle, color, line thickness / fill type, line type)

cv2.rectangle (image, upper left corner vertex, lower right corner vertex, line thickness / fill type, line type)

cv2.fillPoly (image, numpy array with points and number of vertices, number of contours, color, linetype)

cv2.putText (image, text, starting point of text, font type, font scale, color, linetype)

The **thickness** of the boundary is controlled by the thickness parameter. By default, only an outline of the shape is drawn. To draw a shape that is filled with the specified color, the thickness should be -1.

The lineType parameter controls the quality of rendering. When lineType is set to CV_AA, anti-aliased lines are drawn.

C++ code for drawing over an image

C++ [Draw over an Image] [imageDraw.cpp]

```
1 #include <opencv2/core.hpp>
2 #include <opencv2/imgproc.hpp>
3 #include <opencv2/highgui.hpp>
4
5 using namespace cv;
6
```

In this program, we will use OpenCV drawing functions to draw over images. We can draw lines, rectangles, circles, ellipses, polygons and write text over images too.

```
7 int main(void) {
8     // Create black empty images
9     Mat image = Mat::zeros(400, 400, CV_8UC3);
10
```

```
11 // Draw a line
12 // We are making a copy of image because we don't want to draw all the shapes
   on one image
```

```
13 Mat imageLine = image.clone();
14 Point start(15,20);
15 Point end(370,350);
16 int lineThickness=5;

17 // Scalar defines color of the line
18 // CV_AA stands for antialiased line which uses Gaussian filtering
```

```

19 line(imageLine, start, end, Scalar(110, 220, 0), lineThickness, CV_AA);
20 imshow("line", imageLine);
21
22 // Draw a circle
23 Mat imageCircle = image.clone();
24 Point center(200,200);
25 float radius=50.0;
26 int thickness=2;
27 int lineType=8;
28 // Scalar defines the color
29 circle(imageCircle, center, radius, Scalar(0, 0, 255), thickness, lineType);
30 imshow("circle", imageCircle);
31
32 // Draw an ellipse
33 Mat imageEllipse = image.clone();
34 /* Defining all parameters separately for the first ellipse, rest can be
followed */
35 // center of ellipse
36 Point centerEllipse(200,200);
37 // Half of the size of the ellipse main axes
38 Size axes(100.0,160.0);
39 // Angle by which ellipse will be rotated
40 int rotationAngle=45;
41 int startingAngle=0;
42 int endAngle=360;
43 // thickness of the border of the ellipse
44 int thicknessEllipse=1;
45 ellipse(imageEllipse, centerEllipse, axes, rotationAngle, startingAngle,
46         endAngle, Scalar(255, 0, 0), thicknessEllipse, lineType);
47 ellipse(imageEllipse, Point(200, 200), Size(100.0, 160.0), 135, 0, 360, Scalar(255, 0, 0), 10, 8);
48 ellipse(imageEllipse, Point(200, 200), Size(150.0, 50.0), 135, 0, 360, Scalar(0, 255, 0), 1, 8);
49 imshow("ellipse", imageEllipse);
50
51 // Draw a rectangle (5th argument is not -ve)
52 Mat imageRectangle = image.clone();
53 Point vertex(15,20);
54 // Diagonally opposite vertex
55 Point oppVertex(115,120);
56 int thicknessRectangle=1;
57 lineType=4;
58 // Scalar again defines the color of the border
59 rectangle(imageRectangle, vertex, oppVertex, Scalar(0, 55, 255), thicknessRectangle, lineType );
60 imshow("rectangle1", imageRectangle);
61 // Draw a filled rectangle (5th argument is -ve)
62 // Scalar defines the color with which the rectangle will be filled
63 rectangle(imageRectangle, Point(125, 135), Point(225, 235), Scalar(100, 155, 25), -1, 8);
64 imshow("rectangle2", imageRectangle);

```

```

65

66 // Draw a filled Polygon

67 Mat imagePolygon = image.clone();
68 int w = 400;

69 /* Create some points */
70 // These points will form the vertices of the rook

71 Point rookPoints[1][20];
72 rookPoints[0][0] = Point(w/4.0, 7*w/8.0);
73 rookPoints[0][1] = Point(3*w/4.0, 7*w/8.0);
74 rookPoints[0][2] = Point(3*w/4.0, 13*w/16.0);
75 rookPoints[0][3] = Point(11*w/16.0, 13*w/16.0);
76 rookPoints[0][4] = Point(19*w/32.0, 3*w/8.0);
77 rookPoints[0][5] = Point(3*w/4.0, 3*w/8.0);
78 rookPoints[0][6] = Point(3*w/4.0, w/8.0);
79 rookPoints[0][7] = Point(26*w/40.0, w/8.0);
80 rookPoints[0][8] = Point(26*w/40.0, w/4.0);
81 rookPoints[0][9] = Point(22*w/40.0, w/4.0);
82 rookPoints[0][10] = Point(22*w/40.0, w/8.0);
83 rookPoints[0][11] = Point(18*w/40.0, w/8.0);
84 rookPoints[0][12] = Point(18*w/40.0, w/4.0);
85 rookPoints[0][13] = Point(14*w/40.0, w/4.0);
86 rookPoints[0][14] = Point(14*w/40.0, w/8.0);
87 rookPoints[0][15] = Point(w/4.0, w/8.0);
88 rookPoints[0][16] = Point(w/4.0, 3*w/8.0);
89 rookPoints[0][17] = Point(13*w/32.0, 3*w/8.0);
90 rookPoints[0][18] = Point(5*w/16.0, 13*w/16.0);
91 rookPoints[0][19] = Point(w/4.0, 13*w/16.0);
92
93 // A pointer of type Point defined for the first row of rookPoints
94 const Point* ppt[1] = {rookPoints[0]};
95 int npt[] = {20};
96 // Fill the polygon formed by the points
97 lineType=8;
98 // No of countours in the region
99 int contours=1;

100 /* the second argument of fillPoly requires an array of polygons with each
101 polygon represented by an array of Points hence we create ppt which is
102 of type Point* */
103 /* npt is an array with each element corresponding to the number of vertex
104 of the corresponding polygon in ppt */

105 fillPoly(imagePolygon, ppt, npt, contours, Scalar(255, 255, 255), lineType);
106 imshow("Image", imagePolygon);
107

108 // Put text into image

109 Mat imageText = image.clone();
110 // the starting point of the text
111 Point bottomLeftCorner(100,180);
112 // to scale up or down the default size of the font
113 double fontScale=1;
114 lineType=4;
115 putText(imageText, "OpenCV Basics", bottomLeftCorner, FONT_HERSHEY_SIMPLEX,
116         fontScale, Scalar(0, 200, 200), lineType);
117 imshow("text", imageText);

```



```
118 waitKey(0);
119 return 0;
120 }
```

Python code for drawing over an image

Python [Draw over Image] [imageDraw.py]

```
1 import cv2
2 import numpy as np
3
4 image = np.zeros((400, 400, 3), dtype=np.uint8)
5
```

In this program, we will use OpenCV drawing functions to draw over images. We can draw lines, rectangles, circles, ellipses, polygons and write text over images too.

```
6 # Draw a line
7 # We are making a copy of image because we don't want to draw all the shapes on one image
```

```
8 imageLine = image.copy()
9 #First Parameter is the image on which the line will be drawn
10 #Second parameter is the starting point of the line
11 #Third Parameter is the end point of the line
12 #Fourth parameter defines the color of the line
13 cv2.line(imageLine, (15, 20), (370, 350), (110, 220, 0), thickness=2, lineType=cv2.LINE_AA)
14 cv2.imshow("line", imageLine)
15
```

16 # Draw a circle

```
17 imageCircle = image.copy()
18 #Second parameter is the center of the circle
19 #Third paramter is the radius of the circle
20 #Fourth parameter is the color of the border or if thickness is negative the color with which the circle will be filled
21 cv2.circle(imageCircle, (200, 200), 50, (0, 0, 255), thickness=2, lineType=cv2.LINE_AA)
22 cv2.imshow("image", image)
23 cv2.imshow("circle", imageCircle)
24
```

25 # Draw an ellipse

26 # IMP Note: Ellipse Centers and Axis lengths must be integers

```
27 imageEllipse = image.copy()
28 #Second parameter is the center
29 #Third parameter is the semi axes lengths of the ellipse
30 #seventh parameter is the color of the border
31 rotationAngle=45
32 startingAngle=0
33 endAngle=360
34 cv2.ellipse(imageEllipse, (200, 200), (100, 160), rotationAngle, startingAngle, endAngle, (255, 0, 0), thickness=2, lineType=cv2.LINE_AA)
35 cv2.ellipse(imageEllipse, (200, 200), (100, 160), 135, 0, 360, (0, 0, 255), thickness=5, lineType=cv2.LINE_AA)
36 cv2.ellipse(imageEllipse, (200, 200), (150, 50), 135, 0, 360, (0, 255, 0), thickness=2, lineType=cv2.LINE_AA)
```

```

37 cv2.imshow("ellipse", imageEllipse)
38

39 # Draw a rectangle (thickness is a positive integer)

40 imageRectangle = image.copy()
41 cv2.rectangle(imageRectangle, (105, 105), (205, 205), (0, 55, 255), thickness=+2,
lineType=cv2.LINE_8)
42 cv2.imshow("rectangle1", imageRectangle)

43 # Draw a filled rectangle (thickness is a negative integer)

44 cv2.rectangle(imageRectangle, (210, 210), (310, 310), (100, 155, 25), thickness=-1,
lineType=cv2.LINE_8)
45 cv2.imshow("rectangle2", imageRectangle)
46

47 # Draw a filled Polygon. We create some points and add them to an array and
create a polygon out of it.

48 imagePolygon = image.copy()
49 w = 400
50 # Create some points
51 rookPoints = np.zeros((1, 20, 2)) #Initializes a 3 dimensional numpy array with zeros
52 #Set points in the numpy array
53 rookPoints[0, 0, :] = (w/4.0, 7*w/8.0)
54 rookPoints[0, 1, :] = (3*w/4.0, 7*w/8.0)
55 rookPoints[0, 2, :] = (3*w/4.0, 13*w/16.0)
56 rookPoints[0, 3, :] = (11*w/16.0, 13*w/16.0)
57 rookPoints[0, 4, :] = (19*w/32.0, 3*w/8.0)
58 rookPoints[0, 5, :] = (3*w/4.0, 3*w/8.0)
59 rookPoints[0, 6, :] = (3*w/4.0, w/8.0)
60 rookPoints[0, 7, :] = (26*w/40.0, w/8.0)
61 rookPoints[0, 8, :] = (26*w/40.0, w/4.0)
62 rookPoints[0, 9, :] = (22*w/40.0, w/4.0)
63 rookPoints[0, 10, :] = (22*w/40.0, w/8.0)
64 rookPoints[0, 11, :] = (18*w/40.0, w/8.0)
65 rookPoints[0, 12, :] = (18*w/40.0, w/4.0)
66 rookPoints[0, 13, :] = (14*w/40.0, w/4.0)
67 rookPoints[0, 14, :] = (14*w/40.0, w/8.0)
68 rookPoints[0, 15, :] = (w/4.0, w/8.0)
69 rookPoints[0, 16, :] = (w/4.0, 3*w/8.0)
70 rookPoints[0, 17, :] = (13*w/32.0, 3*w/8.0)
71 rookPoints[0, 18, :] = (5*w/16.0, 13*w/16.0)
72 rookPoints[0, 19, :] = (w/4.0, 13*w/16.0)
73

74 # fillPoly expects data points to be in int32 format

75 ppt = np.int32(np.around(rookPoints))
76
77 #Fill polygon formed by the points
78 #third parameter is the color of the filled region
79 cv2.fillPoly(imagePolygon, ppt, (255, 255, 255), lineType=cv2.LINE_8)
80 cv2.imshow("Image", imagePolygon)
81

82 # Put text into image

83 imageText = image.copy()
84 #third parameter is the origin of text
85 #sixth parameter is the color of the text

```

```
86 fontScale=1
87 lineType=4
88 cv2.putText(imageText, "OpenCV Basics", (100, 180), cv2.FONT_HERSHEY_SIMPLEX, fontScale, (0, 200,
100), lineType)
89 cv2.imshow("text", imageText)
90
91 cv2.waitKey(0)
```

References and Further reading

http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_gui/py_image_display/py_image_display.html

http://docs.opencv.org/3.0-beta/doc/tutorials/introduction/display_image/display_image.html

http://docs.opencv.org/3.0-beta/modules/imgcodecs/doc/reading_and_writing_images.html

http://docs.opencv.org/3.0-beta/doc/tutorials/introduction/load_save_image/load_save_image.html

http://docs.opencv.org/3.2.0/da/d6e/tutorial_py_geometric_transformations.html

http://docs.opencv.org/3.1.0/d4/d61/tutorial_warp_affine.html

<https://www.learnopencv.com/image-alignment-ecc-in-opencv-c-python/>

<https://www.learnopencv.com/warp-one-triangle-to-another-using-opencv-c-python/>

http://docs.opencv.org/3.1.0/dc/da5/tutorial_py_drawing_functions.html

<http://opencvexamples.blogspot.com/2013/10/basic-drawing-examples.html>

■ ■ ■