Module 1.6

OpenCV Basics

HighGUI

Satya Mallick, Ph.D.
LearnOpenCV.com

Table of Contents

Highgui

The Highgui module was designed to test certain functionalities quickly and provide immediate results.

It provides easy interface to:

- Create and manipulate windows that can display images and “remember” their content (no need to handle repaint events from OS).
- Add trackbars to the windows, handle simple mouse events as well as keyboard commands.
- Read and write images to/from disk or memory.
- Read video from camera or file and write video to a file.

In this chapter, We will learn how to use the following highgui tools :

1. Obtain keyboard inputs while the program is running.
2. Obtain mouse inputs and perform some simple application like cropping an image.
3. Use of Trackbars for changing variable values at runtime. We will revisit the thresholding operation to illustrate this.
4. We will also see how to take inputs as arguments from the command line while executing the program.

The names of the tools are self-explanatory. So, let us jump right into the code.

How to use the Keyboard in OpenCV

Getting the input from the keyboard is done using the `waitKey()` function.

The code given below opens the webcam and displays text when ‘e/E’ or ‘z/Z’ is pressed. On pressing ‘ESC’ the program terminates and the display window closes. Note the use of **`waitKey`** here and how this time `waitKey(0)` has not been used rather there is some finite delay(10 s). This delay helps to see the text better else the text would disappear as soon as it got displayed.

Code highguiKeyboard (C++)

Let’s look at the code. Remember, the code below is a tutorial with important information highlighted using green boxes. The C++ tutorial is shown below.

C++ [HighGui - Keyboard] [highguiKeyboard.cpp]

```cpp
#include <opencv2/videoio.hpp>
#include <opencv2/core.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>
#include <string>
#include <iostream>

using namespace std;
using namespace cv;

```

The code displays the usage of the highgui header file. This program reads keyboard inputs and display it on image. Specifically, it checks for ‘e’ or ‘z’ keys.

```cpp
int main(void)
{

 // Open webcam
 VideoCapture cap(0);
 Mat frame;
 int k=0;
 // Detect if webcam is working properly
 if(!cap.isOpened())
 {
   cout<<"Unable to detect webcam "<<"\n";
   return 0;
 }
 else
 {
```

We use the waitKey() function to detect the input and respond only if either ‘e’ or ‘z’ is pressed. ‘ESC’(ASCII code = 27) is used to exit the program.

```cpp
   while(1)
   {
     // Capture frame
     cap>>frame;
```

The following if-else block is used to check which key is pressed

```cpp
     if(k==27)
     break;
     // Identify if 'e' or 'E' is pressed
     if(k==101 || k==69)
     putText(frame, "E is pressed, press Z or ESC", Point(100,180), FONT_HERSHEY_SIMPLEX, 1,
Scalar(0,255,0), 3);
     // Identify if 'z' or 'Z' is pressed or not
     if(k==90 || k==122)
     putText(frame, "Z is pressed", Point(100,180), FONT_HERSHEY_SIMPLEX, 1, Scalar(0,255,0), 3);
     imshow("Image",frame);
     // Waitkey is increased so that the display is shown
     k= waitKey(10000) & 0xFF;
```

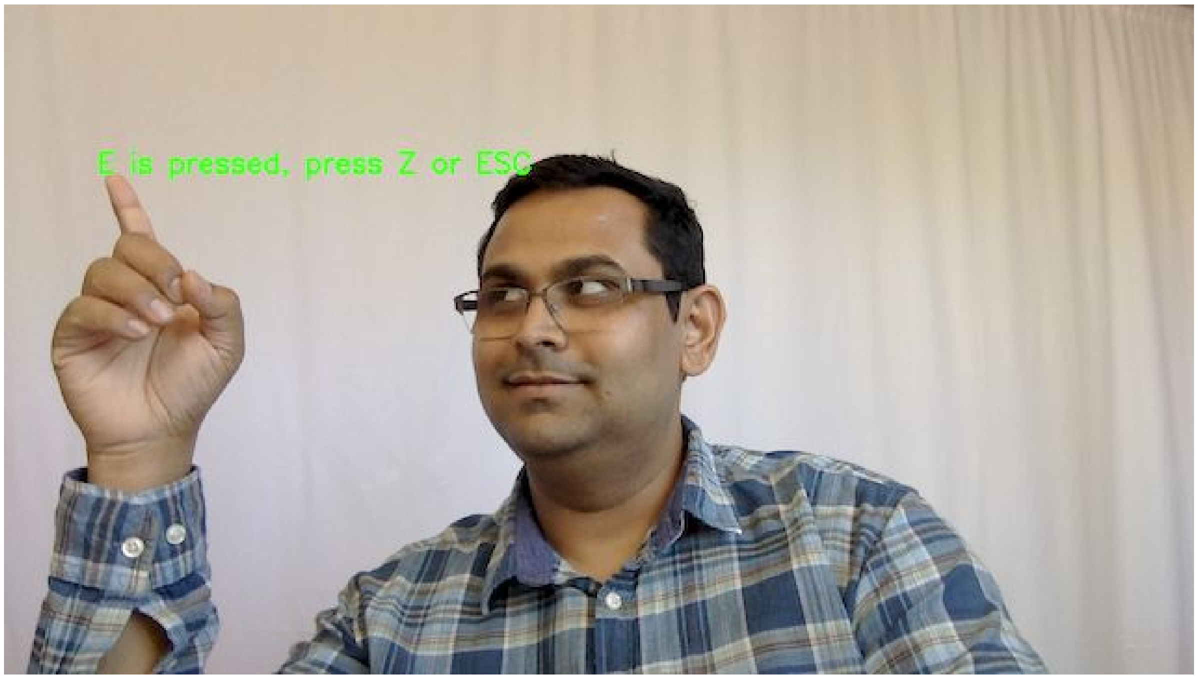


Figure showing the output of the above program

```
    }
  }

}
```

Code highguiKeyboard (Python)

Python [Highgui - Keyboard] [highguiKeyboard.py]

```
import cv2

```

This program shows how highgui enables us to read keyboard inputs and display it on image.

```
# Open webcam
cap = cv2.VideoCapture(0)
k=0
```

We use the waitKey() function to detect the input and respond only if either ‘e’ or ‘z’ is pressed. ‘ESC’(ASCII code = 27) is used to exit the program.

```
while(True):
 # Read frame
 ret,frame = cap.read()
```

The following if-else block is used to check which key is pressed

```
# Identify if 'ESC' is pressed or not
if(k==27):
      break
# Identify if 'e' or 'E' is pressed
if(k==101 or k==69):
      cv2.putText(frame, "E is pressed, press Z or ESC", (100,180), cv2.FONT_HERSHEY_SIMPLEX, 1,
(0,255,0), 3);
# Identify if 'z' or 'Z' is pressed
if(k==90 or k==122):
      cv2.putText(frame, "Z is pressed",(100,180), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255,0), 3)

cv2.imshow("Image",frame)
```

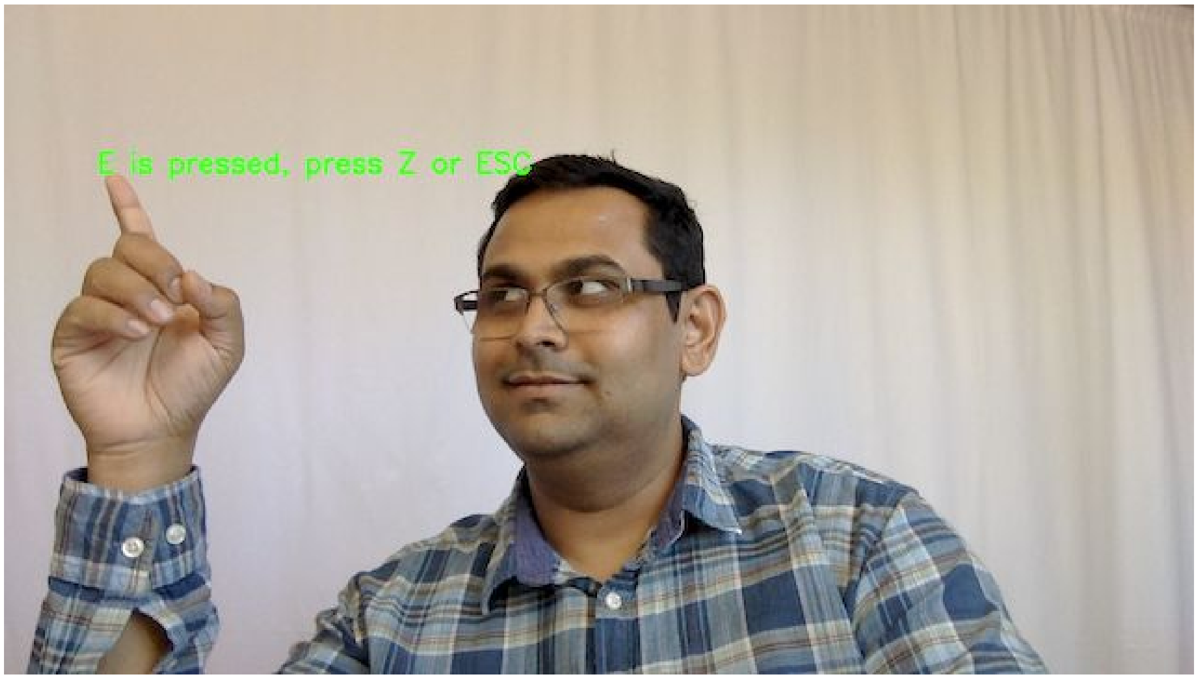


Figure showing the output of the above program

```
# Increase waitkey to show display properly
k= cv2.waitKey(10000) & 0xFF
```

How to use the Mouse in OpenCV

We can detect mouse events like left-click, right-click or position of the mouse on the window using OpenCV. For doing that, we need to create a named window and assign a callback function to the window. We will see how it is done in the code.

The code given below draws a circle on the image. You first mark the center of the circle and then drag the mouse according to the radius desired. Multiple circles can be drawn. ‘c’ is used to clear the screen (the circles) and pressing “ESC’ terminates the program.

Code highguiMouse (C++)

C++ [HighGui - Mouse] [highguiMouse.cpp]

```cpp
#include <opencv2/core.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>
#include <vector>
#include <iostream>
#include <math.h>

using namespace cv;
using namespace std;

```

This program shows how highgui enables us to take mouse inputs. In this code we use mouse input to draw a circle on an image. The mouse is dragged from the center to one of the points on the circumference. ‘c’ can be pressed to remove the drawn circles.

```cpp
// Points to store the center of the circle and a point on the circumference
Point center, circumference;
// Source image
Mat source;

```

drawCircle the callback function is called when there is a mouse event like left click (indicated by EVENT_LBUTTONDOWN). The coordinates relative to the namedWindow is captured by this function in the variables (x,y). The function records the points of the circle’s center and a point on the circumference, hence allowing us to draw the desired circle on the image.

```cpp
// function which will be called on mouse input
void drawCircle(int action, int x, int y, int flags, void *userdata)
{
 // Action to be taken when left mouse button is pressed
 if( action == EVENT_LBUTTONDOWN )
 {
       center = Point(x,y);
       // Mark the center
       circle(source, center, 1, Scalar(255,255,0), 2, CV_AA );
 }
 // Action to be taken when left mouse button is released
```

```
  else if( action == EVENT_LBUTTONUP)
  {
        circumference = Point(x,y);
        // Calculate radius of the circle
        float radius = sqrt(pow(center.x-circumference.x,2)+pow(center.y-circumference.y,2));
        // Draw the circle
        circle(source, center, radius, Scalar(0,255,0), 2, CV_AA );
        imshow("Window", source);
  }

}
int main()
{
 source = imread("../../data/images/source.jpg",1);
 // Make a dummy image, will be useful to clear the drawing
 Mat dummy = source.clone();
 namedWindow("Window");
 // highgui function called when mouse events occur
 setMouseCallback("Window", drawCircle);
 int k=0;
 // loop until escape character is pressed
 while(k!=27)
 {
        imshow("Window", source );
        putText(source,"Choose center, and drag, Press ESC to exit and c to clear" ,Point(10,30),
FONT_HERSHEY_SIMPLEX, 0.7,Scalar(255,255,255), 2 );
        k= waitKey(20) & 0xFF;
        if(k== 99)
                // Another way of cloning
                dummy.copyTo(source);
 }
 return 0;
}
```

Code highguiMouse (Python)

Python [Highgui - Mouse] [highguiMouse.py]

```
import cv2
import math
```

This program shows how highgui enables us to take mouse inputs. In this code we use mouse input to draw a circle on an image. The mouse is dragged from the center to one of the points on the circumference. ‘c’ can be pressed to remove the drawn circles.

```

# Lists to store the points
center=[]
circumference=[]
```

drawCircle the callback function is called when there is a mouse event like left click (indicated by EVENT_LBUTTONDOWN). The coordinates relative to the namedWindow is captured by this function in the variables (x,y). The function records the points of the circle’s center and a point on the circumference, hence allowing us to draw the desired circle on the image.

```python
def drawCircle(action, x, y, flags, userdata):
 # Referencing global variables
 global center, circumference
 # Action to be taken when left mouse button is pressed
 if action==cv2.EVENT_LBUTTONDOWN:
       center=[(x,y)]
       # Mark the center
       cv2.circle(source, center[0], 1, (255,255,0), 2, cv2.LINE_AA );

 # Action to be taken when left mouse button is released
 elif action==cv2.EVENT_LBUTTONUP:
       circumference=[(x,y)]
       # Calculate radius of the circle
       radius = math.sqrt(math.pow(center[0][0]-circumference[0][0],2)+math.pow(center[0][1]-circumference[0][1],2))
       # Draw the circle
       cv2.circle(source, center[0], int(radius), (0,255,0),2, cv2.LINE_AA)
       cv2.imshow("Window",source)

source = cv2.imread("../../data/images/source.jpg",1)
# Make a dummy image, will be useful to clear the drawing
dummy = source.copy()
cv2.namedWindow("Window")
# highgui function called when mouse events occur
cv2.setMouseCallback("Window", drawCircle)
k = 0
# loop until escape character is pressed
while k!=27 :

 cv2.imshow("Window", source)
 cv2.putText(source,"Choose center, and drag, Press ESC to exit and c to clear" ,(10,30), cv2.FONT_HERSHEY_SIMPLEX, 0.7,(255,255,255), 2 );
 k = cv2.waitKey(20)& 0xFF
 # Another way of cloning
 if k==99:
       source= dummy.copy()


cv2.destroyAllWindows()
```

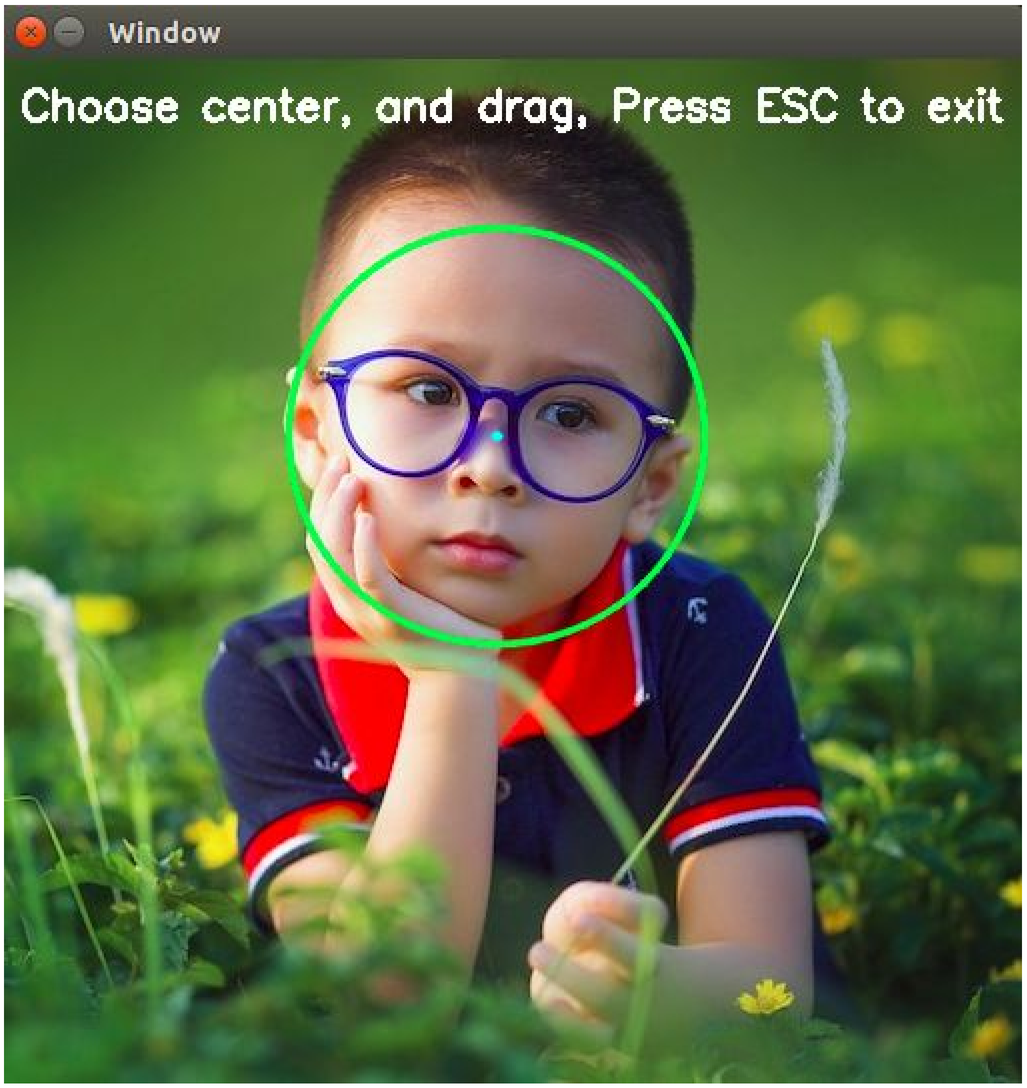


Figure showing the output of the above program

How to use Trackbars in OpenCV

In this section, we will see how trackbars can be used in OpenCV. We will use the thresholding operation to illustrate the usage of trackbars. Please refer to the different types of thresholding operations available in OpenCV here.

For creating trackbars, we have to specify a named window and use the createTrackbar() function in which we need to specify the window name. A callback function needs to be specified for detecting events on the trackbar. Let's see an example code.

Code Trackbar (C++)

[C++] [trackbar.cpp]

```cpp
#include <string>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>

using namespace std;
using namespace cv;

```

This program shows how highgui enables us to dynamically vary variables using trackbars and record the change to produce various results. In this example we use trackbars to threshold images. There are two trackbars which are used, one controls the threshold value while the other controls the threshold type.

```cpp
int threshold_value = 150;
int threshold_type = 3;;
int const max_value = 255;
int const max_type = 4;
int const max_BINARY_value = 255;

Mat im, imGray, dst;

string windowName = "Threshold Demo";
string trackbarType = "Type: \n 0: Binary \n 1: Binary Inverted \n 2: Truncate \n 3: To Zero \n 4: To Zero Inverted";
string trackbarValue = "Value";

void thresholdDemo(int, void*);

int main() {
   // Load an image
   im = imread("../../data/images/theshold.png", IMREAD_COLOR);

   // Convert the image to Gray
   cvtColor(im, imGray, COLOR_RGB2GRAY);

   // Create a window to display results
   namedWindow(windowName, CV_WINDOW_AUTOSIZE);

```

The trackbars are created here using the createTrackbar function. The different parameters of the function are given below.

createTrackbar(const String& trackbarname, const String& winname, int* value, int count, TrackbarCallback onChange=0, void* userdata=0)

Trackbarname is the name that will be displayed alongside the trackbar, winname is the namedWindow associated with the callback function, onChange is the callback function which is associated with the winname window and gets triggered when the trackbar is accessed by the user. Value is a pointer to an integer variable whose value indicates the position of the trackbar. Count is the maximum position of the trackbar, minimum being 0 always. And userdata can be passed to the callback function for controlling the events instead of using global variables.

```
    /// Create Trackbar to choose type of Threshold
    createTrackbar(trackbarType, windowName, &threshold_type, max_type, thresholdDemo);

    createTrackbar(trackbarValue, windowName, &threshold_value, max_value, thresholdDemo);

    /// Call the function to initialize
    thresholdDemo(0, 0);

    /// Wait until user finishes program
    while (true) {
        int c;
        c = waitKey(20);
        if (static_cast<char>(c) == 27)
            { break; }
     }
}


```

This is the callback function which is called every time the trackbar is used. Note: Both trackbars call the same function.

```
void thresholdDemo(int, void*) {
    /*
    0: Binary
    1: Binary Inverted
    2: Threshold Truncated
    3: Threshold to Zero
    4: Threshold to Zero Inverted
    */

    threshold(imGray, dst, threshold_value, max_BINARY_value, threshold_type);

    imshow(windowName, dst);
}
```

Code Trackbar (Python)

[Python] [trackbar.py]

```
import cv2
```

This program shows how highgui enables us to dynamically vary variables using trackbars and record the change to produce various results. In this we use trackbars to threshold images. There are two trackbars which are used, one controls the threshold value while the other controls the threshold type.

```python

thresholdValue = 200
thresholdType = 3
maxValue = 255
maxType = 4
max_BINARY_value = 255

windowName = "Threshold Demo"
trackbarType = "Type: \n 0: Binary \n 1: Binary Inverted \n 2: Truncate \n 3: To Zero \n 4: To Zero Inverted"
trackbarValue = "Value"
# Call the function to initialize
# 0: Binary
# 1: Binary Inverted
# 2: Threshold Truncated
# 3: Threshold to Zero
# 4: Threshold to Zero Inverted

# Load an image
imGray = cv2.imread("../../data/images/threshold.png", cv2.IMREAD_GRAYSCALE)

# # Convert the image to Gray
# imGray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

# Create a window to display results
cv2.namedWindow(windowName, cv2.WINDOW_AUTOSIZE)


```

These are the callback functions which are called by the respective trackbars.

```python
def thresholdTypeDemo(*args):
   global thresholdType
   thresholdType = args[0]
   thresh, result = cv2.threshold(imGray, thresholdValue, max_BINARY_value, thresholdType)
   cv2.imshow(windowName, result)


def thresholdValueDemo(*args):
   global thresholdValue
   thresholdValue = args[0]
   thresh, result = cv2.threshold(imGray, thresholdValue, max_BINARY_value, thresholdType)
   cv2.imshow(windowName, result)

```

The trackbars are created here using the createTrackbar function. The different parameters of the function are given below.

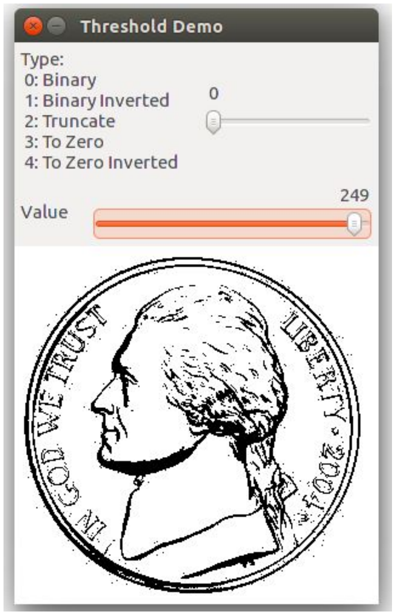
cv2.createTrackbar(trackbarName, windowName, value, count, onChange)

Trackbarname is the name that will be displayed alongside the trackbar, windowName is the namedWindow associated with the callback function, onChange is the callback function which is associated with the winname window and gets triggered when the trackbar is accessed by the user. Value is a pointer to an integer variable whose value indicates the position of the trackbar. Count is the maximum position of the trackbar, minimum being 0 always.

```python
# Create Trackbar to choose type of Threshold
cv2.createTrackbar(trackbarType, windowName, thresholdType, maxType, thresholdTypeDemo)

cv2.createTrackbar(trackbarValue, windowName, thresholdValue, maxValue, thresholdValueDemo)

thresholdValueDemo(0)

# Wait until user finishes program
while True:
    c = cv2.waitKey(20)
    if c == 27:
        break
```

Sample Outputs

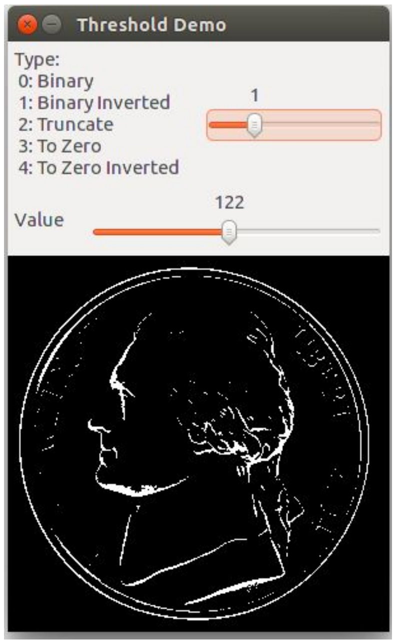
A few sample outputs are shown below.


Threshold Demo

Type:
0: Binary
1: Binary Inverted
2: Truncate
3: To Zero
4: To Zero Inverted

0

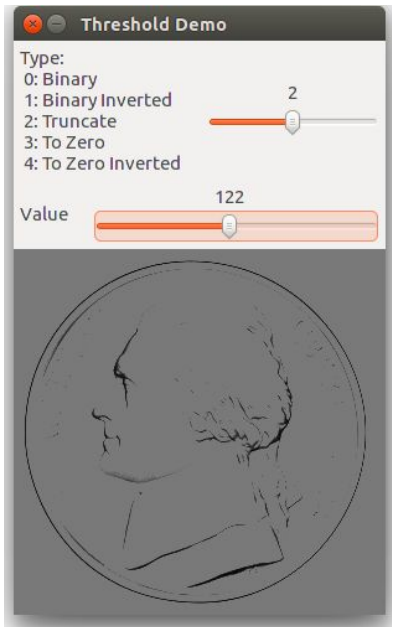
Value 249


Threshold Demo

Type:
0: Binary
1: Binary Inverted
2: Truncate
3: To Zero
4: To Zero Inverted

1

Value 122


Threshold Demo

Type:
0: Binary
1: Binary Inverted
2: Truncate
3: To Zero
4: To Zero Inverted

2

Value 122

References and Further Readings

http://docs.opencv.org/3.0-beta/doc/tutorials/highgui/trackbar/trackbar.html

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

http://opencv-srf.blogspot.in/2011/11/mouse-events.html

http://docs.opencv.org/3.1.0/db/d5b/tutorial_py_mouse_handling.html

http://www.pyimagesearch.com/2015/03/09/capturing-mouse-click-events-with-python-and-opencv/