OpenCV Guide

Meher Krishna Patel

Created on : July, 2019

Last updated : May, 2021

Table of contents

List of figures

Chapter 1

OpenCV with Python

1.1 OpenCV Basics

1.1.1 Introduction

OpenCV-3 is used in this tutorial which can be installed using below command,

```
pip install opencv-python==3.4.5.20
```

1.1.2 Load image

- See comments for details,

```
# load_save_image.py

import cv2

# read image
img = cv2.imread("images/shapes.jpg")

# show image
cv2.imshow("Shapes", img) # Window name -> Shapes

# wait for key before closing the window
cv2.waitKey(0)

# save image
cv2.imwrite("images/saved_by_opencv.jpg", img)
```

- Run the code

```
$ python load_save_image.py
```

- Output is shown Fig. 1.1,

1.1.3 Load Video

- cv2.VideoCapture(0) is use to show the video which is captured by webcam,

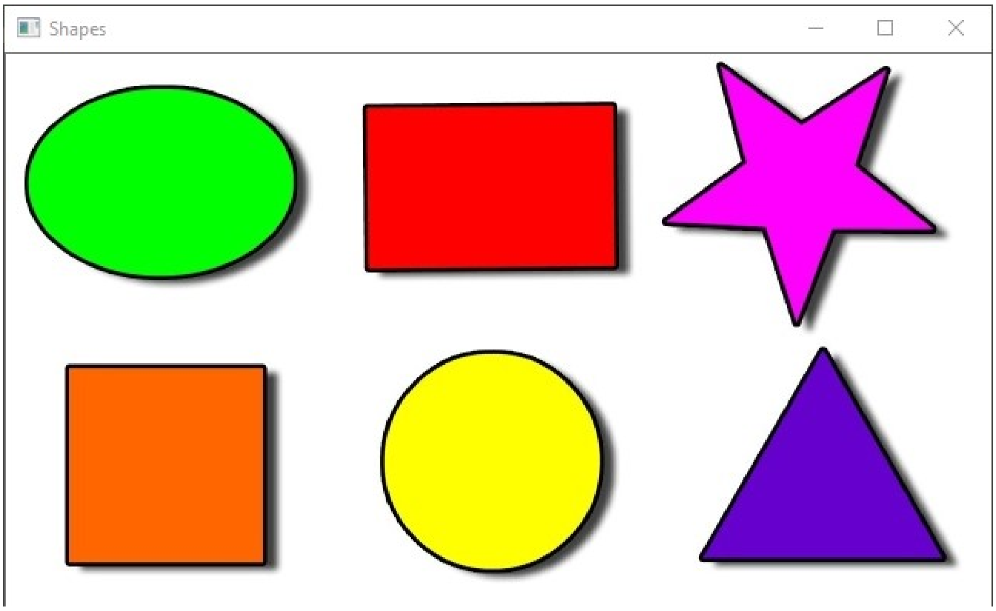

Fig. 1.1: Shapes

```python
# load_video.py

import numpy as np
import cv2

# load video
cap = cv2.VideoCapture("images/timer.mp4")

while(True):
    # Capture frame-by-frame
    ret, frame = cap.read()

    # Display the resulting frame
    cv2.imshow('frame',frame)
    if cv2.waitKey(30) & 0xFF == ord('q'): # press q to exit
        break

# When everything done, release the capture
cap.release()
cv2.destroyAllWindows()
```

```
$ python load_video.py
```

1.1.4 Basic operations on images

1.1.4.1 Accessing and modifying pixel

- In images, the pixel coordinates starts from (0, 0).
- [B, G, R] format is used in OpenCV.

```python
# access_modify_pixel.py


import cv2
import numpy as np
```

(continues on next page)

(continued from previous page)

```


# read image
img = cv2.imread("images/shapes.jpg")


# pixel at point [10, 10] = white i.e. 255, 255, 255
px = img[10, 10]
print("original pixel: ", px) # [255 255 255]
cv2.imshow("Shapes", img)


# modify pixel to red : a dot can be seen in the image
img[10, 10] = (0, 0, 255)
px = img[10, 10]
print("Modified pixel: ", px) # [255 0 0]
cv2.imshow("Red dot at (10 10)", img)


# access the shape of the image
(h, w) = img.shape[:2] # height and width of image
print("height={}, width={}".format(h,w)) # height=360, width=640

print("Image size = ", img.size) # size of image = h*w*3 = 691200


cv2.waitKey(0)
```

- Output is shown Fig. 1.2,

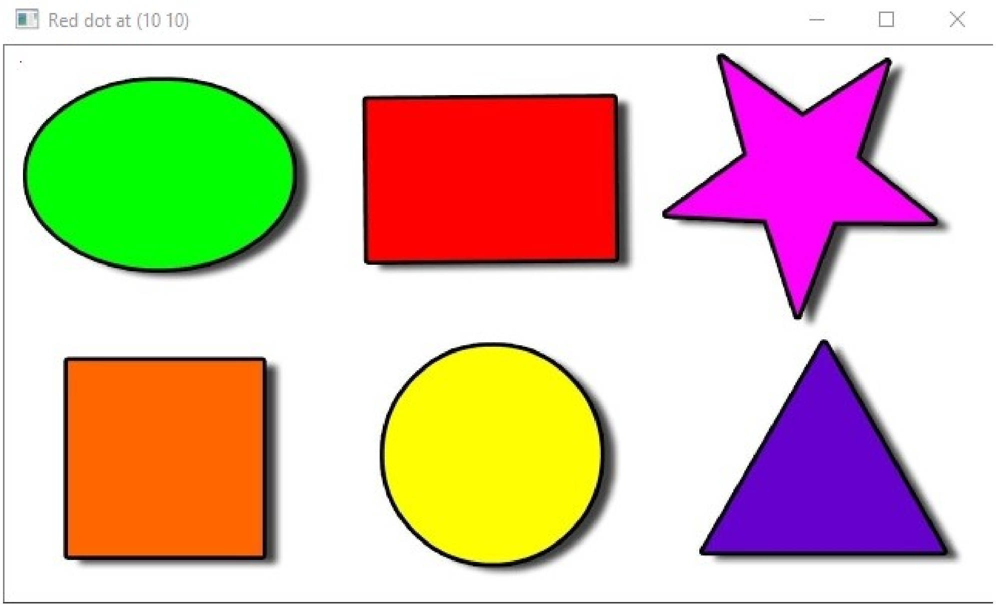

Fig. 1.2: Red dot at (10 10)

1.1.4.2 Split and Merge

- In this section, the color image is split and plotted into R, G and B color. Also, these R, G and B are merged together to get the original image.

```python
# split_merge.py

# square is of red color: R = 255 (i.e. white), B & G = 0 (i.e. black)

# circle is of yellow color: R & G = 255 (i.e. white), B = 0 (i.e. black)

# triangle is purple: a mix of R & B with different ratio; therefore a different
# gray-shades for R and B (more of blue therefore lighter-gray shade) will be shown;
# whereas G = 0 (i.e. black)

import cv2
import numpy as np


# read image
img = cv2.imread("images/shapes.jpg")
(h, w) = img.shape[:2] # height and width of image

# split image into BGR
(B, G, R) = cv2.split(img)

# show B, G, R channels
cv2.imshow("Shapes", img)
cv2.imshow("Blue", B)
cv2.imshow("Green", G)
cv2.imshow("Red", R)

merge_img = cv2.merge([B, G, R])
cv2.imshow("Merged BGR", merge_img)


cv2.waitKey(0)
```

- Output is shown Fig. 1.3,

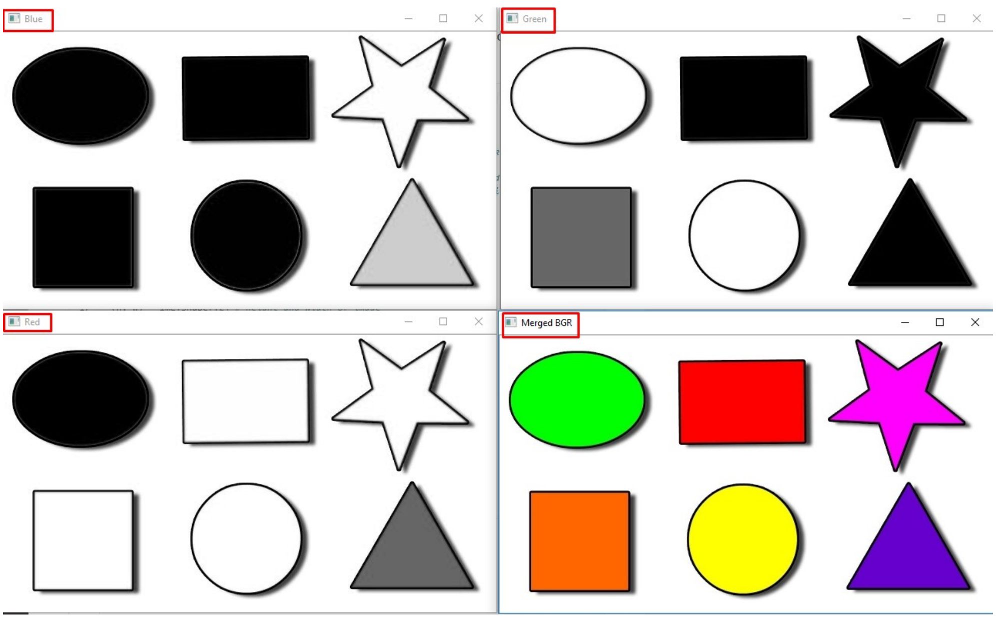

Fig. 1.3: Split and merge

1.1.4.3 Crop image

In this section, we will crop the image in 4 equal part and change the color of 2 parts.

```
# crop_img.py

import cv2

# read image
img = cv2.imread("images/shapes.jpg")
cv2.imshow("Shapes", img) # display image

# Shape = (width, height, channel);  channel = 3 i.e. B, G, R
print("Image shape: ", img.shape) # Image shape:  (360, 640, 3)


# extract height and width i.e. first two values (360, 640)
(h, w) = img.shape[:2]
print("Heigth = {}, Width = {}".format(h, w)) # Heigth = 360, Width = 640

#### Pixel values
# print pixel value (B, G, R) at [0, 0]
print("(B G R) = ", img[0, 0]) # (B G R) =  [255 255 255] i.e. white
# print pixel value (B, G, R) at [40, 310]
print("(B G R) = ", img[40, 310]) # (B G R) =  [  0    0 254] i.e. red


### Crop image

# center point of image
# note that we will use the cX and cY as pixel location
# therefore these need to be an integer value, hence // is used
(cX, cY) = (w//2, h//2)

# top left i.e. 0-to-cY and 0-to-cX
top_left = img[0:cY, 0:cX]
cv2.imshow("Top Left", top_left) # display image

top_right = img[0:cY, cX:w]
cv2.imshow("Top Right", top_right) # display image

bottom_left = img[cY:h, 0:cX]
cv2.imshow("Bottom Left", bottom_left) # display image

bottom_right = img[cY:h, cX:w]
cv2.imshow("Bottom Right", bottom_right) # display image

cv2.waitKey(0)


### change color for cropped sections
img[cY:h, cX:w] = [255, 0, 0] # bottom right to Blue color
cv2.imshow("Bottom Right", bottom_right) # display image

# Green + Red = Yellow
img[0:cY, 0:cX] = [0, 255, 255] # Yellow color for top-left
cv2.imshow("Top Left", top_left) # display image


cv2.waitKey(0)
```

- Output is shown Fig. 1.4,

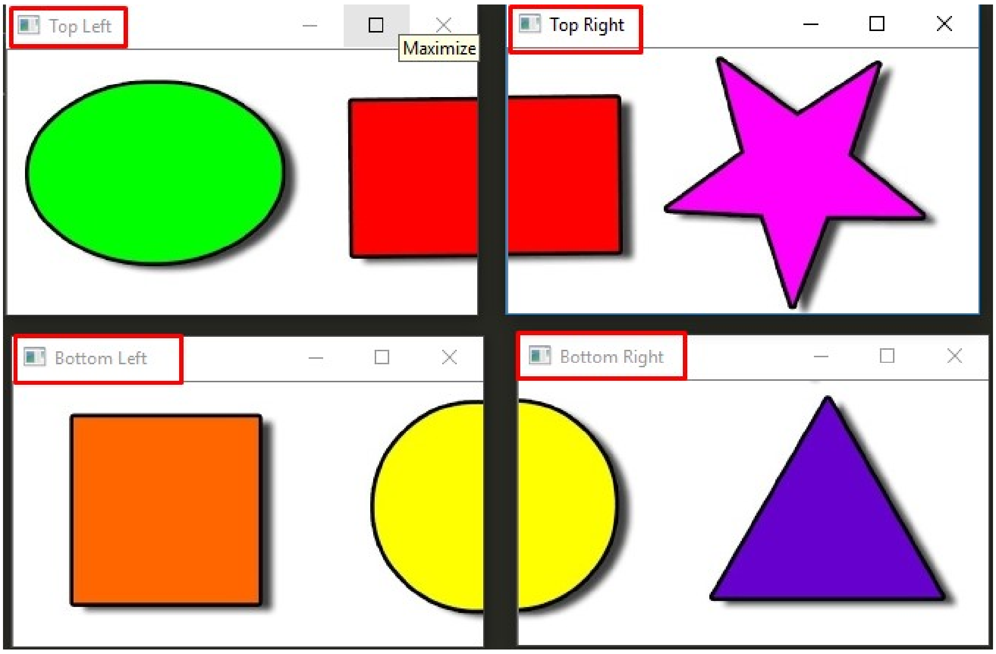

Fig. 1.4: Crop image

1.1.4.4 Image arithmetic

- OpenCV sets the maximum and minimum as 255 and 0 respectively.
- Numpy does the modulo addition.

```python
# image_arith.py

import numpy as np
import cv2

x = np.uint8([250])
y = np.uint8([10])

print("OpenCV 250 + 10: ", cv2.add(x,y)) # 250+10 = 260 => 255
print("Numpy 250 + 10: ", x+y)          # 250+10 = 260 % 256 = 4


img = cv2.imread("images/shapes.jpg")
cv2.imshow("Shapes", img)


print("Initial pixel at [50, 50]\t: ", img[50, 50])
new_pixel =  90 * np.ones(img.shape, dtype = "uint8")

print("Add/subtract 90")

opencv_img = cv2.add(img, new_pixel)
print("OpenCV addition pixel at [50, 50]\t: ", opencv_img[50, 50])
cv2.imshow("OpenCV add", opencv_img)

opencv_img = cv2.subtract(img, new_pixel)
print("OpenCV subtract pixel at [50, 50]\t: ", opencv_img[50, 50])
cv2.imshow("OpenCV subtract", opencv_img)


numpy_img = img + new_pixel
```

(continues on next page)

(continued from previous page)

```
print("Numpy addition pixel at [50, 50]\t: ", opencv_img[50, 50])
cv2.imshow("Numpy add", numpy_img)

numpy_img = img - new_pixel
print("Numpy subtract pixel at [50, 50]\t: ", opencv_img[50, 50])
cv2.imshow("Numpy subtract", numpy_img)

cv2.waitKey(0)
```

- Output will be as below,

```
OpenCV 250 + 10:  [[255]]
Numpy 250 + 10:  [4]
Initial pixel at [50, 50]   :  [  1 255   0]
Add/subtract 90
OpenCV addition pixel at [50, 50]   :  [ 91 255  90]
OpenCV subtract pixel at [50, 50]   :  [  0 165   0]
Numpy addition pixel at [50, 50]    :  [  0 165   0]
Numpy subtract pixel at [50, 50]    :  [  0 165   0]
```

- Output figure is shown Fig. 1.5,

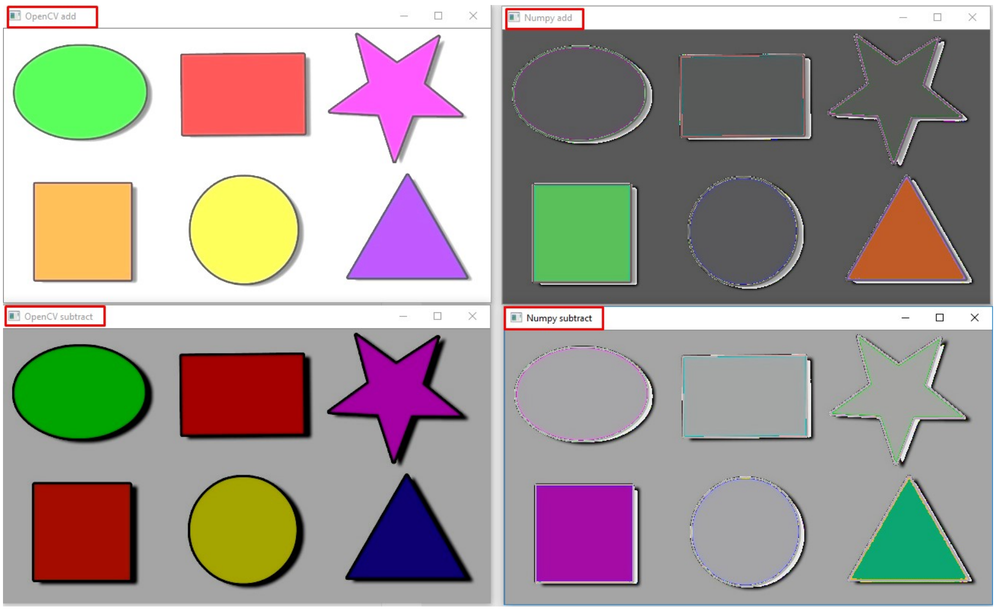

Fig. 1.5: Image arithmetic

1.1.4.5 Threshold

For every pixel, the same threshold value is applied. If the pixel value is smaller than the threshold, it is set to 0, otherwise it is set to a maximum value.

```
# threshold_img.py

import cv2
import numpy as np


# read image
img = cv2.imread("images/rose.jpg")
```

(continues on next page)

(continued from previous page)

```
cv2.imshow("Rose", img)

(T, thresh) = cv2.threshold(img, 100, 255, cv2.THRESH_BINARY)
cv2.imshow("Threshold Binary", thresh)


(T, thresh) = cv2.threshold(img, 100, 255, cv2.THRESH_BINARY_INV)
cv2.imshow("Threshold Binary Inverse", thresh)


cv2.waitKey(0)
```

- Output figure is shown Fig. 1.6,

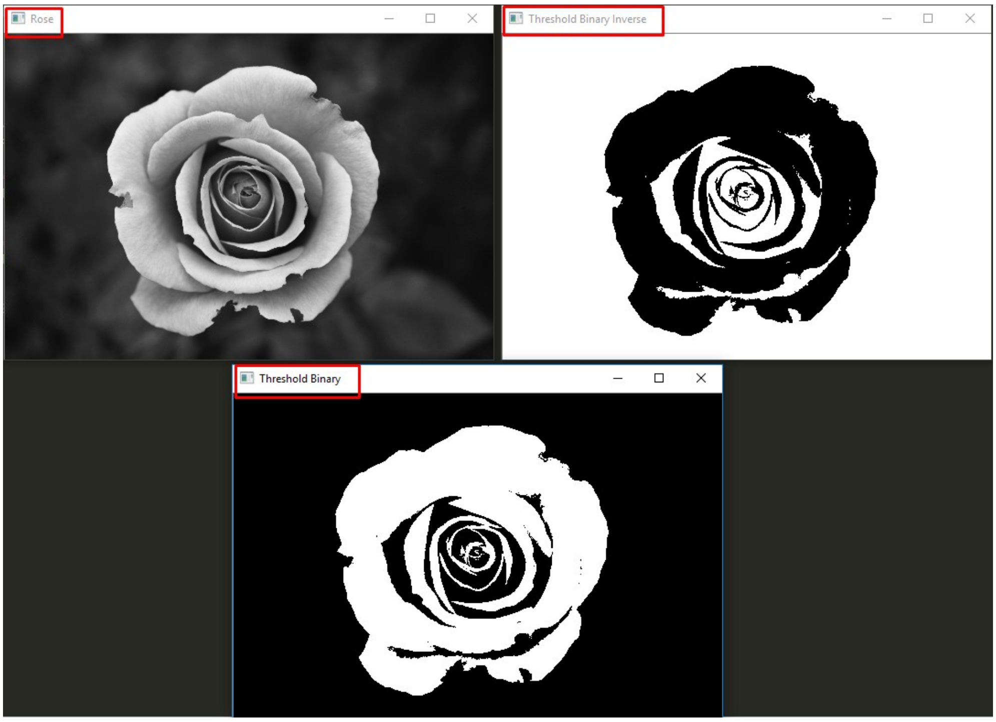

Fig. 1.6: Threshold

1.1.5 Geometric Transformations

1.1.5.1 Scaling

Scaling is just resizing of the image. OpenCV comes with a function cv2.resize() for this purpose.

```
# scale_img.py

import cv2
import numpy as np


# read image
img = cv2.imread("images/shapes.jpg")
(h, w) = img.shape[:2] # height and width of image
cv2.imshow("Shapes", img) # display image

```

(continues on next page)

(continued from previous page)

```
# scale by 0.5 in both x and y direction
scale_img = cv2.resize(img, (w//2, h//2), interpolation = cv2.INTER_CUBIC)
cv2.imshow("Resize Shapes", scale_img) # display image


cv2.waitKey(0)
```

- Output figure is shown Fig. 1.7,

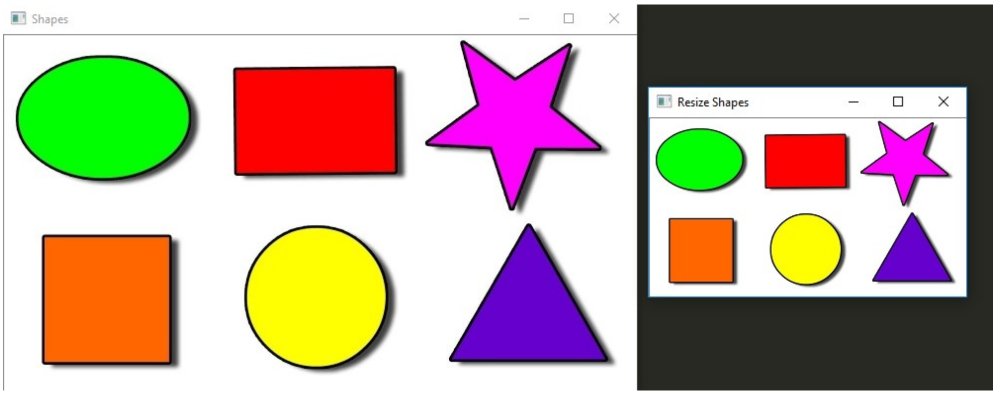

Fig. 1.7: Resize or scaling

1.1.5.2 Flip

Three types of flips are possible,

- 0 : Horizontal flip
- 1 : Vertical flip
- -1 : Both horizontal and vertical flip

```
# flip_img.py

import cv2
import numpy as np


# read image
img = cv2.imread("images/shapes.jpg")
(h, w) = img.shape[:2] # height and width of image
cv2.imshow("Shapes", img) # display image

# flip horizontal
flip_horizontal = cv2.flip(img, 0)
cv2.imshow("Horizontal Flip", flip_horizontal) # display image

# flip vertical
flip_vertical = cv2.flip(img, 1)
cv2.imshow("Vertical Flip", flip_vertical) # display image

# flip vertical and horizontal both
flip_both = cv2.flip(img, -1)
cv2.imshow("Horizontal and Vertical Flip", flip_both) # display image


cv2.waitKey(0)
```

- Output figure is shown Fig. 1.8,

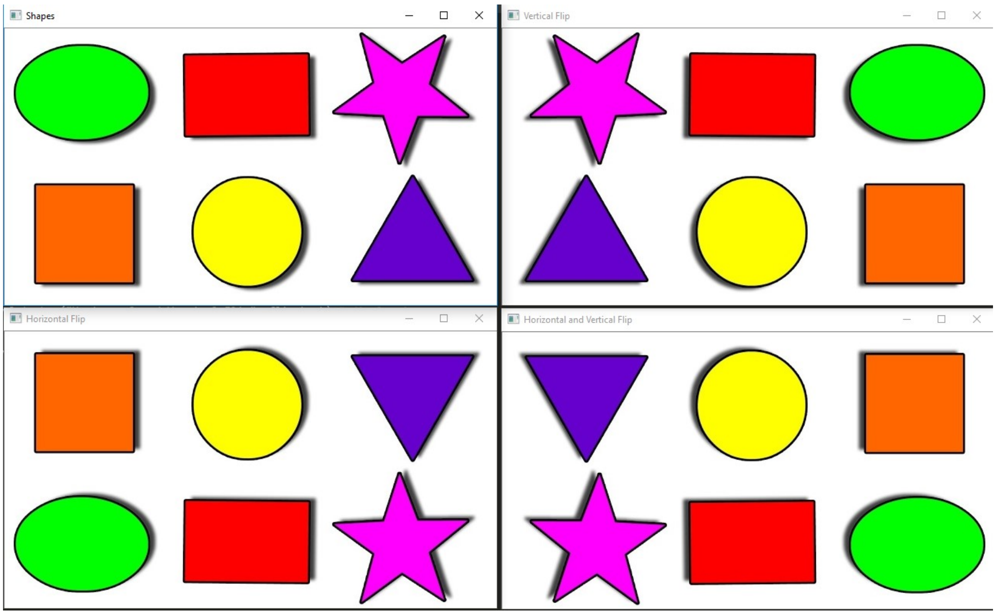

Fig. 1.8: Flip

1.1.5.3 Translation

Translation is the shifting of object's location.

```
# translate_img.py

import cv2
import numpy as np

# translation matrix is defined as [1 0 t_x; 0 1 t_y]
# traslate/shift by t_x and t_y respectively

# shift by 30 (right) and 50 (down) in x and y direction respectively
# similarly -30 for left and -50 for upward shift
shift_matrix = np.float32([[1, 0, 30], [0, 1, 50]])


# read image
img = cv2.imread("images/shapes.jpg")
(h, w) = img.shape[:2] # height and width of image
cv2.imshow("Shapes", img) # display image


####### Now perform shift and rotate operation
shift_img = cv2.warpAffine(img, shift_matrix, (w, h))
cv2.imshow("Shifted Down and Right", shift_img)


cv2.waitKey(0)
```

- Output figure is shown Fig. 1.9,

```
# shift by -30 and -50 in x and y direction respectively
shift_matrix = np.float32([[1, 0, 30], [0, 1, 50]])
```

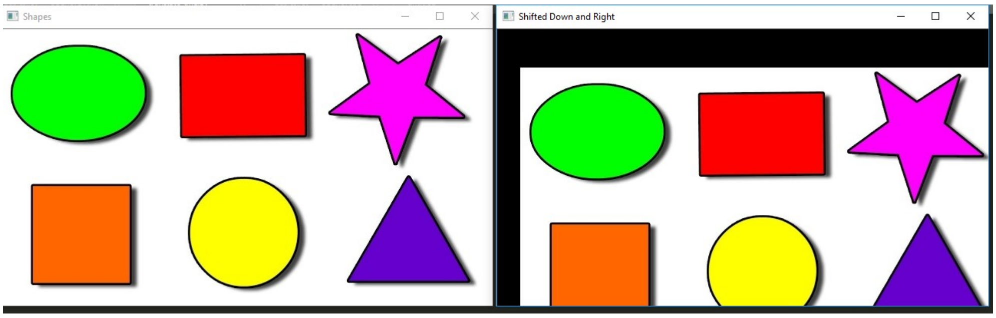

Fig. 1.9: Translation

1.1.5.4 Rotation

- We need to define the rotation angle along with a point for rotation.

```python
# rotate_img.py

import cv2
import numpy as np

# read image
img = cv2.imread("images/shapes.jpg")
(h, w) = img.shape[:2] # height and width of image
cv2.imshow("Shapes", img) # display image


# first define the point of rotation, e.g. (w/2, h/2) i.e. center of the image
(cX, cY) = (w/2, h/2)
# now define rotation matrix with 45 degree of rotation
rotation_matrix = cv2.getRotationMatrix2D((cX, cY), 45, 1.0)

# rotate and plot the image
rotated = cv2.warpAffine(img, rotation_matrix, (w, h))
cv2.imshow("Rotated by 45 Degrees", rotated)


cv2.waitKey(0)
```

- Output figure is shown Fig. 1.10,

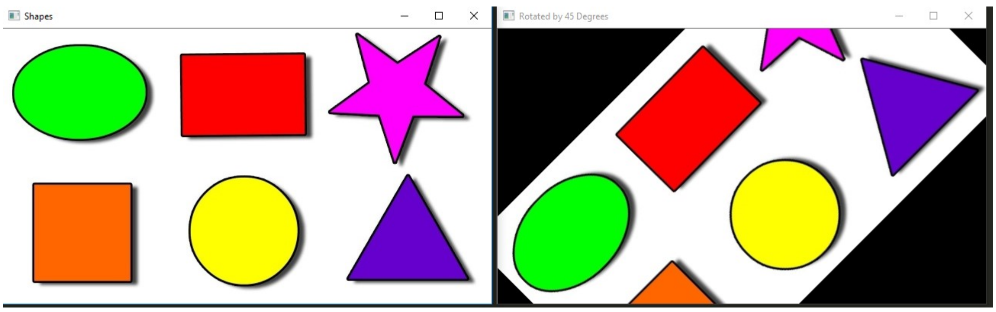

Fig. 1.10: Rotation

1.1.6 Drawing

- In this section, lines, rectangle, circle and ellipse are drawn using OpenCV.

```python
# drawing_img.py

import cv2
import numpy as np


# read image
img = cv2.imread("images/shapes.jpg")
(h, w) = img.shape[:2] # height and width of image

# draw blue horizontal and vertical lines at the center of figure
# initial and final point are required to draw line
cv2.line(img,(0, h//2), (w, h//2), (255,0,0), 3) # horizontal line
cv2.line(img,(w//2, 0), (w//2, h), (255,0,0), 3) # vertical line

# draw rectangle
# top-left corner (5, 10) and bottom-right corner (200, 170) of rectangle
# points are calculated manually
cv2.rectangle(img, (5, 10), (200, 170),(0,0,250),3)


# draw circle
# center coordinates (w//2, h//2) and radius (50) are
# required to to draw circle. 10 is the line width
cv2.circle(img, (w//2, h//2), 50 , (0,0,0), 10) # black
cv2.circle(img, (w//2, h//2), 30, (0,0,255), -1) # -1 : filled circle


# draw ellipse
# center: (w//2, h//2)
# (major axis, minor axis): (100,50)
# direction of rotation: 0;  where 0 : anticlockwise, 1: clockwise
# start angle and end angle: 0, 360
# color: (0, 255, 0)
# width: 5 (-1 for filled)
cv2.ellipse(img, (w//2, h//2), (100,50), 0, 0, 360, (0, 255, 0), 5)


cv2.imshow("Shapes", img) # display image
cv2.waitKey(0)
```

- Output figure is shown Fig. 1.11,

1.1.7 Bitwise operation

```python
# bitwise_img.py

import numpy as np
import cv2


# create and display frame of size 300
rectangle = np.zeros((300, 300), dtype = "uint8")
# display empty frame
cv2.imshow("Frame", rectangle)


# draw white rectangle
cv2.rectangle(rectangle, (20, 20), (280, 280), 255, -1)
cv2.imshow("Rectangle", rectangle)
```

(continues on next page)

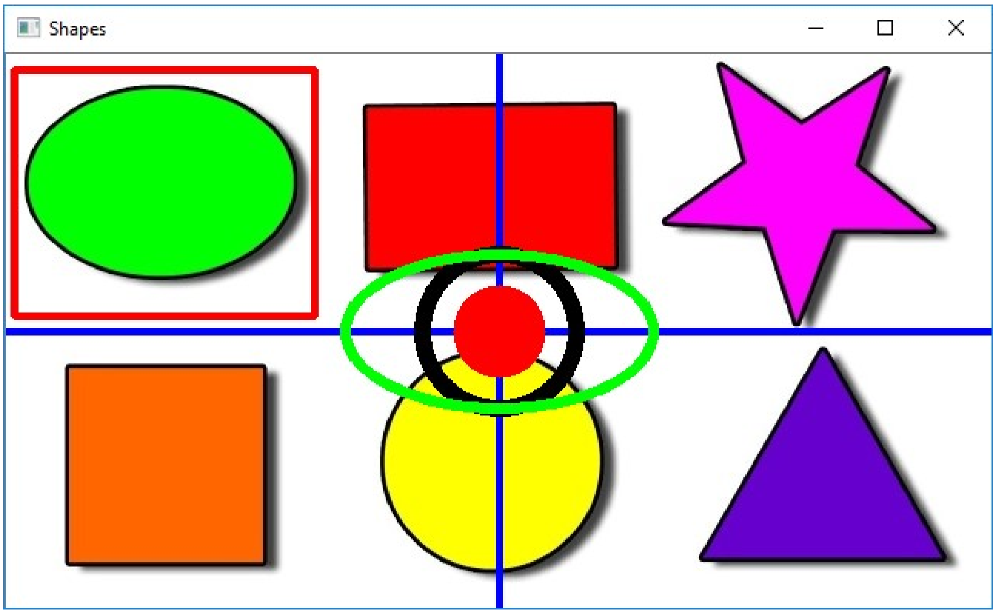

Fig. 1.11: Drawing

(continued from previous page)

```


# create rectangular frame of size 300x300 with name circle
circle = np.zeros((300, 300), dtype = "uint8")
# draw circle in rectangular frame
cv2.circle(circle, (150, 150), 150, 255, -1)
cv2.imshow("Circle", circle)


and_img = cv2.bitwise_and(circle, rectangle)
cv2.imshow("And", and_img)



# another example
rect1 = np.zeros((200, 400), dtype = "uint8")
rect2 = np.zeros((200, 400), dtype = "uint8")

rect1 = cv2.rectangle(rect1, (0, 200), (200, 0), 255, -1)
cv2.imshow("rect1", rect1);
rect2 = cv2.rectangle(rect2, (150, 100), (250, 150), 255, -1)
cv2.imshow("rect2", rect2);

result = cv2.bitwise_and(rect1, rect2);
cv2.imshow("AND", result);

result = cv2.bitwise_or(rect1, rect2);
cv2.imshow("OR", result);

result = cv2.bitwise_xor(rect1, rect2);
cv2.imshow("XOR", result);

result = cv2.bitwise_not(rect2);
cv2.imshow("rect2 NOT", result);

cv2.waitKey()
```

- Output figure is shown Fig. 1.12,

Fig. 1.12: Bitwise operation

1.1.8 Masking

```
# mask_img.py

import cv2
import numpy as np


# Load two images
img = cv2.imread('images/shapes.jpg')
cv2.imshow("Shapes", img)


# create rectangular frame of size 300x300 with name circle
circle_mask = np.zeros(img.shape[:2], dtype="uint8")# draw circle in rectangular frame

# create a circle at (315, 265) to mask the Yellow circle
cv2.circle(circle_mask, (315, 265), 90, 255, -1)
cv2.imshow("Circle", circle_mask)

# mask the Yellow circle
masked_img = cv2.bitwise_and(img, img, mask=circle_mask)
cv2.imshow("Masked image", masked_img)

cv2.waitKey(0)
```

- Output figure is shown Fig. 1.13,

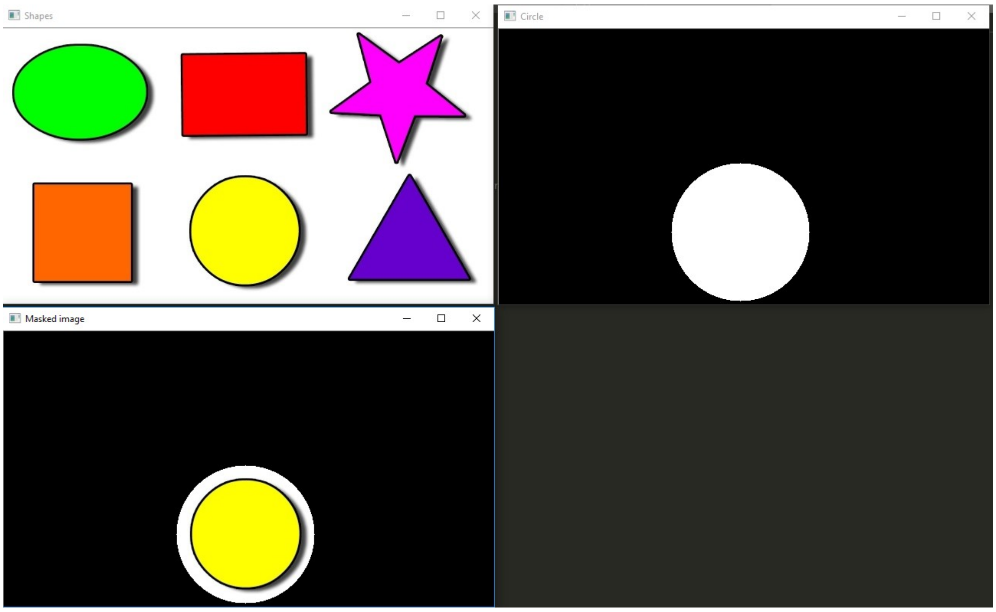

Fig. 1.13: Masking

1.1.9 Edge detection

1.1.9.1 Sobel edge detection

```
# sobel_img.py

import cv2
import numpy as np


# read image
img = cv2.imread("images/lego.jpg")
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.imshow("Lego", gray_img)

# compute gradients along the X and Y axis, respectively
gX = cv2.Sobel(gray_img, cv2.CV_64F, 1, 0)
gY = cv2.Sobel(gray_img, cv2.CV_64F, 0, 1)
#gX value after sobel conversion -52.0
print("gX value after sobel conversion", gX[100,200])

# gX and gY are decimal number with +/- values
# change these values to +ve integer format
gX = cv2.convertScaleAbs(gX)
# gX value after Absolute scaling 52
gY = cv2.convertScaleAbs(gY)
print("gX value after Absolute scaling", gX[100,200])

# combine the sobel X and Y in single image with equal amount
sobelCombined = cv2.addWeighted(gX, 0.5, gY, 0.5, 0)

# show the output images
cv2.imshow("Sobel X", gX)
cv2.imshow("Sobel Y", gY)
cv2.imshow("Sobel Combined", sobelCombined)
```

(continues on next page)

(continued from previous page)

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

- Output figure is shown Fig. 1.14,

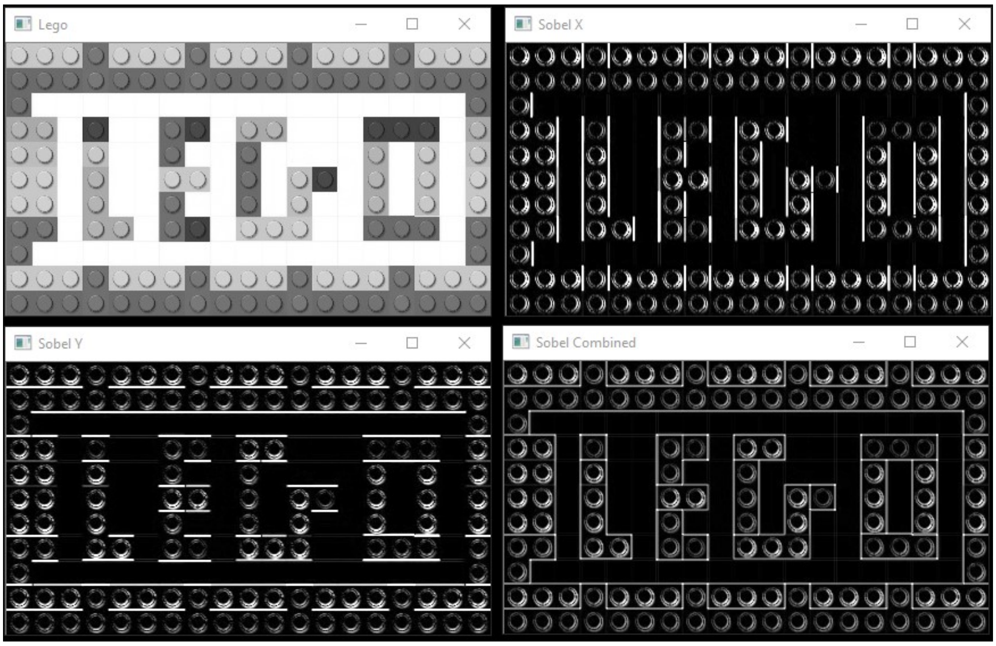

Fig. 1.14: Sobel edge detection

1.1.9.2 Canny edge detection

```python
# canny_img.py

import cv2
import numpy as np


# read image
img = cv2.imread("images/lego.jpg")
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.imshow("Lego", gray_img)

# canny edge detection
# choice depends based on data
canny_wide = cv2.Canny(gray_img, 10, 200) # over detection
canny_medium = cv2.Canny(gray_img, 50, 150) # good detection
canny_narrow = cv2.Canny(gray_img, 200, 250) # missing detection

# show the output images
cv2.imshow("Canny (10, 200)", canny_wide)
cv2.imshow("Canny (50, 150)", canny_medium)
cv2.imshow("Canny (200, 200)", canny_narrow)


cv2.waitKey(0)
```

- Output figure is shown Fig. 1.15,

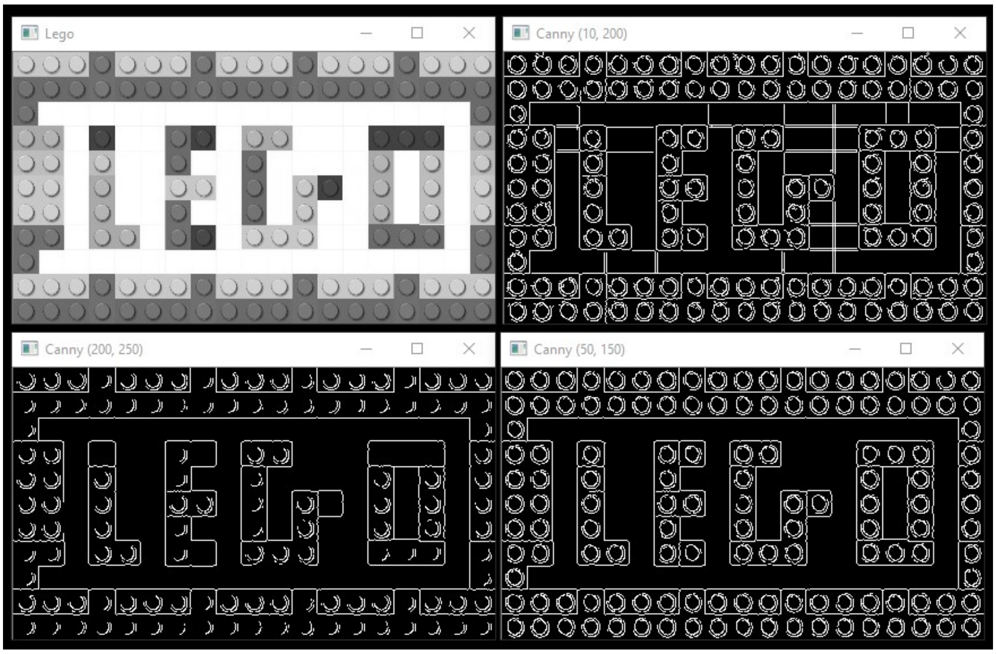

Fig. 1.15: Canny edge detection

Chapter 2

OpenCV with C++

2.1 OpenCV Basics

2.1.1 Introduction

In this section, the procedure to run the C++ code using OpenCV library is shown. Here, “Hello OpenCV” is printed on the screen. Aim is to validate the OpenCV installation and usage therefore the opencv.hpp is included in the code but not used in this example.

- First create the “Hello OpenCV” code as below,

```
// HelloOpenCV.cpp

#include <stdio.h>
#include <opencv2/opencv.hpp>


int main(){
    printf("Hello OpenCV\n");
    return 0;
}
```

- Now, run the code as below,

```
$ g++ HelloOpenCV.cpp -o HelloOpenCV `pkg-config --libs opencv`
$ ./HelloOpenCV
```

2.1.1.1 CMakeLists.txt

ALso, we can crate a CMakeLists.txt file to run the code as below,

- Next, we need to create one CMakeLists.txt file which will included the “OpenCV” library to the path and generate the executable file for the above code,

```
# CMakeLists.txt

cmake_minimum_required(VERSION 2.8)
project( HelloOpenCVExample )
find_package( OpenCV REQUIRED )
include_directories( ${OpenCV_INCLUDE_DIRS} )
add_executable( HelloOpenCV HelloOpenCV.cpp )
target_link_libraries( HelloOpenCV ${OpenCV_LIBS} )
```

- Now, generate the executable as below,

```
$ cmake .
$ make
$ ./HelloOpenCV
Hello OpenCV
```

2.1.2 Load image

```
//  DisplayImage.cpp

// g++ DisplayImage.cpp -o DisplayImage `pkg-config --libs opencv`

// Display the image

#include <stdio.h>
#include <opencv2/opencv.hpp>

int main(int argc, char** argv )
{

    cv::Mat image;  // variable image of datatype Matrix
    image = cv::imread("./OpenCV.png");

    cv::imshow("Display Image", image);
    cv::waitKey(0);
    return 0;
}
```

- Output is shown Fig. 2.1,

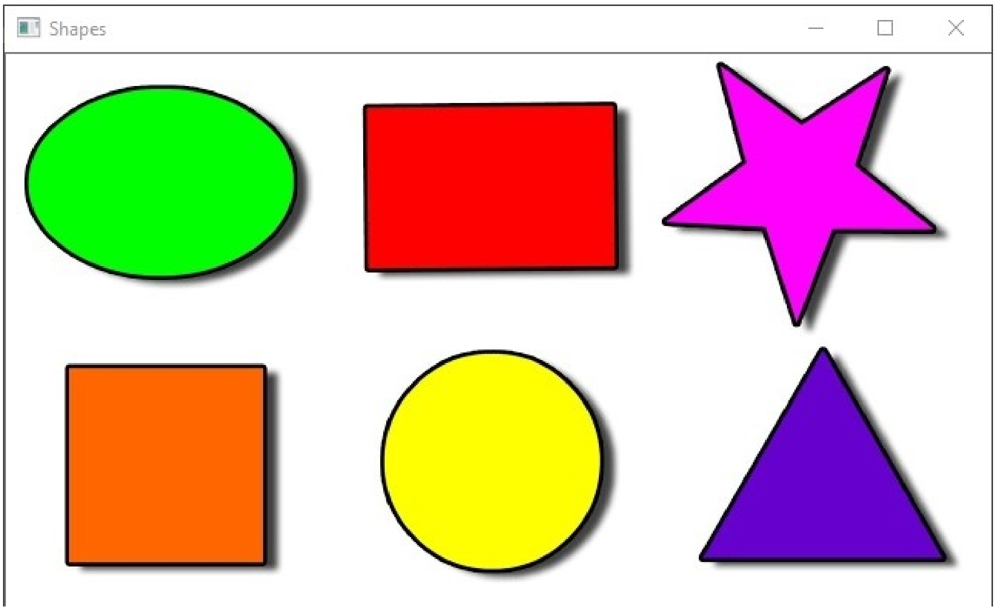

Fig. 2.1: Shapes

2.1.3 Load Video

```
// DisplayVideo.cpp


```

(continues on next page)

(continued from previous page)

```
#include <stdio.h>
#include <opencv2/opencv.hpp>

int main(int argc, char** argv )
{

    cv::Mat frame;  // variable frame of datatype Matrix
    cv::VideoCapture capture;
    capture.open("versal.mp4");

    for(;;){
        capture>>frame;
        if(frame.empty())
            break;
        cv::imshow("Window", frame);

        if(cv::waitKey(30)>=0)
                break;
    }
    return 0;
}
```

```
$  g++ DisplayVideo.cpp -o DisplayVideo `pkg-config --libs opencv`
$ ./DisplayVideo
```

2.1.4 Basic operations on images

2.1.4.1 Accessing and modifying pixel

```
// access_modify_pixel.py


#include <stdio.h>
#include <opencv2/opencv.hpp>

using namespace std;
using namespace cv;

int main(int argc, char** argv )
{

    cv::Mat img;  // variable image of datatype Matrix
    img = cv::imread("images/shapes.jpg");

    // For color image i.e. 3 channel
    Vec3b intensity = img.at<Vec3b>(10, 10);
    cout << "BGR " << intensity << "\n";

    // print individual component [B G R]
    int blue = intensity.val[0];
    cout << "blue " << blue << "\n";
    int green = intensity.val[1];
    cout << "green " << green << "\n";
    int red = intensity.val[2];
    cout << "red " << red << "\n";


    // modify pixel
```

(continues on next page)

(continued from previous page)

```
    img.at<Vec3b>(10, 10) = (0, 0, 255);
    // For color image i.e. 3 channel
    intensity = img.at<Vec3b>(10, 10);
    cout << "BGR after modification " << intensity << "\n";

    cv::imshow("Display Image", img);
    cv::waitKey(0);

    return 0;
}
```

```
g++ access_modify_pixel.cpp -o out `pkg-config --libs opencv` && ./out
```

- Output is shown Fig. 2.2,

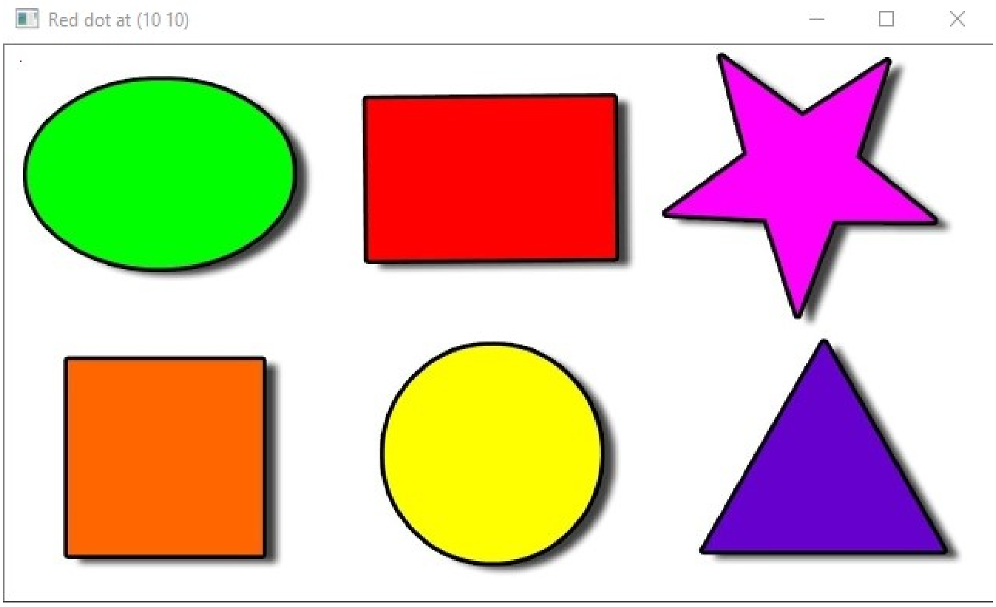

Fig. 2.2: Red dot at (10 10)

2.1.4.2 Split and Merge

- In this section, the color image is split and plotted into R, G and B color. Also, these R, G and B are merged together to get the original image.

```
// split_merge.cpp

#include <stdio.h>
#include <opencv2/opencv.hpp>

using namespace std;
using namespace cv;

int main(int argc, char** argv )
{

    cv::Mat img, sum_rgb;  // variable image of datatype Matrix
    img = cv::imread("images/shapes.jpg");
    cv::imshow("Display Image", img);

```

(continues on next page)

(continued from previous page)

```
    // three channel to store b, g, r
    cv::Mat rgbchannel[3];

    // split image
    cv::split(img, rgbchannel);

    // plot individual component
    cv::namedWindow("Blue",CV_WINDOW_AUTOSIZE);
    cv::imshow("Red", rgbchannel[0]);

    cv::namedWindow("Green",CV_WINDOW_AUTOSIZE);
    cv::imshow("Green", rgbchannel[1]);

    cv::namedWindow("Red",CV_WINDOW_AUTOSIZE);
    cv::imshow("Blue", rgbchannel[2]);

    // merge : (input, num_of_channel, output)
    cv::merge(rgbchannel, 3, sum_rgb);
    cv::imshow("Merged", sum_rgb);

    cv::waitKey(0);

    return 0;
}
```

```
g++ split_merge.cpp -o out `pkg-config --libs opencv` && ./out
```

- Output is shown Fig. 2.3,

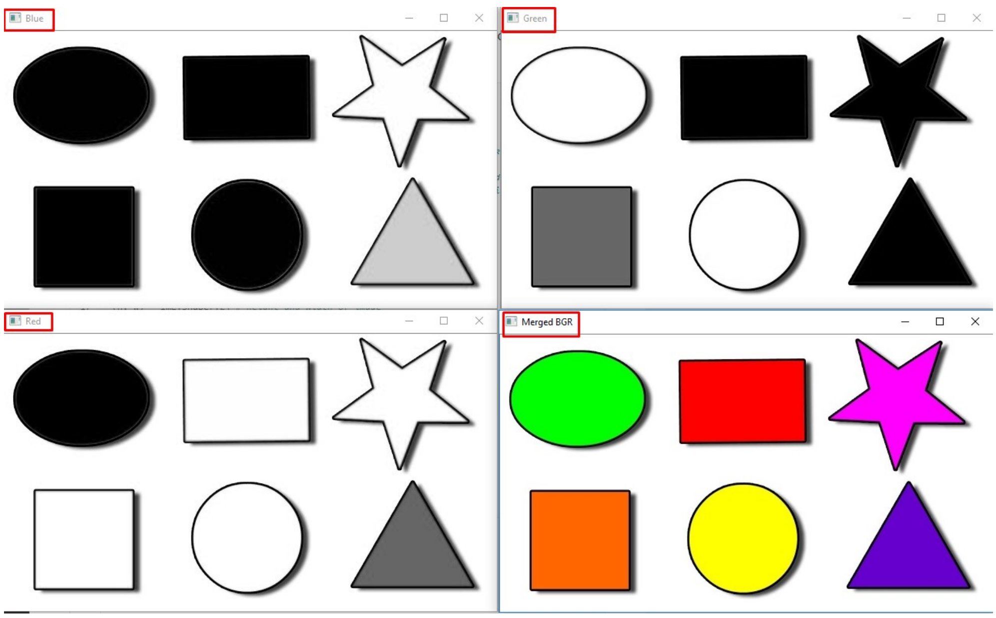

Fig. 2.3: Split and merge

2.1.4.3 Crop image

In this section, we will crop the image in 4 equal part and change the color of 2 parts.

```
// crop_img.cpp

```

(continues on next page)

(continued from previous page)

```cpp
#include <stdio.h>
#include <opencv2/opencv.hpp>

using namespace std;
using namespace cv;

int main(int argc, char** argv )
{

    cv::Mat img;  // variable image of datatype Matrix
    cv::Mat top_left, top_right, bottom_left, bottom_right;
    int w, h, cX, cY;

    img = cv::imread("images/shapes.jpg");

    cout << "(width, height)"<< img.size() << endl;
    cout << "Width : " << img.cols << endl;
    cout << "Height: " << img.rows << endl;

    w = img.size().width;
    h = img.size().height;

    cX = (int)w/2;
    cY = (int)h/2;
    cout << "(cX, cY) = (" << cX << ", " << cY << ")" << endl;


    // (start_x, start_y, len_x, len_y)
    cv::Rect top_left_roi(0, 0, cX, cY);
    top_left = img(top_left_roi);
    cv::imshow("Top left", top_left);

    cv::Rect top_right_roi(cX, 0, cX, cY);
    top_right = img(top_right_roi);
    cv::imshow("Top right", top_right);

    cv::Rect bottom_left_roi(0, cY, cX, cY);
    bottom_left = img(bottom_left_roi);
    cv::imshow("Bottom left", bottom_left);

    cv::Rect bottom_right_roi(cX, cY, cX, cY);
    bottom_right = img(bottom_right_roi);
    cv::imshow("Bottom right", bottom_right);

    // or use above or below, both have same results
    // // (start_x, start_y, len_x, len_y)
    // cv::Rect top_left_roi(0, 0, cX, cY);
    // top_left = img(top_left_roi);
    // cv::imshow("Top left", top_left);

    // cv::Rect top_right_roi(cX, 0, w - cX, cY);
    // top_right = img(top_right_roi);
    // cv::imshow("Top right", top_right);

    // cv::Rect bottom_left_roi(0, cY, cX, h - cY);
    // bottom_left = img(bottom_left_roi);
    // cv::imshow("Bottom left", bottom_left);

    // cv::Rect bottom_right_roi(cX, cY, w - cX, h - cY);
    // bottom_right = img(bottom_right_roi);
    // cv::imshow("Bottom right", bottom_right);
```

(continues on next page)

(continued from previous page)

```


    cv::waitKey(0);

    return(0);
}
```

```
g++ crop_img.cpp -o out `pkg-config --libs opencv` && ./out
```

- Output is shown Fig. 2.4,

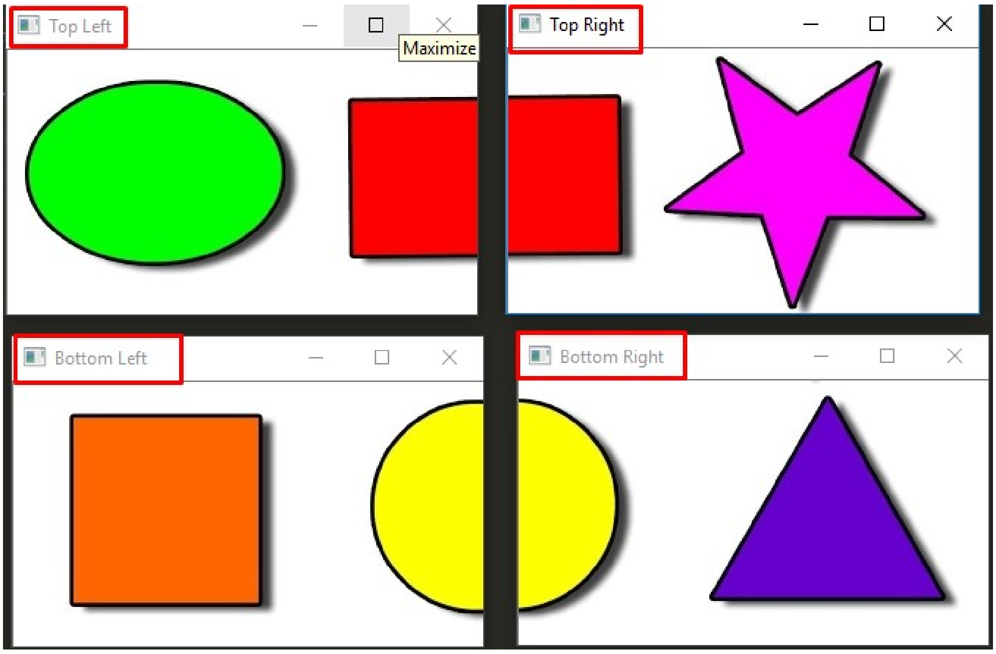

Fig. 2.4: Crop image

2.1.4.4 Image arithmetic

- OpenCV sets the maximum and minimum as 255 and 0 respectively.

```
// image_arith.cpp


#include <stdio.h>
#include <opencv2/opencv.hpp>

using namespace std;
// using namespace cv;

int main(int argc, char** argv )
{


    // ############# Various method to define Matrix ################

    // intialize matrix with contant value 80
    cv::Mat matB(3, 3, CV_8UC1, cv::Scalar(80));
    cout << "matB = " << endl << " " << matB << endl << endl;

```

(continues on next page)

(continued from previous page)

```
    // zero matrix
    cv::Mat matZeros = cv::Mat::zeros(3,3, CV_8UC1);
    cout << "matZeros = " << endl << " " << matZeros << endl << endl;

    // eye matrix
    cv::Mat matEye = cv::Mat::eye(3, 3, CV_64F);
    cout << "matEye = " << endl << " " << matEye << endl << endl;

    // ones matrix
    cv::Mat matOnes = cv::Mat::ones(3, 3, CV_32F);
    cout << "matOnes = " << endl << " " << matOnes << endl << endl;

    float data[10] = { 221, 23, 9, 104, 51, 65, 76, 48, 210 };
    cv::Mat A = cv::Mat(3, 3, CV_32F, data);
    cout << "A = " << endl << " " << A << endl << endl;

    cv::Mat B(3, 3, CV_8UC1, cv::Scalar(80));
    cout << "B = " << endl << " " << B << endl << endl;

    // convert format
    cv::Mat A_convert = cv::Mat(3, 3, CV_8UC1);
    A.convertTo(A_convert, CV_8UC1);
    cout << "A_convert = " << endl << " " << A_convert << endl << endl;


    // define 3x3 matrix
    cv::Mat matOut = cv::Mat(3, 3, CV_8UC1);

    // ######################## Add/subtract ####################################

    // cv::add(A_convert, B, matOut) is not possible due to different data type
    cv::add(A_convert, B, matOut);
    cout << "A_convert + B = \n" << matOut << endl << endl;

    // subtract
    cv::subtract(A_convert, B, matOut);
    cout << "A_convert - B = \n" << matOut << endl << endl;


    // ############ Image addtion

    cv::Mat img, add_img, sub_img;  // variable image of datatype Matrix

    // read image
    img = cv::imread("images/shapes.jpg");
    cv::imshow("Shapes", img);

    // define new mat with same size as img
    cv::Mat new_pixel = 90 * cv::Mat::ones(img.size(), img.type());

    // add and show
    cv::add(img, new_pixel, add_img);
    cv::imshow("Add image", add_img);

    // subtract and show
    cv::subtract(img, new_pixel, sub_img);
    cv::imshow("Subtract image", sub_img);


    cv::waitKey(0);

```

(continues on next page)

(continued from previous page)

```
    return(0);
}
```

```
g++ image_arith.cpp -o out `pkg-config --libs opencv` && ./out
```

- Output will be as below,

```
OpenCV 250 + 10:   [[255]]
Numpy 250 + 10:  [4]
Initial pixel at [50, 50]   :  [  1 255   0]
Add/subtract 90
OpenCV addition pixel at [50, 50]   :  [ 91 255  90]
OpenCV subtract pixel at [50, 50]   :  [  0 165   0]
Numpy addition pixel at [50, 50]    :  [  0 165   0]
Numpy subtract pixel at [50, 50]    :  [  0 165   0]
```

- Output figure is shown Fig. 2.5,

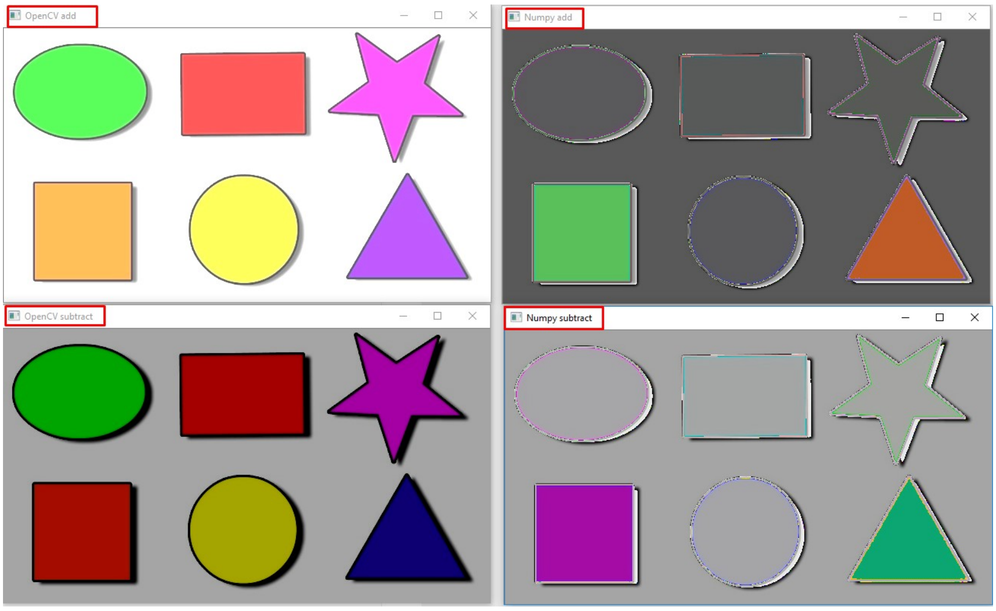

Fig. 2.5: Image arithmetic

2.1.4.5 Threshold

For every pixel, the same threshold value is applied. If the pixel value is smaller than the threshold, it is set to 0, otherwise it is set to a maximum value.

```
// threshold_img.cpp


#include <stdio.h>
#include <opencv2/opencv.hpp>

using namespace std;

int main(int argc, char** argv )
{

    cv::Mat img, thresh_img;  // variable image of datatype Matrix
```

(continues on next page)

(continued from previous page)

```

    img = cv::imread("images/rose.jpg");
    cv::imshow("Rose", img);

    cv::threshold(img, thresh_img, 100, 255, cv::THRESH_BINARY);
    cv::imshow("Threshold Binary", thresh_img);

    cv::threshold(img, thresh_img, 100, 255, cv::THRESH_BINARY_INV);
    cv::imshow("Threshold Binary Inverse", thresh_img);

    cv::waitKey(0);
    return 0;
}
```

- Output figure is shown Fig. 2.6,

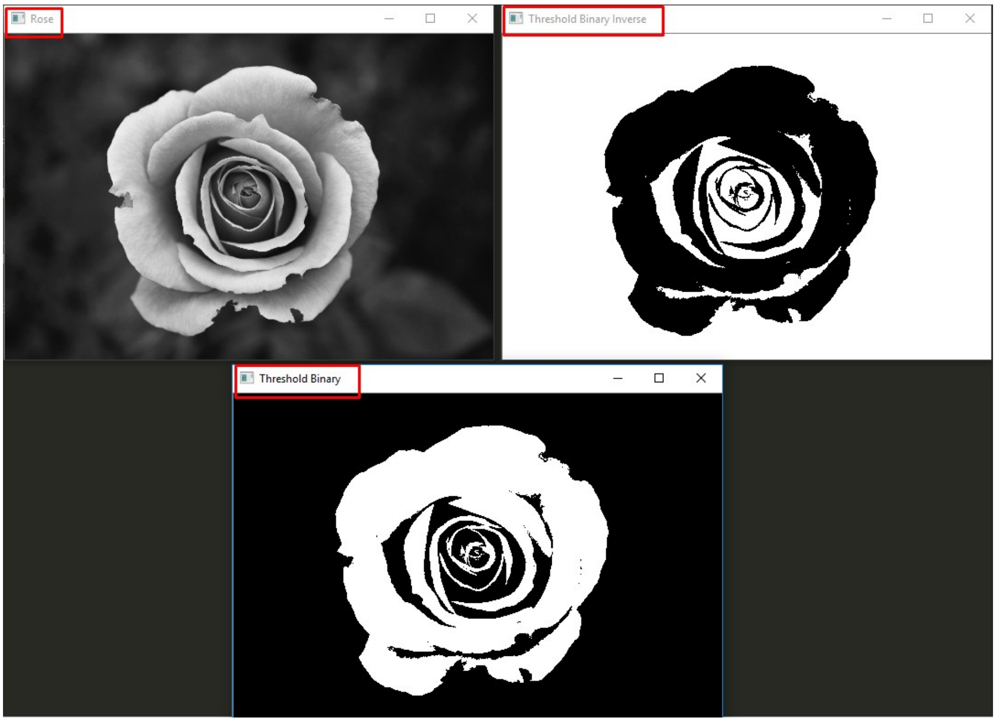

Fig. 2.6: Threshold

2.1.5 Geometric Transformations

2.1.5.1 Scaling

Scaling is just resizing of the image. OpenCV comes with a function cv2.resize() for this purpose.

```
// scale_img.cpp

#include <stdio.h>
#include <opencv2/opencv.hpp>

using namespace std;

int main(int argc, char** argv )
{
```

(continues on next page)

(continued from previous page)

```

    cv::Mat img, resize_img;  // variable image of datatype Matrix
    int w, h;

    img = cv::imread("images/shapes.jpg");
    cv::imshow("Shapes", img);

    w = img.size().width;
    h = img.size().height;

    cv::resize(img, resize_img, cv::Size((int)w/2, (int)h/2), cv::INTER_CUBIC);
    cv::imshow("Resize Shapes", resize_img);

    cv::waitKey(0);
    return 0;
}
```

```
g++ scale_img.cpp -o out `pkg-config --libs opencv` && ./out
```

- Output figure is shown Fig. 2.7,

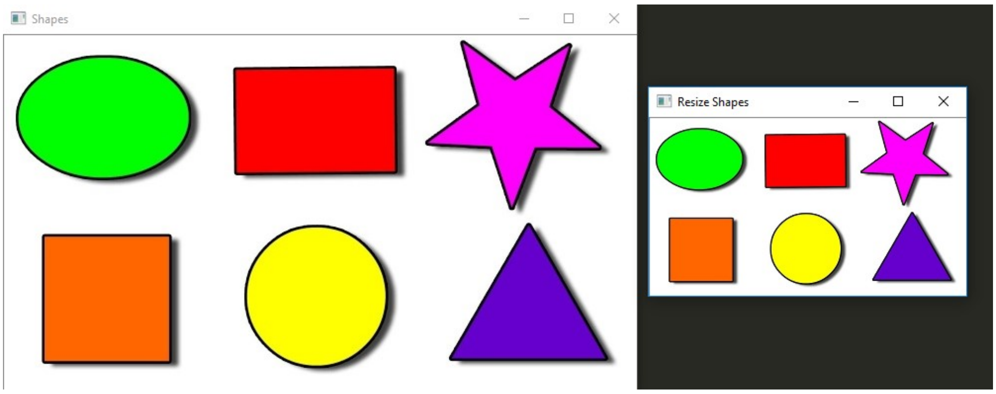

Fig. 2.7: Resize or scaling

2.1.5.2 Flip

Three types of flips are possible,

- 0 : Horizontal flip
- 1 : Vertical flip
- -1 : Both horizontal and vertical flip

```
// flip_img.cpp

#include <stdio.h>
#include <opencv2/opencv.hpp>

using namespace std;

int main(int argc, char** argv )
{

    cv::Mat img, flip_horizontal, flip_vertical, flip_both;  // variable image of datatype Matrix

    img = cv::imread("images/shapes.jpg");
    cv::imshow("Shapes", img);
```

(continues on next page)

(continued from previous page)

```

    // flip horizontal
    cv::flip(img, flip_horizontal, 0);
    cv::imshow("Horizontal Flip", flip_horizontal); // display image

    // flip vertical
    cv::flip(img, flip_vertical, 1);
    cv::imshow("Vertical Flip", flip_vertical); // display image

    // flip vertical and horizontal both
    cv::flip(img, flip_both, -1);
    cv::imshow("Horizontal and Vertical Flip", flip_both); // display image

    cv::waitKey(0);
    return 0;
}
```

```
g++ flip_img.cpp -o out `pkg-config --libs opencv` && ./out
```

- Output figure is shown Fig. 2.8,

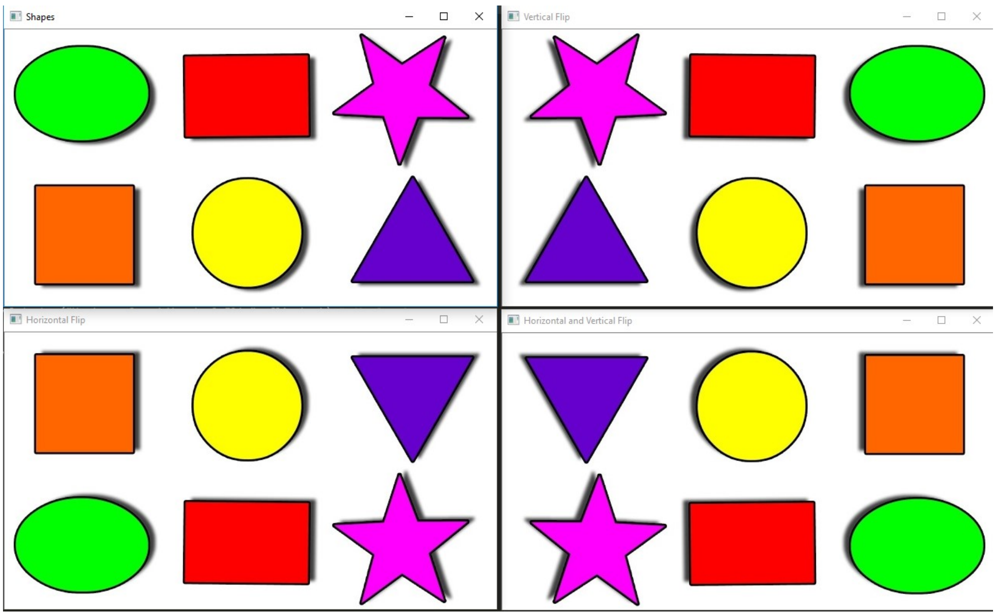

Fig. 2.8: Flip

2.1.5.3 Translation

Translation is the shifting of object's location.

```
// translate_img.cpp

#include <stdio.h>
#include <opencv2/opencv.hpp>

using namespace std;

int main(int argc, char** argv )
{

```

(continues on next page)

(continued from previous page)

```
    cv::Mat img, shift_img;  // variable image of datatype Matrix

    // create shift matrix
    float data[6] = { 1, 0, 30, 0, 1, 50 };
    cv::Mat shift_matrix_float = cv::Mat(2, 3, CV_32F, data);
    cout << shift_matrix_float;

    // convert to CV_64F format
    cv::Mat shift_matrix = cv::Mat(2, 3, CV_64F);
    shift_matrix_float.convertTo(shift_matrix, CV_64F);

    img = cv::imread("images/shapes.jpg");
    cv::imshow("Shapes", img);

    // flip horizontal
    cv::warpAffine(img, shift_img, shift_matrix, img.size());
    cv::imshow("Translate Flip", shift_img); // display image


    cv::waitKey(0);
    return 0;
}
```

```
g++ translate_img.cpp -o out `pkg-config --libs opencv` && ./out
```

- Output figure is shown Fig. 2.9,

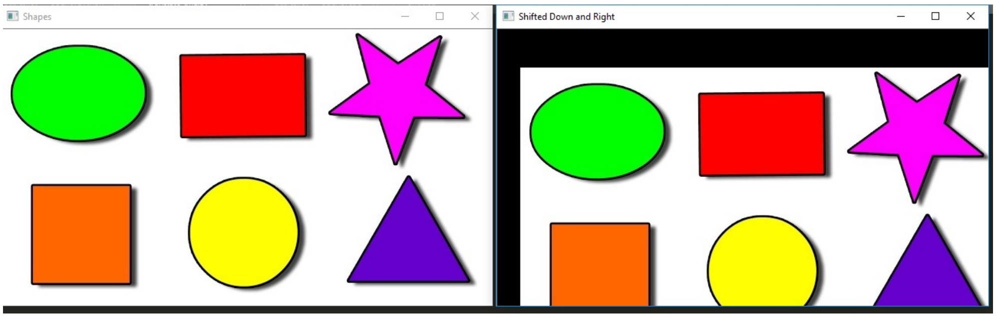

Fig. 2.9: Translation

2.1.5.4 Rotation

- We need to define the rotation angle along with a point for rotation.

```
// rotate_img.cpp


#include <stdio.h>
#include <opencv2/opencv.hpp>

using namespace std;

int main(int argc, char** argv ){

    cv::Mat img, rotate_matrix, rotated;
    int w, h; // width, height

    // read image
```

(continues on next page)

(continued from previous page)

```
    img = cv::imread("images/shapes.jpg");
    cv::imshow("Shapes", img);

    // width and height of image
    w = img.size().width;
    h = img.size().height;

    // rotation points
    cv::Point2f rotation_center(w/2, h/2);

    rotate_matrix = cv::getRotationMatrix2D(rotation_center, 45, 1.0);

    cv::warpAffine(img, rotated, rotate_matrix, img.size());
    cv::imshow("Rotated by 45 Degrees", rotated);

    cv::waitKey(0);
    return 0;
}
```

```
g++ rotate_img.cpp -o out `pkg-config --libs opencv` && ./out
```

- Output figure is shown Fig. 2.10,

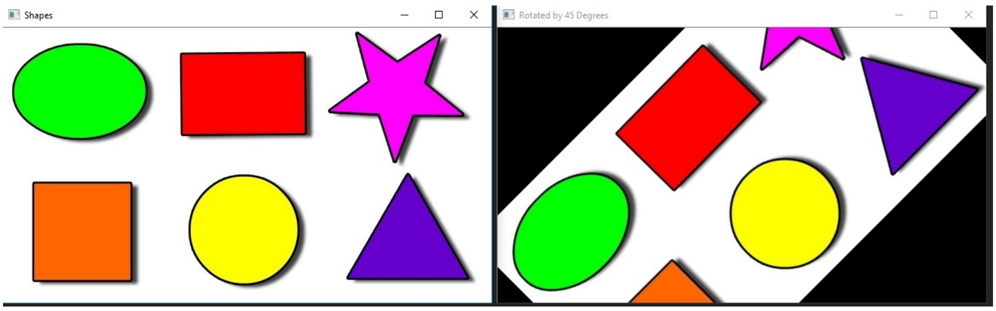

Fig. 2.10: Rotation

2.1.6 Drawing

- In this section, lines, rectangle, circle and ellipse are drawn using OpenCV.

```
// drawing_img.cpp

#include <stdio.h>
#include <opencv2/opencv.hpp>

using namespace std;

int main(int argc, char** argv ){

    cv::Mat img, rotate_matrix, rotated;
    int w, h; // width, height

    // read image
    img = cv::imread("images/shapes.jpg");
    cv::imshow("Shapes", img);

    // width and height of image
    w = img.size().width;
```

(continues on next page)

(continued from previous page)

```
    h = img.size().height;

    // draw horizontal line
    cv::line( img, cv::Point( 0, (int)h/2 ), cv::Point( w, (int)h/2), cv::Scalar( 255, 0, 0 ), 3);
    // draw vertical line
    cv::line( img, cv::Point( (int)w/2, 0 ), cv::Point( (int)w/2, h), cv::Scalar( 255, 0, 0 ), 3);

    // draw rectangle
    cv::rectangle( img, cv::Point(5, 10), cv::Point(200, 170), cv::Scalar( 0, 0, 255 ), 3);

    // draw circle
    // center coordinates (w//2, h//2) and radius (50) are
    // required to to draw circle. 10 is the line width
    cv::circle(img, cv::Point((int)w/2, (int)h/2), 50, cv::Scalar(0, 0, 0), 10); // black
    cv::circle(img, cv::Point((int)w/2, (int)h/2), 30, cv::Scalar(0, 0, 255), -1); // -1 : filled circle

    cv::imshow("Shapes", img);

    cv::waitKey(0);
    return 0;
}
```

```
g++ drawing_img.cpp -o out `pkg-config --libs opencv` && ./out
```

- Output figure is shown Fig. 2.11,

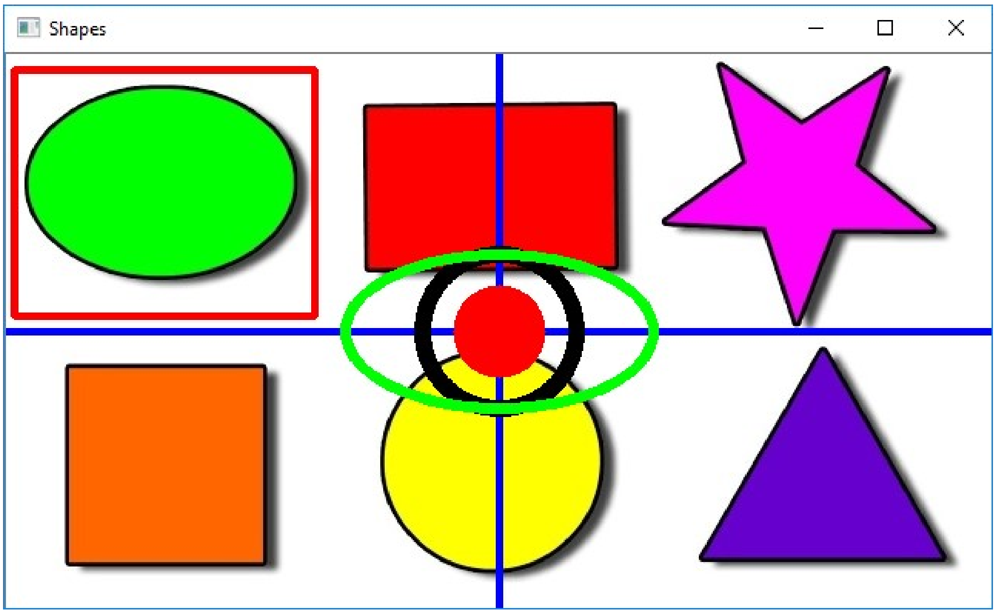

Fig. 2.11: Drawing

2.1.7 Bitwise operation

```
// bitwise_img.cpp

#include <stdio.h>
#include <opencv2/opencv.hpp>

using namespace std;

```

(continues on next page)

(continued from previous page)

```
int main(int argc, char** argv ){

    cv::Mat and_img;

    // create and display frame of size 300 for rectangle and circle
    cv::Mat rectangle(300, 300, CV_8UC1, cv::Scalar(0)); // rectangle
    cv::Mat circle(300, 300, CV_8UC1, cv::Scalar(0)); // circle


    // draw and show rectangle
    cv::rectangle( rectangle, cv::Point(20, 20), cv::Point(280, 280), cv::Scalar( 255 ), -1);
    cv::imshow("Rectangle", rectangle);

    // draw and show circle
    cv::circle(circle, cv::Point(150, 150), 150, cv::Scalar(255), -1); // black
    cv::imshow("Circle", circle);

    // bitwise and operation
    cv::bitwise_and(circle, rectangle, and_img);
    cv::imshow("And", and_img);



    // another example
    cv::Mat rect1 = cv::Mat::zeros( cv::Size(400,200), CV_8UC1);
    cv::Mat rect2 = cv::Mat::zeros( cv::Size(400,200), CV_8UC1);


    rect1( cv::Range(0, 200), cv::Range(0, 200) ) = 255;
    cv::imshow("rect1", rect1);

    rect2( cv::Range(100, 150), cv::Range(150, 250) ) = 255;
    cv::imshow("rect2", rect2);

    cv::Mat result;

    bitwise_and(rect1, rect2, result);
    cv::imshow("AND", result);

    bitwise_or(rect1, rect2, result);
    cv::imshow("OR", result);

    bitwise_xor(rect1, rect2, result);
    cv::imshow("XOR", result);

    bitwise_not(rect2, result);
    cv::imshow("rect2 NOT", result);


    cv::waitKey(0);
    return 0;
}
```

```
g++ bitwise_img.cpp -o out `pkg-config --libs opencv` && ./out
```

- Output figure is shown Fig. 2.12,

Fig. 2.12: Bitwise operation

2.1.8 Masking

```
// mask_img.cpp

#include <stdio.h>
#include <opencv2/opencv.hpp>

using namespace std;

int main(int argc, char** argv ){

    cv::Mat img, masked_img;
    int w, h; // width, height

    // read image
    img = cv::imread("images/shapes.jpg");
    cv::imshow("Shapes", img);

    // width and height of image
    w = img.size().width;
    h = img.size().height;


    cv::Mat circle = cv::Mat::zeros( cv::Size(w, h), CV_8UC3);
    cv::circle(circle, cv::Point(315, 265), 90, cv::Scalar(255, 255, 255), -1); // black
    cv::imshow("Circle", circle);

    cv::bitwise_and(img, circle, masked_img);
    cv::imshow("Masked image", masked_img);
```

(continues on next page)

(continued from previous page)

```

    cv::waitKey(0);
    return 0;
}
```

```
g++ mask_img.cpp -o out `pkg-config --libs opencv` && ./out
```

- Output figure is shown Fig. 2.13,

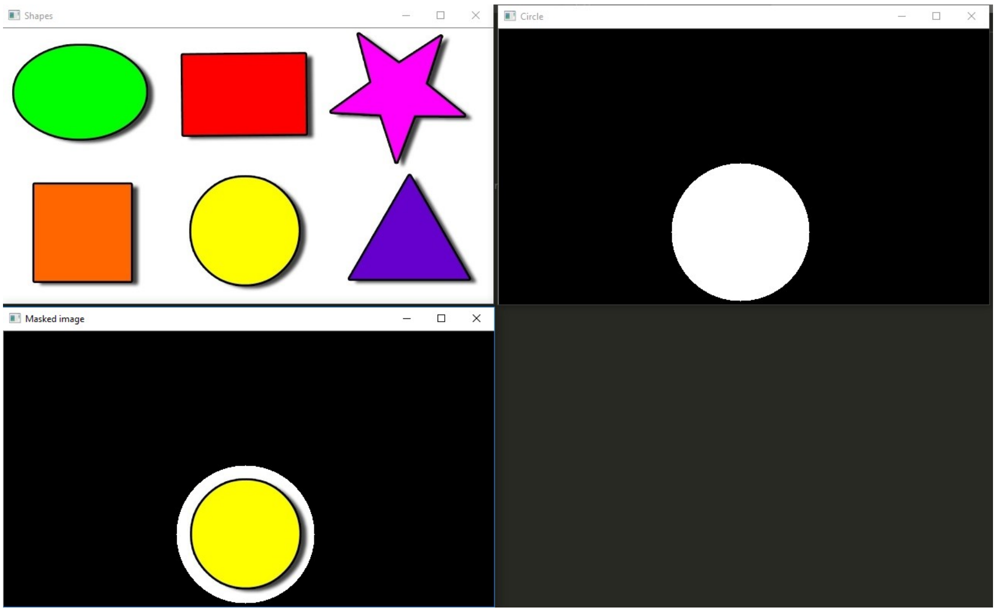

Fig. 2.13: Masking

2.1.9 Edge detection

2.1.9.1 Sobel edge detection

```
// sobel_img.cpp

#include <stdio.h>
#include <opencv2/opencv.hpp>

using namespace std;

int main(int argc, char** argv ){

    cv::Mat img, gray_img;
    int w, h; // width, height

    // read image
    img = cv::imread("images/lego.jpg");
    cv::imshow("Lego color", img);

    // width and height of image
    w = img.size().width;
    h = img.size().height;
```

(continues on next page)

(continued from previous page)

```
    cv::cvtColor(img, gray_img, cv::COLOR_BGR2GRAY);
    cv::imshow("Lego ", gray_img);

    cv::Mat gX, gY;
    //  compute gradients along the X and Y axis, respectively
    cv::Sobel(gray_img, gX, CV_64F, 1, 0);
    cv::Sobel(gray_img, gY, CV_64F, 0, 1);
    // gX value after sobel conversion -52.0
    cout << "gX value after sobel conversion: " << (int)gX.at<double>(100, 200) << endl;


    // gX and gY are decimal number with +/- values
    // change these values to +ve integer format
    cv::convertScaleAbs(gX, gX);
    // gX value after Absolute scaling 52
    cv::convertScaleAbs(gY, gY);
    cout << "gX value after Absolute scaling: " << (int)gX.at<uchar>(100, 200) << endl;


    cv::Mat sobelCombined;
     cv::addWeighted(gX, 0.5, gY, 0.5, 0, sobelCombined);

    // show the output images
    cv::imshow("Sobel X", gX);
    cv::imshow("Sobel Y", gY);
    cv::imshow("Sobel Combined", sobelCombined);

    cv::waitKey(0);
    return 0;
}
```

```
g++ sobel_img.cpp -o out `pkg-config --libs opencv` && ./out
```

- Output figure is shown Fig. 2.14,

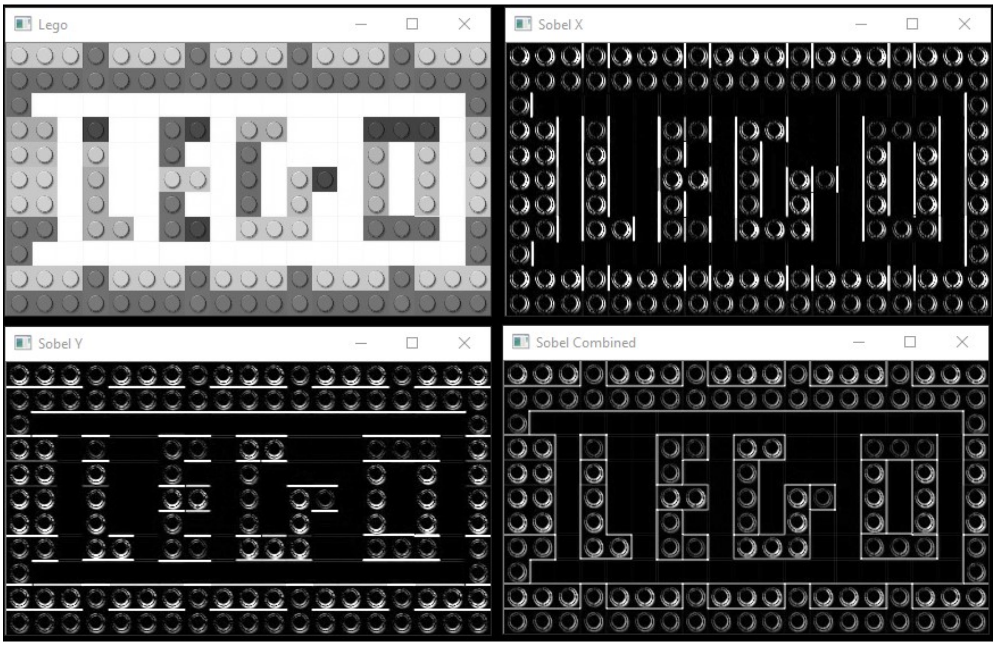

Fig. 2.14: Sobel edge detection

2.1.9.2 Canny edge detection

```
// canny_img.cpp

#include <stdio.h>
#include <opencv2/opencv.hpp>

using namespace std;

int main(int argc, char** argv ){

    cv::Mat img, gray_img;
    int w, h; // width, height

    // read image
    img = cv::imread("images/lego.jpg");
    cv::imshow("Lego color", img);

    // width and height of image
    w = img.size().width;
    h = img.size().height;


    cv::cvtColor(img, gray_img, cv::COLOR_BGR2GRAY);
    cv::imshow("Lego ", gray_img);

    cv::Mat canny_wide, canny_medium, canny_narrow;

    cv::Canny(gray_img, canny_wide, 10, 200);
    cv::Canny(gray_img, canny_medium, 50, 150);
    cv::Canny(gray_img, canny_narrow, 200, 250);

    // show the output images
    cv::imshow("Canny (10, 200)", canny_wide);
    cv::imshow("Canny (50, 150)", canny_medium);
    cv::imshow("Canny (200, 250)", canny_narrow);


    cv::waitKey(0);
    return 0;
}
```

```
g++ canny_img.cpp -o out `pkg-config --libs opencv` && ./out
```

- Output figure is shown Fig. 2.15,

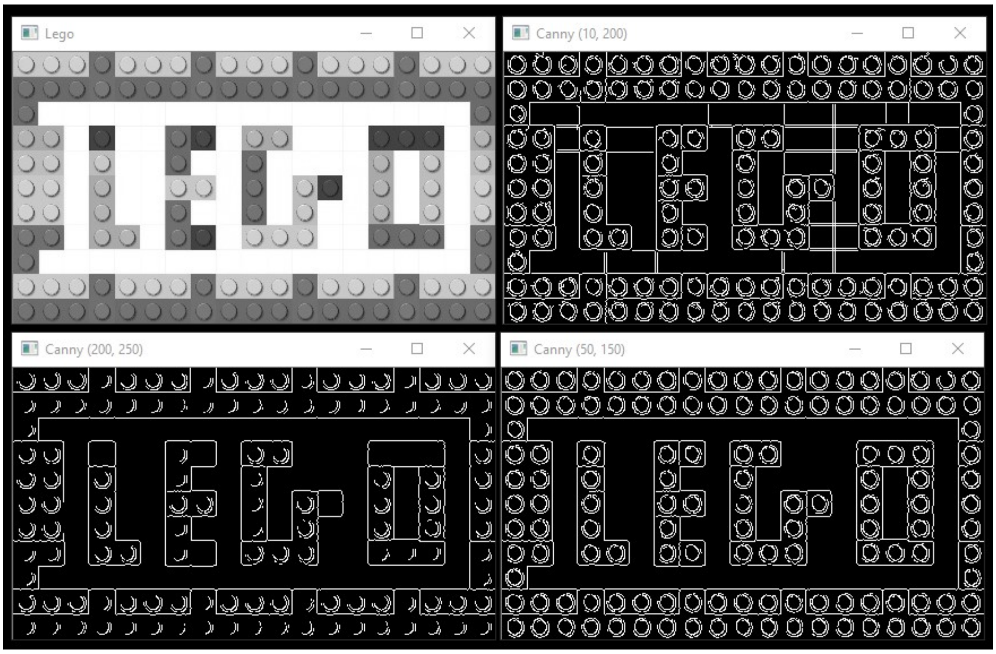

Fig. 2.15: Canny edge detection

Chapter 3

OpenCV with petalinux

3.1 OpenCV Basics

3.1.1 Create Petalinux project

```
(activate the petalinux)
$ ts -petalinux 2018.2

(download the bsp first, and then use the below command)
$ petalinux-create -t project -n opencvExamples -s /proj/css/meherp/bsp/v2018_2/xilinx-zcu104-v2018.2-
↪final.bsp

$ cd opencvExamples/


$ petalinux-config -c rootfs
    (select following options-> save -> exit)

    Filesystem Packages ---> libs ---> libmali-xlnx ---> [*] libmali-xlnx

    Petalinux Package Groups ---> packagegroup-petalinux-display-debug ---> [*] packagegroup-petalinux-
↪display-debug

    Petalinux Package Groups ---> packagegroup-petalinux-opencv ---> [*] packagegroup-petalinux-opencv

    Petalinux Package Groups ---> packagegroup-petalinux-v4lutils ---> [*] packagegroup-petalinux-
↪v4lutils

    Petalinux Package Groups ---> packagegroup-petalinux-x11 ---> [*] packagegroup-petalinux-x11


$ petalinux-build


(Next, create a petalinux application as below)
$ petalinux-create -t apps -n ocvtest --enable
```

- Above will create a default 'hello world' app. Now, we need to modify the code for rotating the image by 90 degree. Modify the below files (vi editor is used below)

Note: If there are some error during petalinux-build then recreated the project and remove below line. Then run the petlainux build command,

```
do_compile() {
     oe_runmake
}
```

```
$ vi project-spec/meta-user/recipes-apps/ocvtest/ocvtest.bb
# (replace the code with below code)

#
# This file is the ocvtest recipe.
#

SUMMARY = "Simple ocvtest application"
SECTION = "PETALINUX/apps"
LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"

SRC_URI = "file://ocvtest.cpp \
           file://CMakeLists.txt \
          "

S = "${WORKDIR}"

DEPENDS += "opencv"

inherit pkgconfig cmake

do_compile() {
         oe_runmake
}

do_install() {
         install -d ${D}${bindir}
         install -m 0755 ocvtest ${D}${bindir}
}
```

- Modify CMakeLists.txt

```
$ vi project-spec/meta-user/recipes-apps/ocvtest/files/CMakeLists.txt
# (replace the code with below code)


# cmake needs this line
cmake_minimum_required(VERSION 2.8)

add_definitions(-std=c++11 -Werror=return-type)

# Define project name
project(ocvtest)

# Find OpenCV, you may need to set OpenCV_DIR variable
# to the absolute path to the directory containing OpenCVConfig.cmake file
# via the command line or GUI
find_package(OpenCV REQUIRED)

# If the package has been found, several variables will
# be set, you can find the full list with descriptions
# in the OpenCVConfig.cmake file.
# Print some message showing some of them
message(STATUS "OpenCV library status:")
message(STATUS "    version: ${OpenCV_VERSION}")
```

(continues on next page)

(continued from previous page)

```
message(STATUS "    libraries: ${OpenCV_LIBS}")
message(STATUS "    include path: ${OpenCV_INCLUDE_DIRS}")

if(CMAKE_VERSION VERSION_LESS "2.8.11")
  # Add OpenCV headers location to your include paths
  include_directories(${OpenCV_INCLUDE_DIRS})
endif()


# Declare the executable target built from your sources
add_executable(ocvtest ocvtest.cpp)

# Link your application with OpenCV libraries
target_link_libraries(ocvtest ${OpenCV_LIBS})
```

- Modify makefile

```
$ vi project-spec/meta-user/recipes-apps/ocvtest/files/Makefile
# replace the code with below code,



APP = ocvtest

# Add any other object files to this list below
APP_OBJS = ocvtest.o

all: build

build: $(APP)

CXXFLAGS += $(shell pkg-config --cflags opencv)
LDFLAGS += $(shell pkg-config --libs opencv)

$(APP): $(APP_OBJS)
    $(CXX) $(CXXFLAGS) $(LDFLAGS) -o $@ $(APP_OBJS) $(LDLIBS)

clean:
    rm -f $(APP) *.elf *.gdb *.o
```

3.1.2 Write OpenCV code

Below OpenCV code is exactly same as ‘OpenCV code for C++ i.e. bitwise_img.cpp’.

- Modify ocvtest.cpp

```
// ocvtest.cpp

#include <stdio.h>
#include <opencv2/opencv.hpp>

using namespace std;

int main(int argc, char** argv ){

    cv::Mat and_img;

    // create and display frame of size 300 for rectangle and circle
    cv::Mat rectangle(300, 300, CV_8UC1, cv::Scalar(0)); // rectangle
    cv::Mat circle(300, 300, CV_8UC1, cv::Scalar(0)); // circle
```

(continues on next page)

(continued from previous page)

```
    // draw and show rectangle
    cv::rectangle( rectangle, cv::Point(20, 20), cv::Point(280, 280), cv::Scalar( 255 ), -1);
    cv::imshow("Rectangle", rectangle);

    // draw and show circle
    cv::circle(circle, cv::Point(150, 150), 150, cv::Scalar(255), -1); // black
    cv::imshow("Circle", circle);

    // bitwise and operation
    cv::bitwise_and(circle, rectangle, and_img);
    cv::imshow("And", and_img);



    // another example
    cv::Mat rect1 = cv::Mat::zeros( cv::Size(400,200), CV_8UC1);
    cv::Mat rect2 = cv::Mat::zeros( cv::Size(400,200), CV_8UC1);


    rect1( cv::Range(0, 200), cv::Range(0, 200) ) = 255;
    cv::imshow("rect1", rect1);
    cv::imwrite("/home/root/rect1.jpg", rect1);
    cout << "Image written at /home/root/rect1.jpg " << endl;

    rect2( cv::Range(100, 150), cv::Range(150, 250) ) = 255;
    cv::imwrite("/home/root/rect2.jpg", rect2);
    cout << "Image written at /home/root/rect2.jpg " << endl;
    cv::imshow("rect2", rect2);

    cv::Mat result;

    bitwise_and(rect1, rect2, result);
    cv::imwrite("/home/root/result_and.jpg", result);
    cv::imshow("AND", result);

    bitwise_or(rect1, rect2, result);
    cv::imwrite("/home/root/result_or.jpg", result);
    cv::imshow("OR", result);

    bitwise_xor(rect1, rect2, result);
    cv::imwrite("/home/root/result_xor.jpg", result);
    cv::imshow("XOR", result);

    bitwise_not(rect2, result);
    cv::imwrite("/home/root/not_rect2.jpg", result);
    cv::imshow("rect2 NOT", result);

    cout << "before waitkey" << endl;
    cv::waitKey(0);

    return 0;
}
```

- Build the project,

```
$  petalinux-build -c ocvtest
$  petalinux-build
```

3.1.3 Run the desing on FPGA

- copy image.up and BOOT.bin file in SD card; and boot the FPGA.
- In the below commands, 190.122.11.229 is IP address of FPGA.

```
(check the IP address of FPGA)
$ ifconfig


(below will connet to FPGA, 190.122.11.229 will be shown by above command)
(run from windows machine)
ssh root@190.122.11.155

(run on fpga i.e. after running the above command)
root@xilinx-zcu104-2018_2:~# mount /dev/mmcblk0p1 /mnt/
root@xilinx-zcu104-2018_2:~# ocvtest
    Image written at /home/root/rect1.jpg
    Image written at /home/root/rect2.jpg
    before waitkey

root@xilinx-zcu104-2018_2:~# cp *.jpg /mnt/outimg/
root@xilinx-zcu104-2018_2:~# ls /mnt/outimg/
    not_rect2.jpg   rect2.jpg       result_or.jpg
    rect1.jpg       result_and.jpg  result_xor.jpg



(run windows machine i.e. copy the images from the folder-outimg to local machine)
(./meher is the location in windows harddisk)


(windows terminal)
cd C:
scp root@190.122.11.155:/mnt/outimg/*.jpg ./meher/outputs
```

- Output figure is shown Fig. 3.1,

3.1.4 Drawing

- In the above code, we did not read the image in the C++ code (we created the square and rectangle using commands only).
- We need to modify the code slightly to read the images from the SD card. In the other word, we need to provide the location of the image manually which requires the 'argv' in main function. Note that, the same code can be used with OpenCV C++ as well.
- In the below code, ocvtest.cpp is modified for drawing images. The C++ code 'drawing_img.cpp' is slightly modified to read the images (see highlighted section); and rest of the code is same.

```
// ocvtest.cpp

#include <stdio.h>
#include <opencv2/opencv.hpp>

using namespace std;

int main(int argc, char **argv)
{
    if( argc != 2)
    {
        cout <<" Usage: ./ocvtest <image-name>.jpg" << std::endl;
        return -1;
```

(continues on next page)

Fig. 3.1: Bitwise operation

(continued from previous page)

```
    }

    // read image
    cv::Mat img = cv::imread(argv[1]);

    // Check for invalid input
    if( img.empty() )
    {
        cout <<  "Could not open or find the image" << std::endl ;
        return -1;
    }

    // img = cv::imread("images/shapes.jpg");
    cv::imshow("Shapes", img);

    int w, h; // width, height
    // width and height of image
    w = img.size().width;
    h = img.size().height;

    // draw horizontal line
    cv::line( img, cv::Point( 0, (int)h/2 ), cv::Point( w, (int)h/2), cv::Scalar( 255, 0, 0 ), 3);
    // draw vertical line
    cv::line( img, cv::Point( (int)w/2, 0 ), cv::Point( (int)w/2, h), cv::Scalar( 255, 0, 0 ), 3);

    // draw rectangle
    cv::rectangle( img, cv::Point(5, 10), cv::Point(200, 170), cv::Scalar( 0, 0, 255 ), 3);

```

(continues on next page)

(continued from previous page)

```
    // draw circle
    // center coordinates (w//2, h//2) and radius (50) are
    // required to to draw circle. 10 is the line width
    cv::circle(img, cv::Point((int)w/2, (int)h/2), 50, cv::Scalar(0, 0, 0), 10); // black
    cv::circle(img, cv::Point((int)w/2, (int)h/2), 30, cv::Scalar(0, 0, 255), -1); // -1 : filled circle

    cv::imshow("Shapes", img);
    cv::imwrite("/home/root/drawing_img.jpg", img);
    cv::waitKey(0);
    return 0;
}
```

- Build the project,

```
$  petalinux-build -c ocvtest
$  petalinux-build
```

- Copy the image.ub and BOOT.bin file in SD-card and run the design on FPGA

```
ssh root@190.122.11.155
mount /dev/mmcblk0p1 /mnt/
ocvtest
    Usage: ./ocvtest <image-name>.jpg
ocvtest /mnt/images/shapes.jpg

cp /home/root/drawing_img.jpg /mnt/outimg/

(windows terminal)
cd C:
scp root@190.122.11.155:/mnt/outimg/*.jpg ./meher/outputs
```

- Output figure is shown Fig. 3.2,

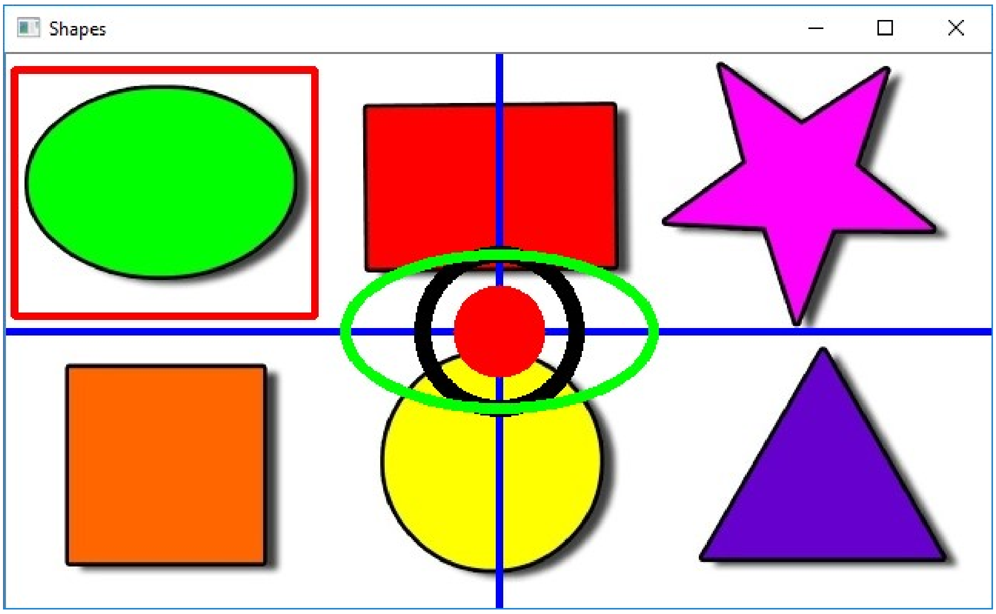

Fig. 3.2: Drawing

3.1.5 Sobel and Canny Edge detection

In the below code, Sobel edge detection and Canny edge detection algorithm are implemented.

```
// ocvtest.cpp


#include <stdio.h>
#include <opencv2/opencv.hpp>

using namespace std;

int main(int argc, char **argv)
{
    if( argc != 2)
    {
        cout <<" Usage: ./rotateimage <image-name>.jpg" << std::endl;
        return -1;
    }

    // read image
    cv::Mat img = cv::imread(argv[1]);

    // Check for invalid input
    if( img.empty() )
    {
        cout <<  "Could not open or find the image" << std::endl ;
        return -1;
    }

    // img = cv::imread("images/shapes.jpg");

    cv::imshow("Lego color", img);

    int w, h;
    // width and height of image
    w = img.size().width;
    h = img.size().height;

    cv::Mat gray_img;
    cv::cvtColor(img, gray_img, cv::COLOR_BGR2GRAY);
    cv::imshow("Lego ", gray_img);


    // sobel edge detection
    cv::Mat gX, gY;
    //  compute gradients along the X and Y axis, respectively
    cv::Sobel(gray_img, gX, CV_64F, 1, 0);
    cv::Sobel(gray_img, gY, CV_64F, 0, 1);
    // gX value after sobel conversion -52.0
    cout << "gX value after sobel conversion: " << (int)gX.at<double>(100, 200) << endl;


    // gX and gY are decimal number with +/- values
    // change these values to +ve integer format
    cv::convertScaleAbs(gX, gX);
    // gX value after Absolute scaling 52
    cv::convertScaleAbs(gY, gY);
    cout << "gX value after Absolute scaling: " << (int)gX.at<uchar>(100, 200) << endl;


    cv::Mat sobelCombined;
    cv::addWeighted(gX, 0.5, gY, 0.5, 0, sobelCombined);

    // show the output images
    cv::imshow("Sobel X", gX);
```

(continues on next page)

(continued from previous page)

```
    cv::imwrite("/home/root/Sobel X.jpg", gX);
    cv::imshow("Sobel Y", gY);
    cv::imwrite("/home/root/Sobel Y.jpg", gY);
    cv::imshow("Sobel Combined", sobelCombined);
    cv::imwrite("/home/root/Sobel Combined.jpg", sobelCombined);


    // Canny edge detection
    cv::Mat canny_wide, canny_medium, canny_narrow;

    cv::Canny(gray_img, canny_wide, 10, 200);
    cv::Canny(gray_img, canny_medium, 50, 150);
    cv::Canny(gray_img, canny_narrow, 200, 250);

    // show the output images
    cv::imshow("Canny (10, 200)", canny_wide);
    cv::imwrite("/home/root/canny_wide.jpg", canny_wide);
    cv::imshow("Canny (50, 150)", canny_medium);
    cv::imwrite("/home/root/canny_medium.jpg", canny_medium);
    cv::imshow("Canny (200, 250)", canny_narrow);
    cv::imwrite("/home/root/canny_narrow.jpg", canny_narrow);

    cv::waitKey(0);
    return 0;
}
```

- Build the project,

```
$  petalinux-build -c ocvtest
$  petalinux-build
```

- Copy the image.ub and BOOT.bin file in SD-card and run the design on FPGA

```
ssh root@190.122.11.155
mount /dev/mmcblk0p1 /mnt/
ocvtest /mnt/images/lego.jpg

cp /home/root/*.jpg /mnt/outimg/

(windows terminal)
cd C:
scp root@190.122.11.155:/mnt/outimg/*.jpg ./meher/outputs
```

- Output figure for Sobel detection is shown Fig. 3.3,
- Output figure for Canny detection is shown Fig. 3.4,

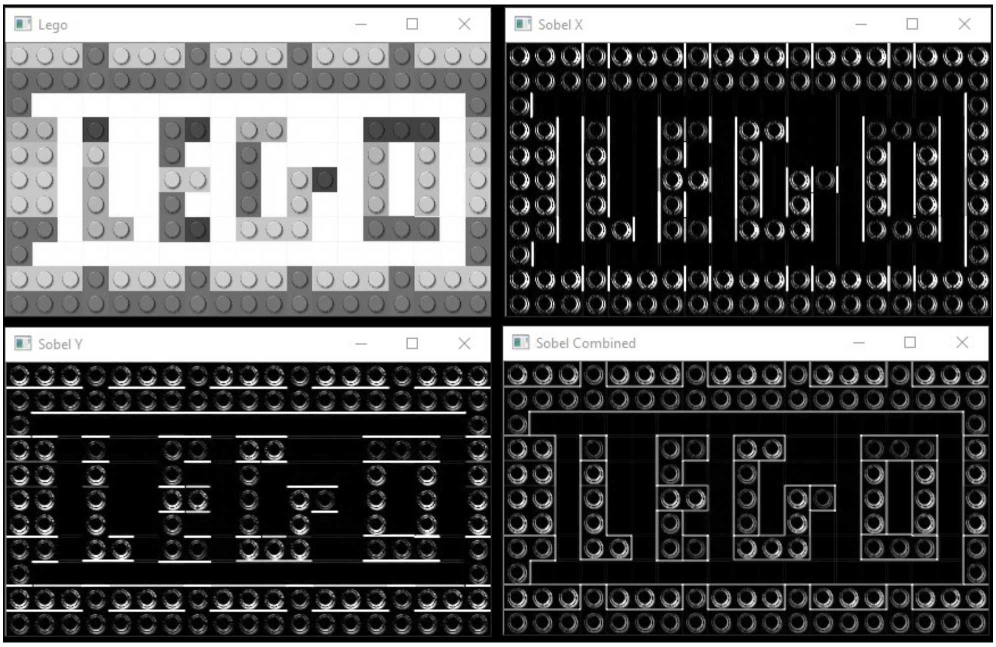

Fig. 3.3: Sobel edge detection

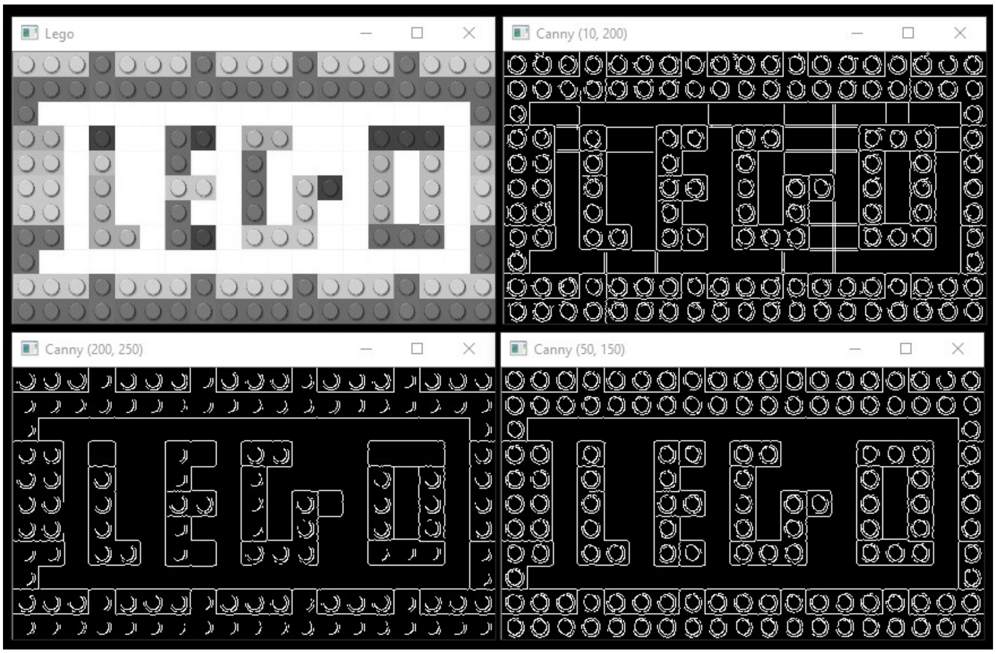

Fig. 3.4: Canny edge detection