Module 1.4.2

OpenCV Basics

Binary Image Processing

Part 2

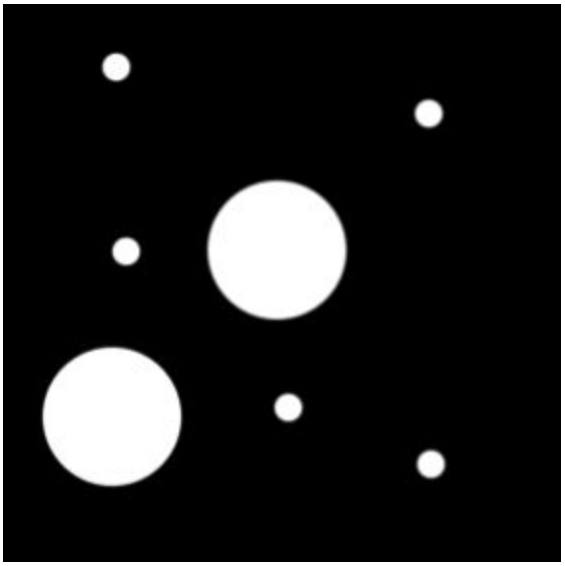
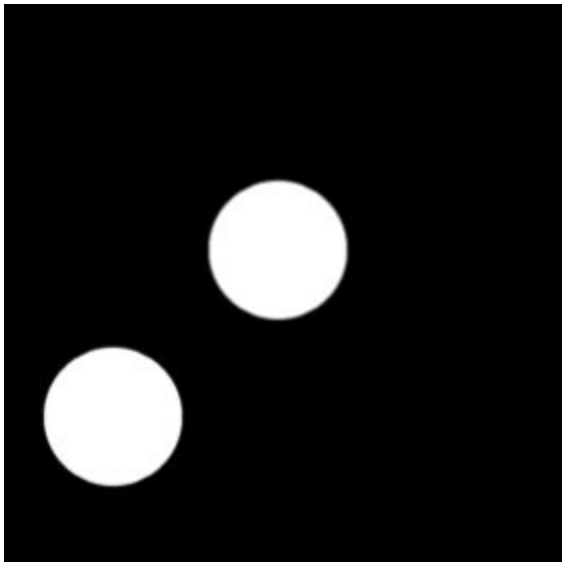
Satya Mallick, Ph.D.
LearnOpenCV.com

Table of Contents

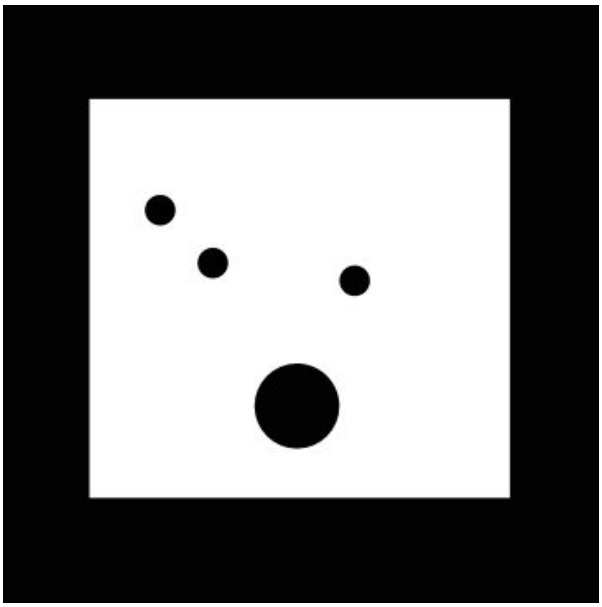
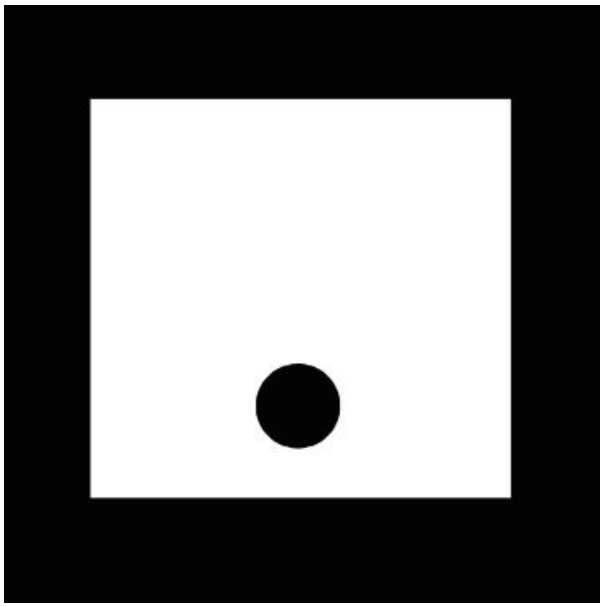
Binary Image Processing, Part 2

Opening and Closing

Dilation and Erosion operations are often combined to achieve interesting results. For example, small blobs can be removed by performing erosion followed by dilation. This operation is also called **morphological opening.**

	
Original Image	**After Opening Operation**

On the other hand, small holes inside blobs can be filled by performing dilation followed by erosion. This operation is called **morphological closing.**

	
Original Image	**After Closing Operation**

Opening and Closing using OpenCV

In OpenCV, the opening and closing operations are implemented using **MorphologyEx** function.
To specify between the opening and closing operation to be performed we specify an argument in the function **MorphologyEx** definition. The argument for opening operation and closing operations are MORPH_OPEN and MORPH_CLOSE respectively.

Opening:

```
C++: void morphologyEx(Mat initialImage, Mat imageMorphOpened, MORPH_OPEN,
Mat structuringElement)

Python: imageMorphOpened = cv2.morphologyEx(initialImage, cv.MORPH_OPEN,
structuringElement)
```

Closing:

```
C++: void morphologyEx(Mat initialImage, Mat imageMorphClosed, MORPH_CLOSE,
Mat structuringElement)

Python: imageMorphClosed = cv2.morphologyEx(initialImage, cv.MORPH_CLOSE,
structuringElement)
```

Note: In the functions above, the parameter 'iterations' is optional and if not mentioned the default value is taken to be 1. In case, we need to run the opening/closing function more than once, we specify the number of iterations in the **`MorphologyEx`** function call statement. In the first sample code provided below, i.e., the desired opening operation requires the number of iterations to be 3. In Python, the optional parameters are passed by name "iterations=3" (Line 14 of the python version of the Morphological Opening code).

On the other hand, in C++ to use an optional parameter, we need to specify the values for previous optional parameters as well. Lines 14 and 23 of the C++ version of the morphological opening shows the usage. We need to declare the variable "iterations" and initialize with the number of iterations we require, before calling the **`MorphologyEx`** function.

From line 23 of the code, it is clear that we have to pass two extra parameters. One is the anchor point Point(-1,-1). The anchor point Point(-1,-1) is taken as default when not mentioned. But we need to specify explicitly, the position of the anchor within the element, while passing the number of iterations as the second parameter. In the second sample code, i.e., the code for closing operation the number of iteration is 1. So, the standard function call is used.

The following C++ and Python codes provide the implementation of Opening and Closing Operation.

Code Opening (C++)

C++ [Morphological Opening] [opening.cpp]

```
1 // Import OpenCV
2 #include <iostream>
3 #include <opencv2/highgui.hpp>
4 #include <opencv2/imgproc.hpp>
5
6 using namespace cv;
7 using namespace std;
8
9 int main() {
10   Mat image;
```

In this program, we read an image from the images directory, and store it in 'Mat image',then perform Morphological Opening operation on the image. Finally the Morphological Opened image is stored in 'Mat imageMorphOpened'. At the end, both the images are displayed.

```
11   // Image read in and stored in Mat image
12   image = imread("../../data/images/opening.png", IMREAD_GRAYSCALE);
```

In some cases, it may be required to implement a particular morphological operation more than once. In those cases, we need to define the number of iterations required.

```
13   // Initializing number of iterations required
14   int iterations = 3;
```

We create the kernel/structuring element that is used for opening operation.

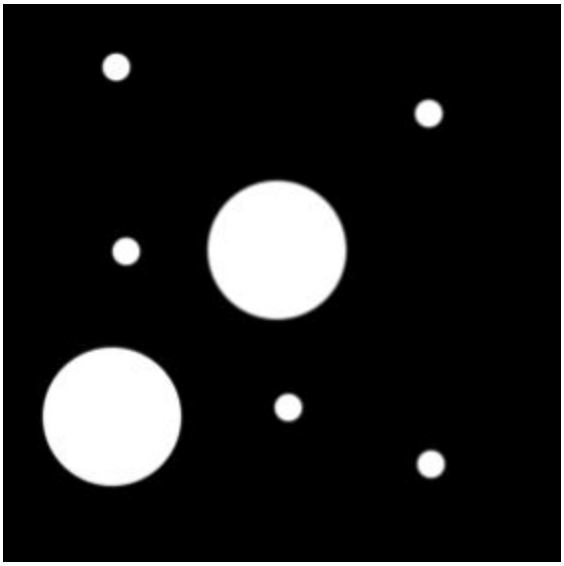
```
15   // Create a structuring element
16   int openingSize = 3;
17   // Selecting a elliptical kernel
18   Mat element = getStructuringElement(cv::MORPH_ELLIPSE,cv::Size(2 * openingSize + 1,
                                2 * openingSize + 1),cv::Point(openingSize, openingSize));
19   // Mat imageMorphOpened stores the image after opening operation
20   Mat imageMorphOpened;
```

Opening operation performed on input image and stored in 'Mat imageMorphOpened'.

```
21   // Applying erosion followed by dilation on image by using morphologyEx function and specifying
MORPH_OPEN tag(denotes opening operation)
22   morphologyEx(image, imageMorphOpened, MORPH_OPEN, element, Point(-1,-1),iterations);
```

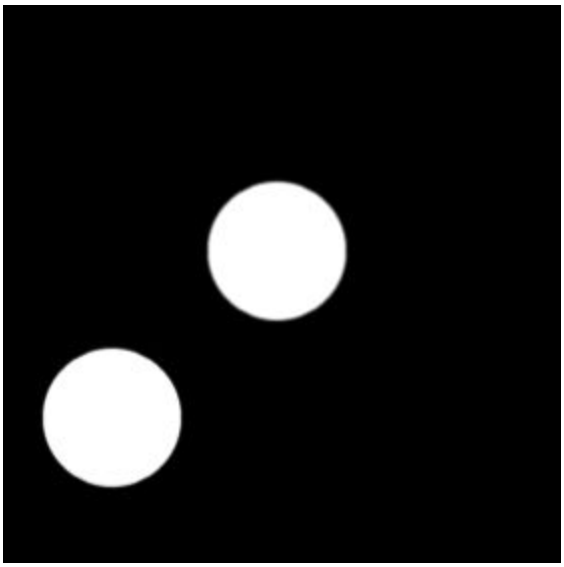
Finally, we display both the images on the screen.

```
23   // Display original image
24   namedWindow("Original Image", CV_WINDOW_AUTOSIZE);
25   imshow("Original Image", image);
```



Original Image

```
26   // Display the image after morphological opening is done
27   namedWindow("Opening", CV_WINDOW_AUTOSIZE);
28   imshow("Opening", imageMorphOpened);
```



Opened Image

```
29   // Wait for the user to press any key
30   waitKey(0);
31   return 0;
32}
```

Code Opening (Python)

Python[Morphological Opening] [opening.py]

```
 1 # Import OpenCV and numpy
 2 import cv2
```

In this program, we read an image from the images directory, and store it in 'NumPy array image',then perform Morphological Opening operation on the image. Finally the Morphological Opened image is stored in 'NumPy array imageMorphOpened'. At the end, both the images are displayed.

```
 3 # Image taken as input
 4 imageName = "../../data/images/opening.png"
 5 image = cv2.imread(imageName, cv2.IMREAD_GRAYSCALE)
 6 # Check for invalid input
 7 if image is None:
 8   print("Could not open or find the image")
```

We create the kernel/structuring element that is used for opening operation.

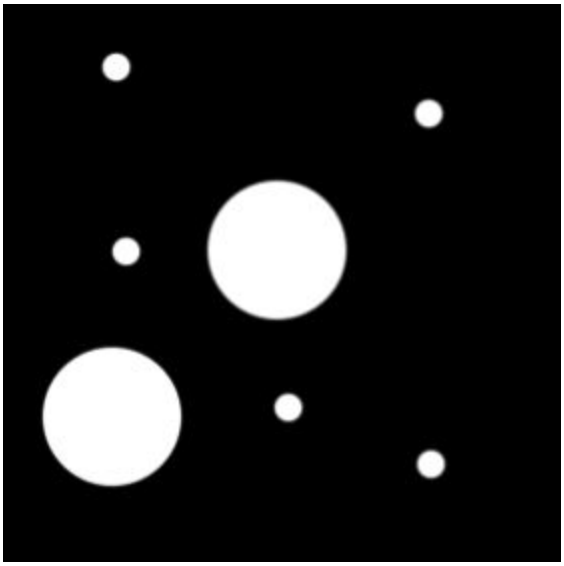
```
 9 # Get structuring element/kernel which will be used for opening operation
10 openingSize = 3
11 # Selecting a elliptical kernel
12 element = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,
                      (2 * openingSize + 1, 2 * openingSize + 1),(openingSize,openingSize))
```

Opening operation performed on input image and stored in ‘NumPy array imageMorphOpened’.In some cases, it may be required to implement a particular morphological operation more than once. In those cases, we need to define the number of iterations required.

```
13 # Applying erosion followed by dilation on image by using morphologyEx function and specifying
MORPH_OPEN tag(denotes opening operation)
14 imageMorphOpened = cv2.morphologyEx(image, cv2.MORPH_OPEN, element,iterations=3)
```

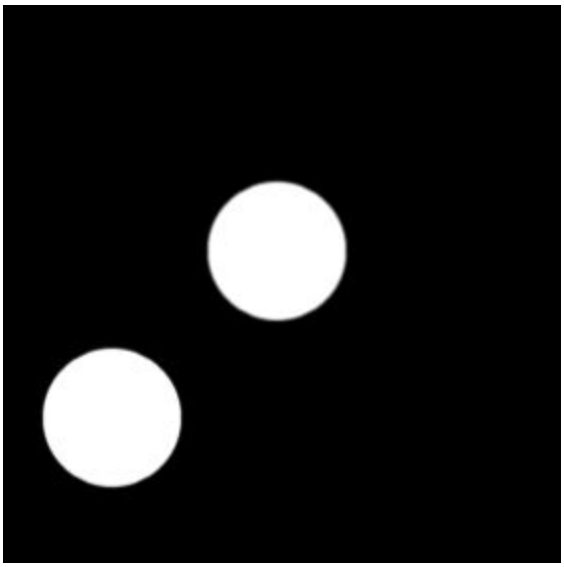
At the end we display both the images on the screen.

```
15 # Display the original image
16 cv2.namedWindow("Original Image", cv2.WINDOW_AUTOSIZE)
17 cv2.imshow("Original Image", image)
```



Original Image

```
18 # Image after morphological opening operation
19 cv2.namedWindow("Opening", cv2.WINDOW_AUTOSIZE)
20 cv2.imshow("Opening",imageMorphOpened)
```



Opened Image

```
21 # Wait for user to press any key
22 cv2.waitKey(0)
```

Code Closing (C++)

C++ [Morphological Closing] [closing.cpp]

```
 1 // Import OpenCV
 2 #include <iostream>
 3 #include <opencv2/highgui.hpp>
 4 #include <opencv2/imgproc.hpp>
 5
 6 using namespace cv;
 7 using namespace std;
 8
 9 int main() {
10   Mat image;
```

In this program, we read an image from the images directory, and store it in 'Mat image',then perform Morphological Closing operation on the image. Finally the Morphological Closed image is stored in 'Mat imageMorphClosed'. At the end, both the images are displayed.

```
11   // Image read in and stored in Mat image
12   image = imread("../../data/images/closing.png", IMREAD_GRAYSCALE);
```

We create the kernel/structuring element that is used for closing operation.

```
13   // Create a structuring element
14   int closingSize = 10;
15   // Selecting a elliptical kernel and storing in Mat element
16   Mat element = getStructuringElement(cv::MORPH_ELLIPSE,cv::Size(2 * closingSize + 1,
```

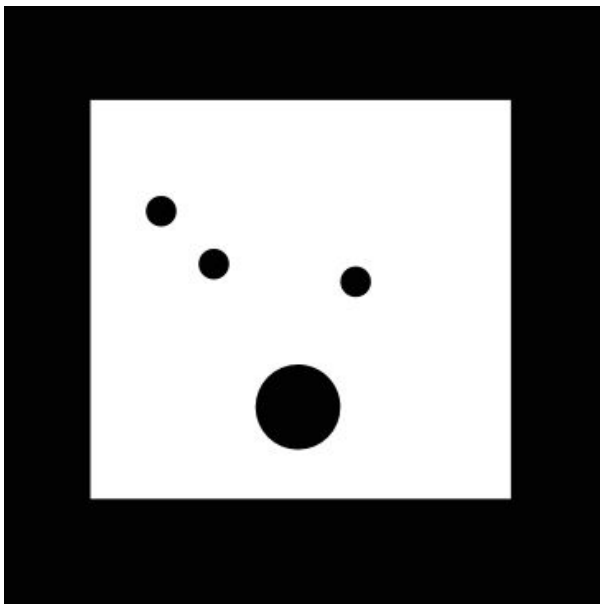
```
                            2 *  closingSize + 1),cv::Point(closingSize, closingSize));
17   // Mat imageMorphClosed stores the image after closing operation
18   Mat imageMorphClosed;
```

Closing operation performed on input image and stored in ‘Mat imageMorphClosed’.

```
19   // Applying dilation followed by closing on image by using morphologyEx function and specifying
MORPH_CLOSE tag(denotes closing operation)
20   morphologyEx(image, imageMorphClosed, MORPH_CLOSE, element);
```

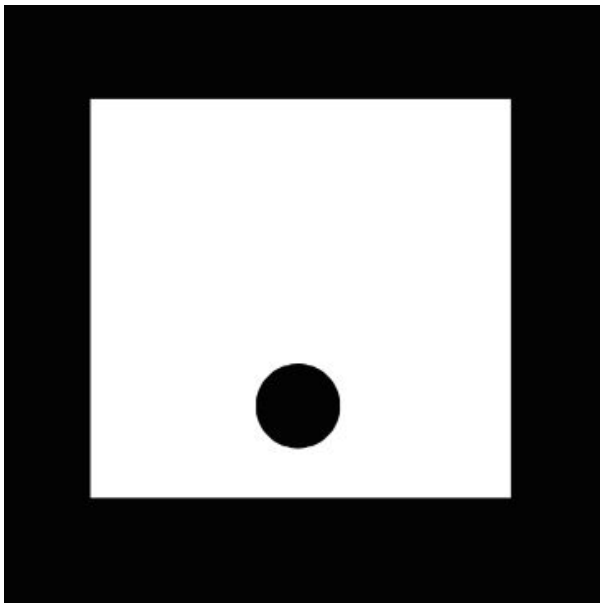
At the end we display both the images on the screen.

```
21   // Display original image
22   namedWindow("Original Image", CV_WINDOW_AUTOSIZE);
23   imshow("Original Image", image);
```



Original Image

```
24   // Display the image after morphological closing is done
25   namedWindow("Closing", CV_WINDOW_AUTOSIZE);
26   imshow("Closing", imageMorphClosed);
```



Closed Image

```
27   // Wait for the user to press any key
28   waitKey(0);
29   return 0;
30 }
```

Code Closing (Python)

Python[Morphological Closing] [closing.py]

```
1 # Import OpenCV and numpy
2 import cv2
```

In this program, we read an image from the images directory, and store it in 'NumPy array image',then perform Morphological Closing operation on the image. Finally the Morphological Closed image is stored in 'NumPy array imageMorphClosed'. At the end, both the images are displayed.

```
3 # Image taken as input
4 imageName = "../../data/images/closing.png"
5 image = cv2.imread(imageName, cv2.IMREAD_GRAYSCALE)
6 # Check for invalid input
7 if image is None:
8   print("Could not open or find the image")
```

We create the kernel/structuring element that is used for closing operation.

```
9 # Get structuring element/kernel which will be used for closing operation
10 closingSize = 10
11 # Selecting a elliptical kernel
12 element = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,
```

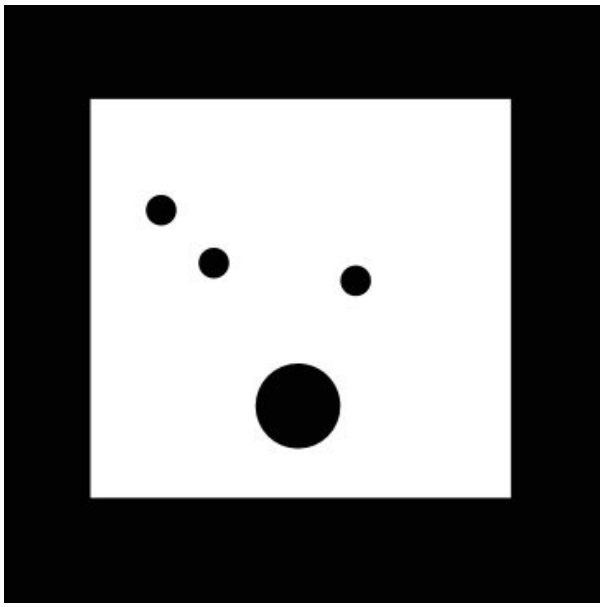
```
                    (2 * closingSize  + 1, 2 * closingSize  + 1),(closingSize ,closingSize ))
```

Closing operation performed on input image and stored in ‘NumPy array imageMorphClosed’.

```
13 # Applying dilation followed by closing on image by using morphologyEx function and specifying
MORPH_CLOSE tag(denotes closing operation)
13 imageMorphClosed = cv2.morphologyEx(image, cv2.MORPH_CLOSE, element)
```

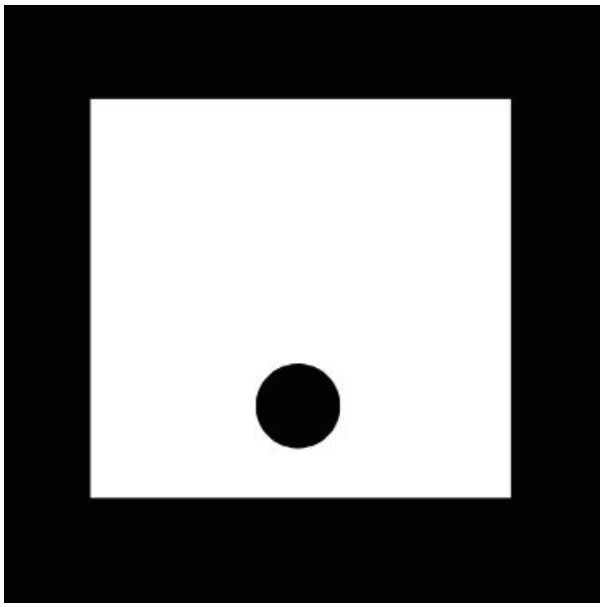
At the end we display both the images on the screen.

```
14 # Display the original image
15 cv2.namedWindow("Original Image", cv2.WINDOW_AUTOSIZE)
16 cv2.imshow("Original Image", image)
```



Original Image

```
17 # Image after morphological closing operation
18 cv2.namedWindow("Closing", cv2.WINDOW_AUTOSIZE)
19 cv2.imshow("Closing",imageMorphClosed)
```



Closed Image

```
20 # Wait for user to press any key
21 cv2.waitKey(0)
```

References and Further reading

http://docs.opencv.org/3.0-beta/doc/tutorials/imgproc/opening_closing_hats/opening_closing_hats.html

https://pythonprogramming.net/morphological-transformation-python-opencv-tutorial/