

Introduction to OpenCV

1. Introduction

OpenCV (Open Source Computer Vision Library) is an open-source BSD-licensed library that includes several hundreds of computer vision algorithms, and hence it is free for both academic and commercial use. It has C++, Python and Java interfaces and supports Windows, Linux, MacOS iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications.

OpenCV is a cross-platform library using which we can develop real-time computer vision applications. It mainly focuses on image processing, video capture and analysis including features like face detection and object detection. The features of OpenCV library are as follows.

- (a) Reading and writing images
- (b) Capturing and saving videos
- (c) Processing images (filter, transform)
- (d) Performing feature detection
- (e) Detecting specific objects such as faces, eyes, cars, in the videos or images
- (f) Analyzing the video, i.e. estimating the motion in it, subtracting the background, and tracking objects in it.

2. Mat Class

To capture an image, we use devices like cameras and scanners. These devices record numerical values of the image (Ex: pixel values). OpenCV is a library which processes the digital images, therefore we need to store these images for processing.

The **Mat** class of OpenCV library is used to store the values of an image. It represents an n-dimensional array and is used to store image data of grayscale or color images, etc.

The Mat class comprises of two data parts as follows.

- ✓ Header: contains information like size, method used for storing, and the address of the matrix (constant in size).
- ✓ Pointer; stores the pixel values of the image (keeps on varying).

The Mat class is provided within the package **org.opencv.core**. Table 1 shows some constructors of Mat class to construct the Mat object. Table 2 shows some of the methods provided by the Mat class.

Table 1. The main constructors of the Mat class.

Constructor of Mat
Mat() This is the default constructor with no parameters in most cases. We use this to constructor to create an empty matrix and pass this to other OpenCV methods.
Mat(Mat m, Rect roi) This constructor accepts two objects, one representing another matrix and the other representing the Region Of Interest . Rect class will be introduced in Chapter 6.

Table 2. The main methods provided by the Mat class.

Method & Description of Mat
int channels() Returns the number of matrix channels
int cols() Returns the number of columns in the matrix, which is same as width().
long elemSize() Returns the matrix element size in bypes.
int get(int row, int col, byte[] data) A Mat image is recorded in byte format from (<i>col</i> , <i>row</i>) to <i>data</i> .
int height() Returns the number of vertical pixels.
int rows() Returns the number of rows in the matrix, which is same as height().
int width() Returns the number of horizontal pixels.

3. Imgcodecs Class

The **Imgcodecs** class of the packate **org.opencv.imgcodecs** provides methods to read and write images. Using OpenCV, we can read an image and store it in a matrix (perform trasformations on the matrix if needed).

The imread method of the Imgcodecs class is used to read an image using OpenCV. Following is the syntax of this method.

```
Mat imread(String filename)
```

The argument *filename* is a variable of the String type representing the path of the file that is to be read.

The *imwrite* method of the *Imgcodecs* class is used to write an image using OpenCV. Following is the syntax of this method.

`boolean imwrite(String filename, Mat img)`

filename: A String variable representing the path where to save the file.

img: A Mat object representing the image to be written.

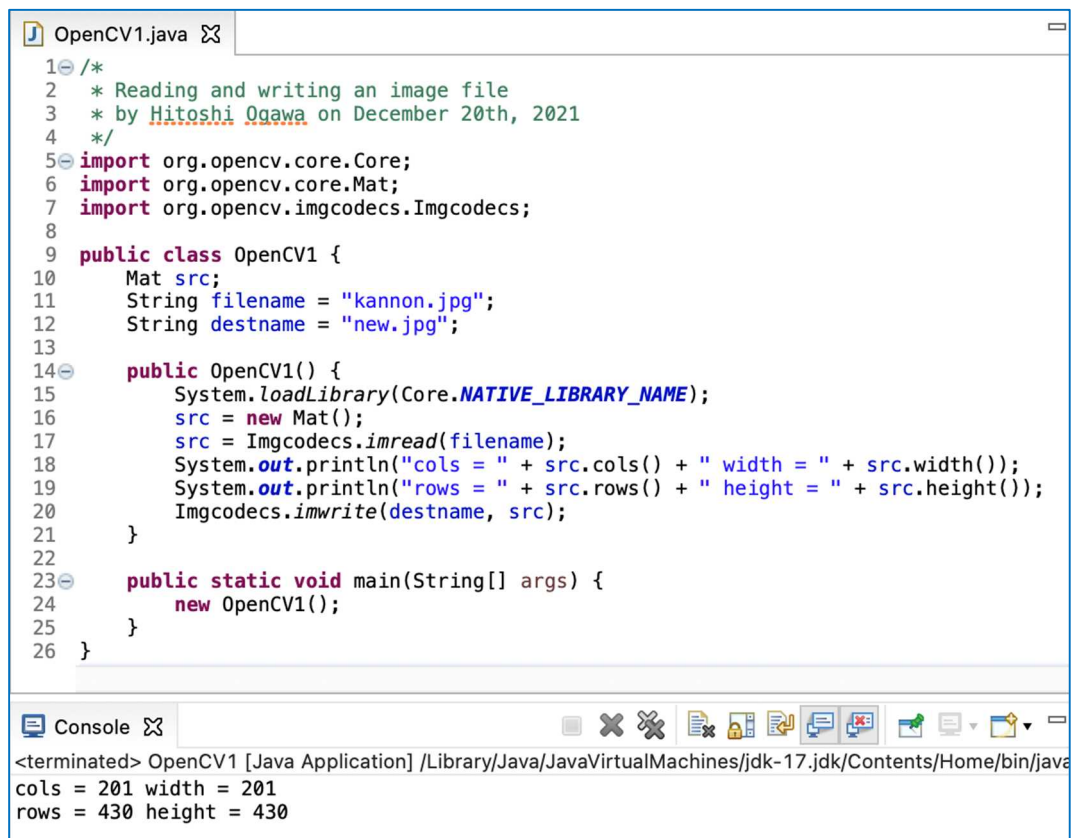
Figure 1 shows a program loading/saving of image file and using methods of Mat class and a result of the program.

In the field of OpenCV1 class, the Mat variable “src” is declared, and the String variable “filename” of the initial value “kannon.jpg”, and the String variable “destname” of the initial value “new.jpg” are declared.

At the beginning of the constructor, we need to execute

“System.loadLibrary(Core.NATIVE_LIBRARY_NAME);”

to load OpenCVnative (line 15). Otherwise, OpenCV instructions cannot be used.



```
1  /*
2  * Reading and writing an image file
3  * by Hitoshi Ogawa on December 20th, 2021
4  */
5  import org.opencv.core.Core;
6  import org.opencv.core.Mat;
7  import org.opencv.imgcodecs.Imgcodecs;
8
9  public class OpenCV1 {
10     Mat src;
11     String filename = "kannon.jpg";
12     String destname = "new.jpg";
13
14     public OpenCV1() {
15         System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
16         src = new Mat();
17         src = Imgcodecs.imread(filename);
18         System.out.println("cols = " + src.cols() + " width = " + src.width());
19         System.out.println("rows = " + src.rows() + " height = " + src.height());
20         Imgcodecs.imwrite(destname, src);
21     }
22
23     public static void main(String[] args) {
24         new OpenCV1();
25     }
26 }
```

<terminated> OpenCV1 [Java Application] /Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java
cols = 201 width = 201
rows = 430 height = 430

Figure 1. A program loading/saving of image file.

In line 17, the image in the “kannon.jpg” is loaded to “src”. In the execution result of lines 18 and 19, it is understood that “cols” and “width”, and “rows” and “height” obtain the same result.

In line 20, the image assigned to “src” is saved in the file “destname”.

4. Image Display

OpenCV does not provide a way to display images in Java. Therefore, there is a method using BufferedImage. An example is shown in Figure 2.

Since the structure of “OpenCV2.java” is similar to the structure of AffineMapX (X = 1 to 6) of the twelfth day, the takelImage method and matToBufferedImage method will be introduced.

In the takelImage method, the file “kannon.jpg” is loaded into the Mat variable “src” (line 34). The content of “src” is converted to BufferedImage by matToBufferedImage method, and it is assigned to “bimage” (line 35).

In the matToBufferedImage method, in order to obtain the necessary work area “data”, the horizontal length (cols), the vertical length (rows) and the number of bytes used for one pixel (elemSize) are acquired (lines 46 to 48). The variable “data” is a byte array of size which is (cols * rows * elemSize) (line 49). The image recorded in Mat is copied from coordinate (0, 0) to array “data” (line 52).

The conversion process varies depending on whether the image is a binary image or color image. In the case of a binary image, the value of “matrix.channels()” is 1, and “BufferedImage.[TYPE_BYTE_GRAY](#)” is assigned to the variable “type” (line 56).

In the case of a color image, the value of “matrix.channels()” is 3, and “BufferedImage.[TYPE_3BYTE_BGR](#)” is substituted for the variable “type” (line 59), and the color expression in each pixel is changed from BGR to RGB (lines 61 to 65). An area for storing the image is prepared in the BufferedImage variable “image2” (line 70). Image data of a horizontal length “cols” and a vertical length “rows” are written from (0, 0) in “data” to a writable raster of “image2” (line 71). Where, raster is a rectangular array for images.

Figure 3 shows the execution result of “OpenCV2.java”.

The source code of “matToBufferedImage” method can be obtained from “DisplayTest.java” or “CameraCapture.java” published on “OpenCV” page in Resources of Programming Language course, manaba+R.

```
*OpenCV2.java  ✕

1  /*
2   * Displaying images loaded with OpenCV
3   * by Hitoshi Ogawa on December 20th, 2021
4   */
5  import java.awt.Graphics;
6  import java.awt.image.BufferedImage;
7  import javax.swing.JFrame;
8  import org.opencv.core.Core;
9  import org.opencv.core.Mat;
10 import org.opencv.imgcodecs.Imgcodecs;
11
12 public class OpenCV2 extends JFrame {
13     BufferedImage bimage;
14     Mat src;
15     String filename = "kannon.jpg";
16
17     public OpenCV2() {
18         System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
19         setTitle("OpenCV2");
20         setSize(700, 500);
21         setDefaultCloseOperation(EXIT_ON_CLOSE);
22         setVisible(true);
23         takeImage();
24     }
25
26     public void paint(Graphics g){
27         if(bimage != null){
28             g.drawImage(bimage, 10, 30, this);
29         }
30     }
31
32     public void takeImage(){
33         src = new Mat();
34         src = Imgcodecs.imread(filename);
35         bimage = matToBufferedImage(src);
36         System.out.println("width = " + bimage.getWidth()
37                             + " height = " + bimage.getHeight());
38         repaint();
39     }
40
41     public static void main(String[] args) {
42         new OpenCV2();
43     }
44
45     public static BufferedImage matToBufferedImage(Mat matrix) {
46         int cols = matrix.cols();
47         int rows = matrix.rows();
48         int elemSize = (int)matrix.elemSize();
49         byte[] data = new byte[cols * rows * elemSize];
50         int type;
51
52         matrix.get(0, 0, data);
53
54         switch (matrix.channels()) {
55             case 1:
56                 type = BufferedImage.TYPE_BYTE_GRAY;
57                 break;
58             case 3:
59                 type = BufferedImage.TYPE_3BYTE_BGR;
60                 byte b;
61                 for(int i = 0; i < data.length; i = i + 3) {
62                     b = data[i];
63                     data[i] = data[i+2];
64                     data[i+2] = b;
65                 }
66                 break;
67             default:
68                 return null;
69         }
70         BufferedImage image2 = new BufferedImage(cols, rows, type);
71         image2.getRaster().setDataElements(0, 0, cols, rows, data);
72         return image2;
73     }
74 }
```

Figure 2. An example program displaying Mat.

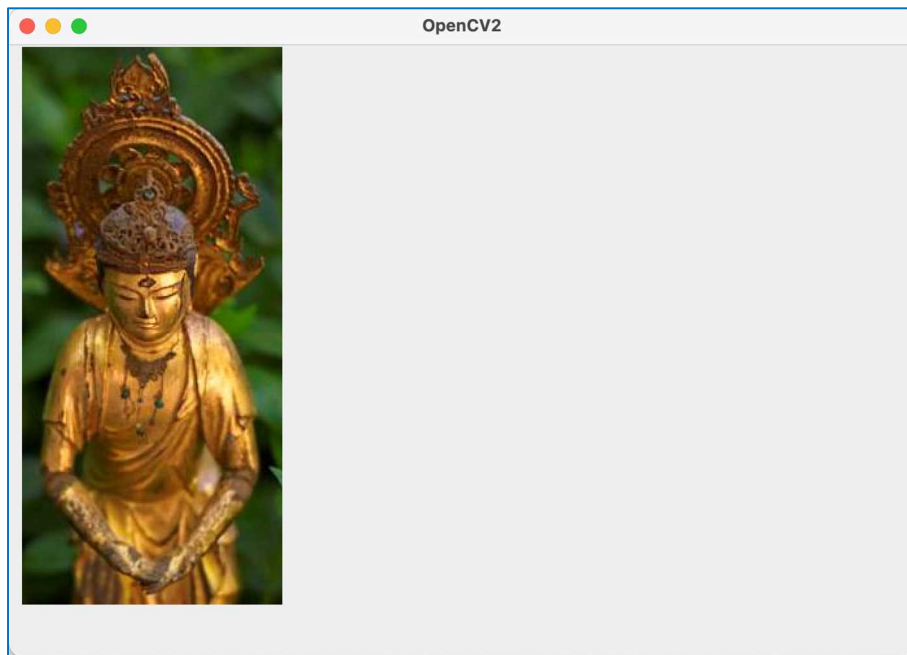


Figure 3. The execution result of OpenCV2.

5. Colored Image to GrayScale and Binary

An example of a program that converts a color image into a gray image and a binary image. The source code of the new paint method and the new takelImage method is shown in Figure 4.

In the paint method, bimage1, bimage2 and bimage3 are displayed from the left. In the takelImage method, the cvtColor method and threshold method of the Imgproc class.

The syntax of the cvtColor method is as follows.

```
void cvtColor(Mat src, Mat dst, int code)
```

src: A matrix representing the source.

dst: A matrix representing the destination.

code: An integer code representing the type of the conversion.

In the case of converting BGR to gray, "Imgproc.COLOR_BGR2GRAY".

The syntax of the threshold method is as follows.

```
void threshold(Mat src, Mat dst, double thresh, double maxval, int code)
```

src: A matrix representing the source.

dst: A matrix representing the destination.

thresh: An integer representing the threshold value.

maxval: An integer representing the maximum value.

code: An integer code representing the type of the conversion.

In the case of converting gray to binary, "Imgproc.THRESH_BINARY".

In this example, the color image "src" is converted to a gray image "gray". The gray image "gray" is converted to a binary image "binary". The value of *thresh* is set to 80, and the value of *maxval* is set to 255.

The execution result is shown in Figure 5.

```
26
27 public void paint(Graphics g){
28     if(bimage1 != null){
29         g.drawImage(bimage1, 10, 30, this);
30     }
31     if(bimage2 != null){
32         g.drawImage(bimage2, 240, 30, this);
33     }
34     if(bimage3 != null){
35         g.drawImage(bimage3, 470, 30, this);
36     }
37 }
38
39 public void takeImage(){
40     src = new Mat();
41     gray = new Mat();
42     binary = new Mat();
43     src = Imgcodecs.imread(filename);
44     Imgproc.cvtColor(src, gray, Imgproc.COLOR_BGR2GRAY);
45     Imgproc.threshold(gray, binary, 80, 255, Imgproc.THRESH_BINARY);
46     bimage1 = matToBufferedImage(src);
47     bimage2 = matToBufferedImage(gray);
48     bimage3 = matToBufferedImage(binary);
49     repaint();
50 }
51
```

Figure 4. The paint and takeImage for GrayScale and Binary.

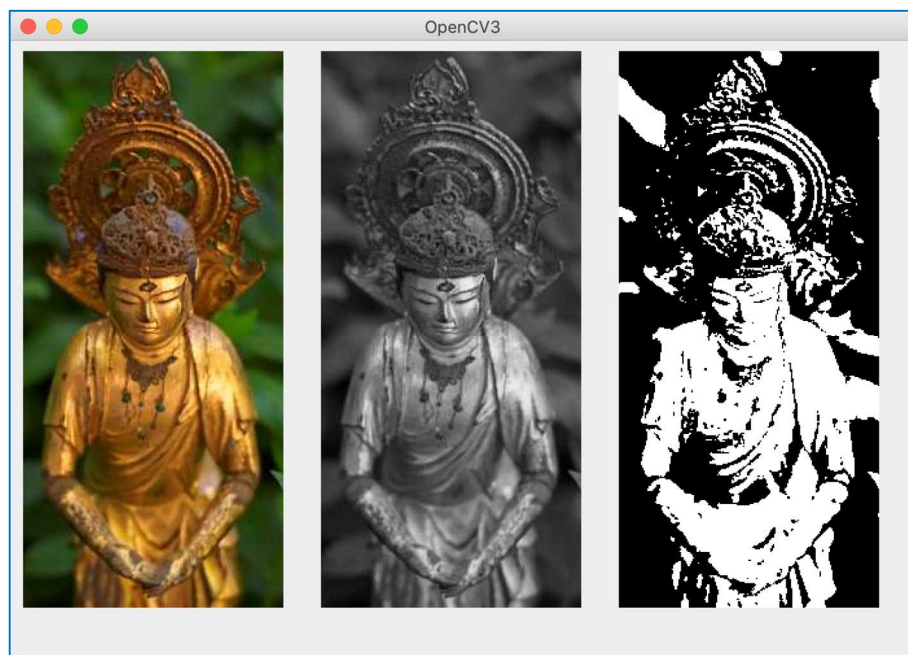


Figure 5. Obtaining GrayScale image and Binary image.

6. Cropping out images and Mosaics

Figure 6 shows an example of a program “OpenCV4.java” that cuts out a partial image from an image and makes the image mosaic.

In the field, Mat variables “src”, “part” and “moaic” are declared for the original image, the partial image and the mosaic image, respectively (line 19). An array “rect” for recording the coordinates of the rectangle created by the mouse is declared (line 22).

In the constructor, MouseListener and MouseMotionListener are registered. In the paint method, 3 BufferedImages and red rectangle are displayed. In the takeImage method, the original image is loaded and assigned to “src” (line 53). “src” is converted to BufferedImage which is assigned to “bimage1” (line 54).

A rectangular region is specified with the mouse, and when the mouse button is released, the partImage method is executed (line 114). A rectangle is set with the Rect class with the following syntax.

`Rect(int x, int y, int width, int height)`

(x, y) indicates the coordinate of the upper left corner of the image to be extracted. The *width* and *height* indicate the size of the image to be taken out.

The object indicating a rectangle specified by a mouse is assigned to the variable “roi” (line 59). Using “roi”, the Mat constructor is extracting the image to “part” and “mosaic” (lines 61 and 63). “part” is converted to BufferedImage which is assigned to “bimage2” (line 62). “mosaic” is reduced to 1/10 (lines 64 and 65). The image enlarging each pixel by ten times is obtained (lines 66 and 67), and is converted into BufferedImage which is assigned to biamge3 (lines 68).

Figure 7 shows the result of OpenCV4.

The Rect class fields are referenced to extract data from Rect class objects. Assuming that an instance of the Rect class is assigned to the variable “roi”, the values are referenced as shown in Table 3.

Table 3. Extracting data from Rect class objects “roi”.

field of “roi”	Data to be extraced
roi.x	Data of the first argument: Upper left x coordinate
roi.y	Data of the second argument: Upper left y coordinate
roi.width	Data of the third argument: Width size
roi.height	Data of the fourth argument: Vertical size


```

OpenCV4.java
1  /*
2   * Cropping out image and Mosaic image
3   * by Hitoshi Ogawa on December 20th, 2021
4   */
5  import java.awt.Color;
6  import java.awt.Graphics;
7  import java.awt.event.MouseAdapter;
8  import java.awt.event.MouseEvent;
9  import java.awt.image.BufferedImage;
10 import javax.swing.JFrame;
11 import org.opencv.core.Core;
12 import org.opencv.core.Mat;
13 import org.opencv.core.Rect;
14 import org.opencv.core.Size;
15 import org.opencv.imgcodecs.Imgcodecs;
16 import org.opencv.imgproc.Imgproc;
17
18 public class OpenCV4 extends JFrame {
19     BufferedImage bimage1, bimage2, bimage3;
20     Mat src, part, mosaic;
21     String filename = "kannon.jpg";
22     int[][] rect = {{0, 0}, {0, 0}};
23
24     public OpenCV4() {
25         System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
26         setTitle("OpenCV4");
27         setSize(700, 500);
28         setDefaultCloseOperation(EXIT_ON_CLOSE);
29         MyListener myListener = new MyListener();
30         addMouseListener(myListener);
31         addMouseMotionListener(myListener);
32         setVisible(true);
33         takeImage();
34     }
35
36     public void paint(Graphics g){
37         if(bimage1 != null){
38             g.drawImage(bimage1, 10, 30, this);
39         }
40         if(bimage2 != null){
41             g.drawImage(bimage2, 240, 30, this);
42         }
43         if(bimage3 != null){
44             g.drawImage(bimage3, 470, 30, this);
45         }
46         g.setColor(Color.red);
47         g.drawRect(rect[0][0], rect[0][1],
48                 rect[1][0] - rect[0][0], rect[1][1] - rect[0][1]);
49     }
50
51     public void takeImage(){
52         src = new Mat();
53         src = Imgcodecs.imread(filename);
54         bimage1 = matToBufferedImage(src);
55         repaint();
56     }
57
58     public void partImage() {
59         Rect roi = new Rect(rect[0][0] - 10, rect[0][1] - 30,
60                 rect[1][0] - rect[0][0], rect[1][1] - rect[0][1]);
61         part = new Mat(src, roi);
62         bimage2 = matToBufferedImage(part);
63         mosaic = new Mat(src, roi);
64         Imgproc.resize(mosaic, mosaic, new Size(),
65                 0.1, 0.1, Imgproc.INTER_NEAREST);
66         Imgproc.resize(mosaic, mosaic, new Size(),
67                 10.0, 10.0, Imgproc.INTER_NEAREST);
68         bimage3 = matToBufferedImage(mosaic);
69         repaint();
70     }

```

Figure 6. An example program for cutting out image and mosaic.

```

71
72 public static void main(String[] args) {
73     new OpenCV4();
74 }
75
76 public static BufferedImage matToBufferedImage(Mat matrix) {

        /* Omitted */

104 }
105
106 class MyListener extends MouseAdapter {
107
108     public void mousePressed(MouseEvent e) {
109         rect[0][0] = e.getX();
110         rect[0][1] = e.getY();
111     }
112
113     public void mouseReleased(MouseEvent e){
114         partImage();
115         repaint();
116     }
117
118     public void mouseDragged(MouseEvent e) {
119         rect[1][0] = e.getX();
120         rect[1][1] = e.getY();
121         repaint();
122     }
123 }
124 }

```

Figure 6. (continure).

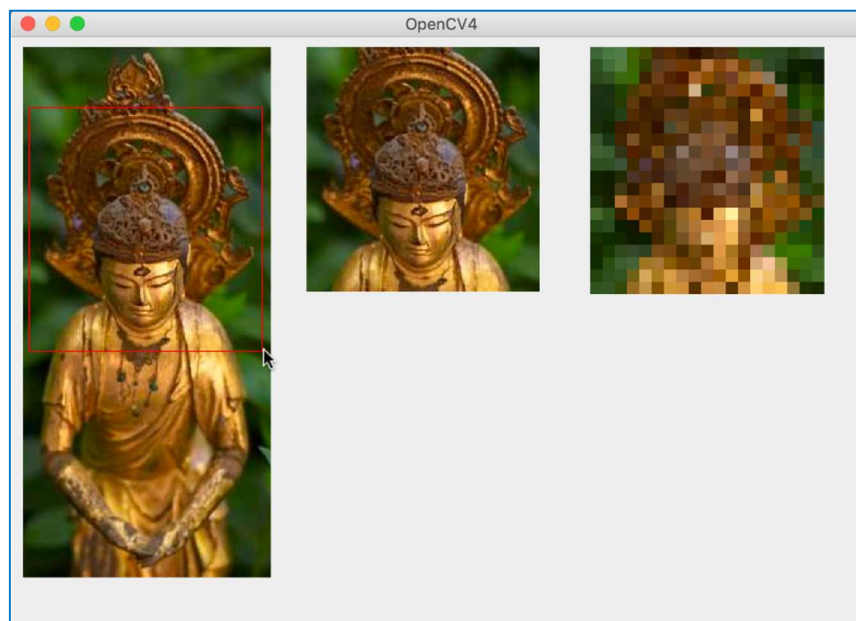


Figure 7. The result of OpenCV4.

7. Using Built-in Camera

The built-in camera cannot be accessed from IDE (Integrated Development Environment, for example, Eclipse) due to security issues. However, in order to handle a program that processes images obtained from a built-in camera with OpenCV, consider the system structure shown in Figure 8.

7. 1 Program to acquire images from the built-in camera

Photo Booth is provided as standard on MacBook. Despite the image from the camera is 1280x720, the size of the image to be saved in Photo Booth is 1080X720. Therefore, in the program using OpenCV, the image size is assumed to be 1080x720.

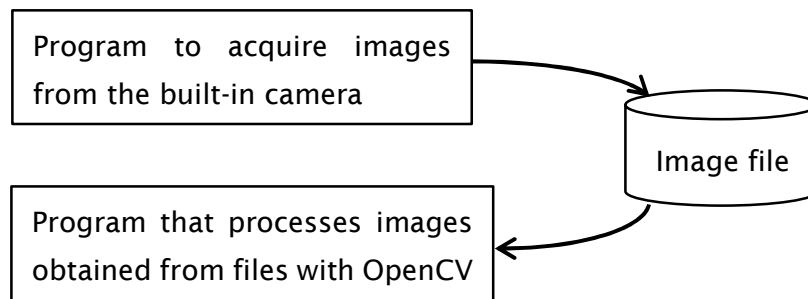


Figure 8. The system structure to process images from built-in camera with OpenCV.

The program started in the terminal can handle the built-in camera. If the Java program "CamraCapture.java"¹ using OpenCV is started in the terminal, the image can be acquired from the built-in camera. This program saves an image with a size of 1080x720 to the file name "FromCamera.jpg" in the same folder where it was started.

7. 2 Image Drawing Program

The program "FromCamera1.java" that displays the image from the built-in camera is almost the same as "OpenCV2.java".

The source code of "FromCamera1.java" is shown in Figure 9. The following items are different.

¹ "CamraCapture.java" is introduced in Chapter 5 of Install OpenCV 4.x under macOS (Mojave and Catalina) (the file "OpenCV_install.docx"). "CameraCapture.java" can be downloaded from "OpenCV" page in Resources of "Programming Language" course, manaba+R.

(a) File path

"CameraCapture.java" was used for image acquisition. This program was placed in the OpenCV folder in the Ogawa's home folder and executed. So, the image file is in /Users/ogawa/OpenCV/FromCamera.jpg (refer line 15).

(b) Image display size

The size of the image supplied by "CameraCapture.java" or Photo Booth is 1080x720. Since the image is too large to be displayed on the monitor in this size, including the processing result, it is displayed in half the length (lines 28 and 29).

Figure 10 shows an example of the execution result of the program "FromCamera1.java".

```
*FromCamera1.java
1  /*
2   * Displaying the image from the built-in camera
3   * by Hitoshi Ogawa on December 20th, 2021
4   */
5  import java.awt.Graphics;
6  import java.awt.image.BufferedImage;
7  import javax.swing.JFrame;
8  import org.opencv.core.Core;
9  import org.opencv.core.Mat;
10 import org.opencv.imgcodecs.Imgcodecs;
11
12 public class FromCamera1 extends JFrame {
13     BufferedImage bimage;
14     Mat src;
15     String filename = "/Users/ogawa/OpenCV/FromCamera.jpg";
16
17     public FromCamera1() {
18         System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
19         setTitle("FromCamera1");
20         setSize(560, 400);
21         setDefaultCloseOperation(EXIT_ON_CLOSE);
22         setVisible(true);
23         takeImage();
24     }
25
26     public void paint(Graphics g){
27         if(bimage != null){
28             g.drawImage(bimage, 10, 30,
29                       bimage.getWidth() / 2, bimage.getHeight() / 2, this);
30         }
31     }
32
33     public void takeImage(){
34         src = new Mat();
35         src = Imgcodecs.imread(filename);
36         bimage = matToBufferedImage(src);
37         System.out.println("width = " + bimage.getWidth()
38                           + " height = " + bimage.getHeight());
39         repaint();
40     }
41
42     public static void main(String[] args) {
43         new FromCamera1();
44     }
45 }
```

Figure 9. The source code of "FromCamera1.java".

```

46 public static BufferedImage matToBufferedImage(Mat matrix) {
47     int cols = matrix.cols();
48     int rows = matrix.rows();
49     int elemSize = (int)matrix.elemSize();
50     byte[] data = new byte[cols * rows * elemSize];
51     int type;
52
53     matrix.get(0, 0, data);
54
55     switch (matrix.channels()) {
56     case 1:
57         type = BufferedImage.TYPE_BYTE_GRAY;
58         break;
59     case 3:
60         type = BufferedImage.TYPE_3BYTE_BGR;
61         byte b;
62         for(int i = 0; i < data.length; i = i + 3) {
63             b = data[i];
64             data[i] = data[i+2];
65             data[i+2] = b;
66         }
67         break;
68     default:
69         return null;
70     }
71     BufferedImage image2 = new BufferedImage(cols, rows, type);
72     image2.getRaster().setDataElements(0, 0, cols, rows, data);
73     return image2;
74 }
75 }

```

Figure 9. The source code of "FromCamera1.java" (Continue).

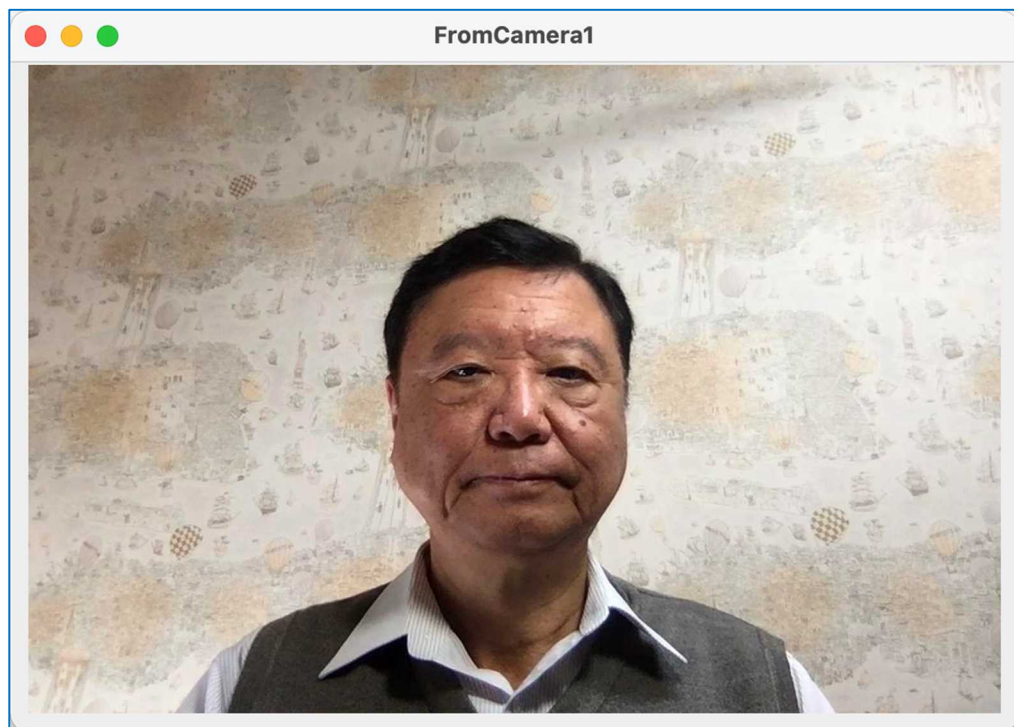


Figure 10. An example of the execution result of "FromCamera1.java".