Bureau d'étude Twizy
Programmation OpenCV sous Java

Abdelkader Lahmadi, Jean-Philippe Mangeot

Ingénierie des systèmes numériques

Avril 2017

Introduction

- OpenCV (www.opencv.org) est une bibliothèque de traitement d'image et des algorithmes pour la vision par ordinateur
- Elle supporte les langages de programmation : C++, Python et Java
- Elle est disponible pour Linux, Windows, Mac OS et Android
- Modules
  - Core : structures de base, opérations sur les matrices, dessiner sur les images
  - Imgproc : traitement et transformation d'images, détection de contours, filtrage, point d'intérêt,…
  - Highgui : lecture et écriture de fichiers
  - Features2d : descripteurs
  - video : traitement de flux vidéo.

- Alternative
  - www.simpleCV.org pour python

Installation de OpenCV sous Windows

- Nous utilisons la version 2.4.13 de OpenCV disponible sous :
  - https://github.com/opencv/opencv/releases
  - https://sourceforge.net/projects/opencvlibrary/files/opencv-win/2.4.13/
- Configurer Eclipse pour inclure OpenCV comme bibliothèque utilisateur
  - http://docs.opencv.org/2.4/doc/tutorials/introduction/java_eclipse/java_eclipse.html

Votre première application OpenCV

```
1  import org.opencv.core.Core;
2  import org.opencv.core.CvType;
3  import org.opencv.core.Mat;
4
5
6  public class HelloOpenCV {
7
8      public static void main(String[] args) {
9          System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
10         Mat mat = Mat.eye( 3, 3, CvType.CV_8UC1 );
11         System.out.println( "mat = " + mat.dump() );
12     }
```

Charger la bibliothèque native d'OpenCV

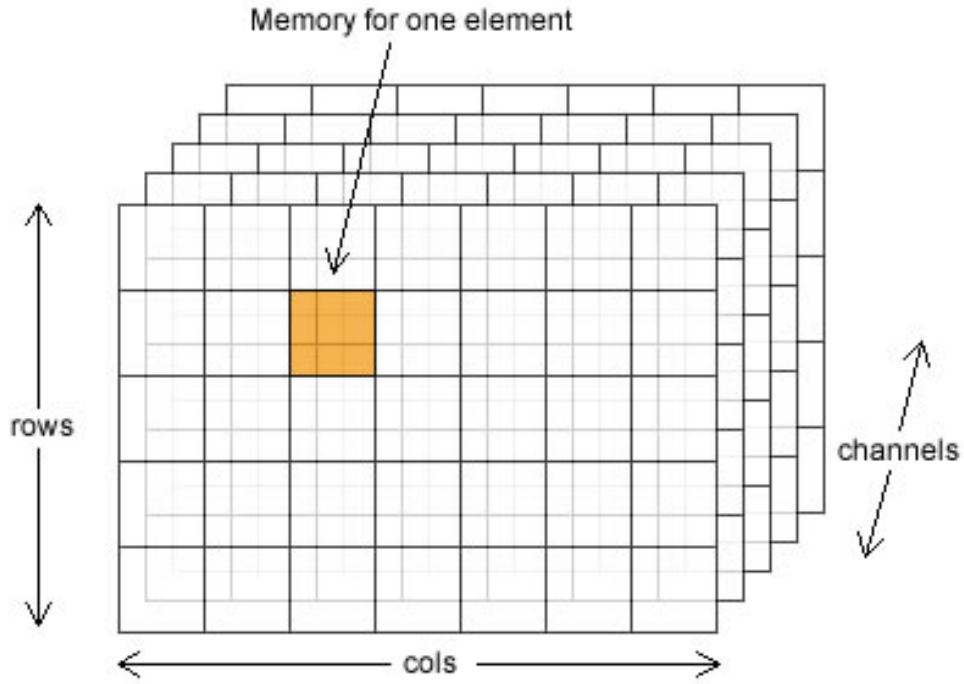
Créer une matrice identité 3x3 d'une seule dimension : CV_8UC1

Problems @ Javadoc Declaration Console LogCat

```
<terminated> HelloOpenCV [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_101.jdk/Contents/Home/bin/java (Apr 2, 2017, 11:50:58 AM)
mat = [1, 0, 0;
  0, 1, 0;
  0, 0, 1]
```

OpenCV : l'objet Mat

- Une classe pour stocker des matrices, des images, des vecteurs …
- Un tableau à n-dimensions : lignes, colonnes et canaux (composantes)



CvType.***CV_8UC1 : mono canal 8-bit***
CvType.***CV_8UC3 : 3 canaux 8-bit***
CvType.***CV_32FC1 : mon canal 32-bits***
CvType.***CV_32FC3 : 3 canaux 32-bits***

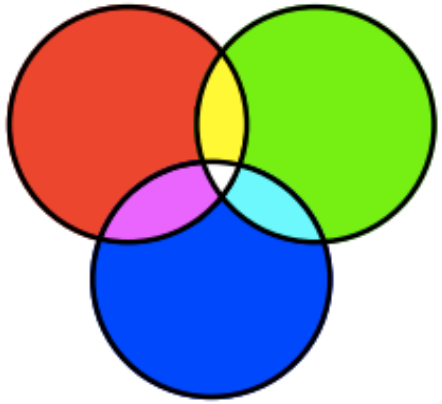
Les couleurs : quelques définitions

- La teinte : le nom de la couleur, c'est à dire la longueur d'onde dominante
- La saturation : le degré de dilution de la couleur dans la lumière blanche
- La luminosité : l'intensité de la lumière achromatique (c'est la lumière avec toutes les longueurs d'onde de façon approximativement égales)

Les espaces de couleurs

- L'espace RGB (Red, Green, Blue) ou (Rouge Vert Bleu) : est calqué sur notre perception visuelle sur les 3 couleurs de base
  - Rouge = (255,0,0), Vert = (0,255,0), Bleu=(0,0,255),
  - Noir= (0,0,0), Blanc = (255,255,255), Jaune = (255,255,0)

- L'espace YCM (Yellow, Cyan, Magenta) est basé sur 3 couleurs : le jaune, le cyan et le magenta

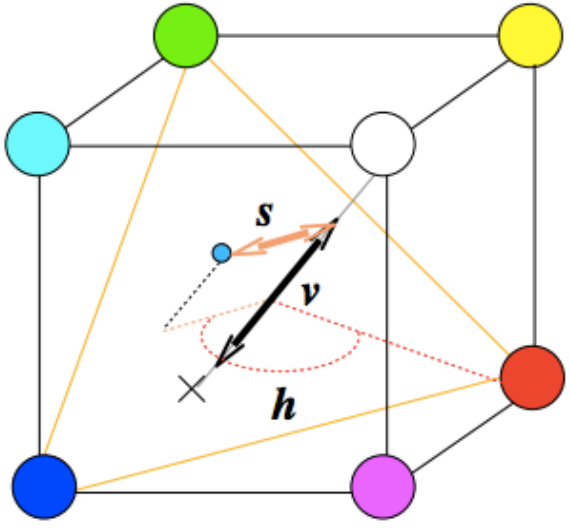


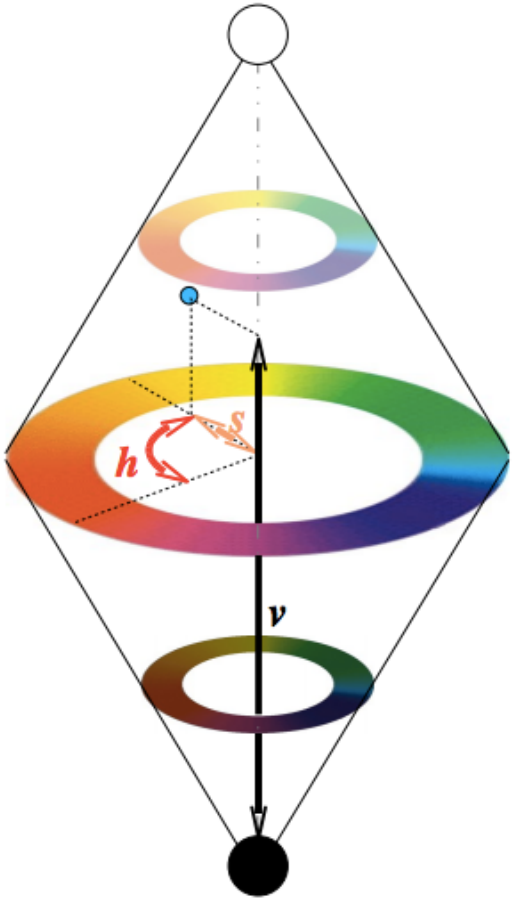
Les espaces de couleurs

- L'espace luminance Y, chrominance bleue (U) et chrominance rouge (V): YUV
  - $Y = 0.299\ R + 0.587\ G + 0.114\ B$
  - $U = 0.493\ (B - Y)$
  - $V = 0.877\ (R - Y)$
- L'espace HSV (Hue, Saturation, Value) ou (teinte, saturation, valeur)
  - La teinte (Hue) est codée suivant l'angle qui lui correspond sur le cercle des couleurs (c'est la couleur à désigner)
  - La saturation est l'intensité de la couleur : taux de pureté de la couleur de l'éclatant au niveau de gris
  - La valeur est la brillance de la couleur : l'intensité lumineuse de la couleur du noir absolu au blanc

Les couleurs : l'espace HSV

- Le passage de RGB à HSV se fait par une transformation non linéaire





$$v = \frac{r+g+b}{3}$$

$$s = 1 - \frac{3\,min(r,g,b)}{r+g+b}$$

$$h = \begin{cases} \theta & si \quad b \leqslant g \\ 2\pi - \theta & si \quad b > g \end{cases} \quad \theta = \arccos\left(\frac{(r-g)+(r-b)}{2\sqrt{(r-g)^2+(r-b)(g-b)}} \right)$$

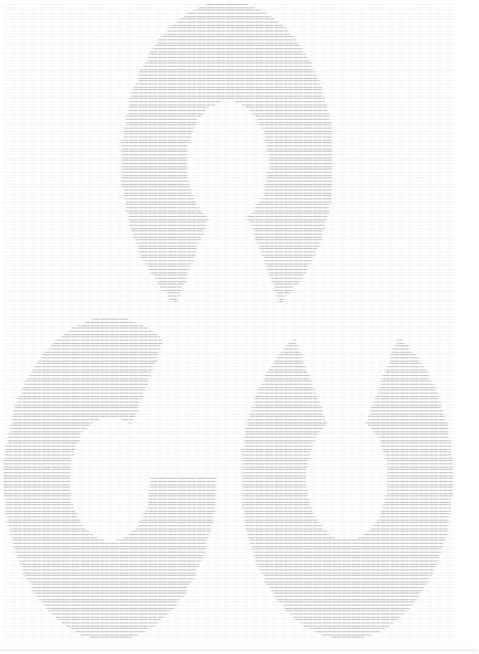
Lecture d'un fichier image

- *Classe : org.opencv.Highgui*
- *public static Mat **imread**(java.lang.String filename)*
- *public static Mat **imread**(java.lang.String filename, int flags)*
- Formats supportés : bmp,jpeg,jpg,jpe,png, pbm, pgm,ppm, tiff, tif

```
public static Mat LectureImage(String fichier) {
    File f = new File(fichier);
    Mat m = Highgui.imread(f.getAbsolutePath());
    return m;

}
```

Exercice 1: lecture et affichage d'une image

- Récupérez depuis arche le fichier opencv.png
- Ecrire une fonction pour lire le fichier : la fonction retourne la matrice de l'image
- Afficher le contenu de la matrice en mode texte : le symbole . pour le blanc et le symbole + pour les autres couleurs



Solution

```
public static void main(String[] args) {
    System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
    Mat m = utils.LectureImage("opencv.png");
    for (int i=0; i < m.height();i++) {
        for(int j=0; j < m.width();j++) {
            double[] BGR = m.get(i, j);
            if (BGR[0] == 255 && BGR[1] == 255 && BGR[1] == 255){
                System.out.print(".");
            }else{
                System.out.print("+");
            }
        }
        System.out.println();
    }
}
```

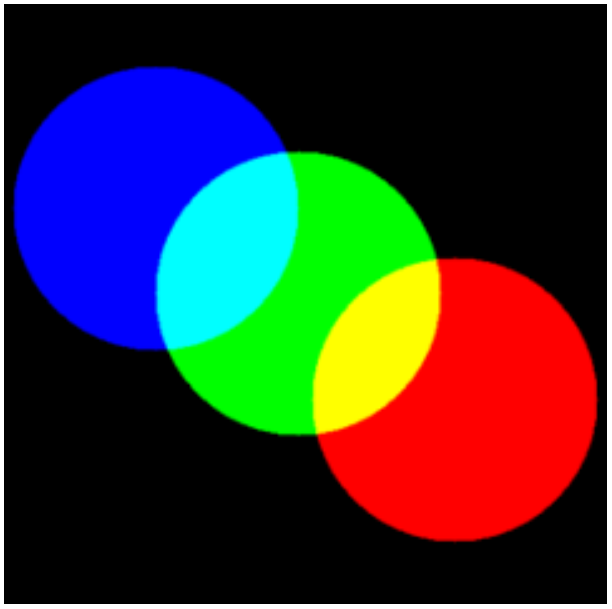
Exercice : afficher les canaux couleur d'une l'image

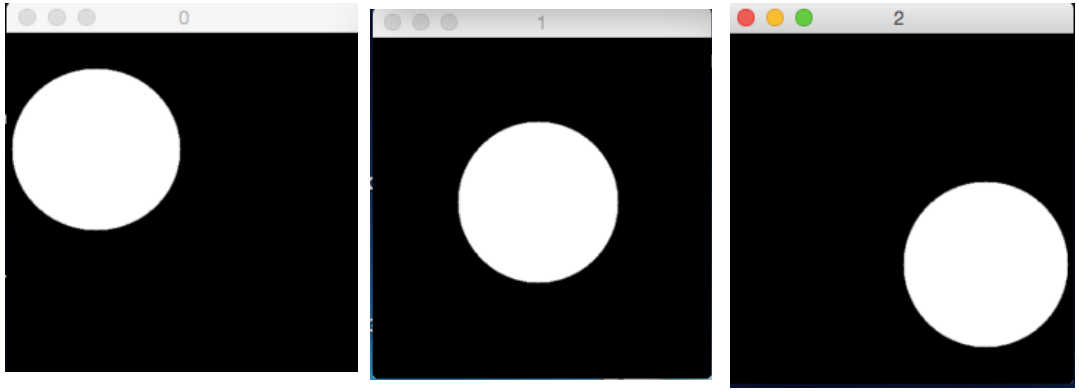
- Afficher dans 3 fenêtres graphiques les 3 composantes couleur de l'image *bgr.png* : Rouge, Vert et Bleu
- Pour décomposer une matrice d'une image : *core.split*
- Pour afficher une matrice dans une fenêtre graphique, utiliser la méthode suivante :

```
public static void ImShow(String title, Mat img) {
    MatOfByte matOfByte = new MatOfByte();
    Highgui.imencode(".png", img, matOfByte);
    byte[] byteArray = matOfByte.toArray();
    BufferedImage bufImage = null;
    try {
        InputStream in = new ByteArrayInputStream(byteArray);
        bufImage = ImageIO.read(in);
        JFrame frame = new JFrame();
        frame.setTitle(title);
        frame.getContentPane().add(new JLabel(new ImageIcon(bufImage)));
        frame.pack();
        frame.setVisible(true);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Première solution : mode niveaux de gris

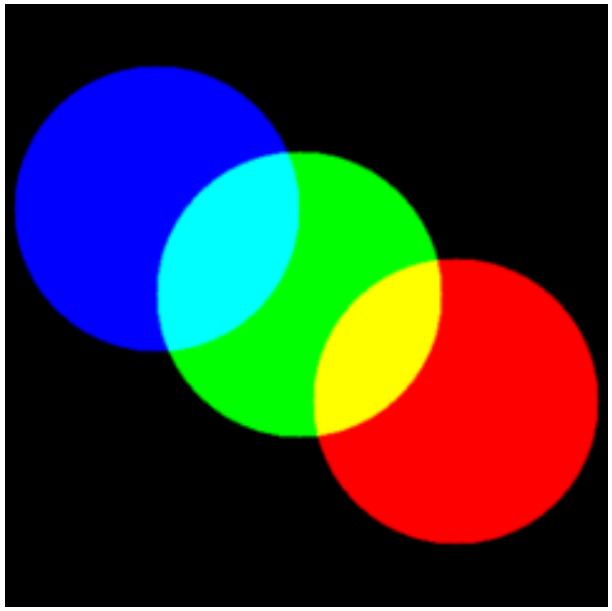
```
public static void main(String[] args) {
    System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
    Mat m = utils.LectureImage("bgr.png");
    Vector<Mat> channels = new Vector<Mat>();
    Core.split(m, channels);
    // BGR order
    for (int i=0; i < channels.size();i++) {
        utils.ImShow(Integer.toString(i), channels.get(i));
    }
}
```

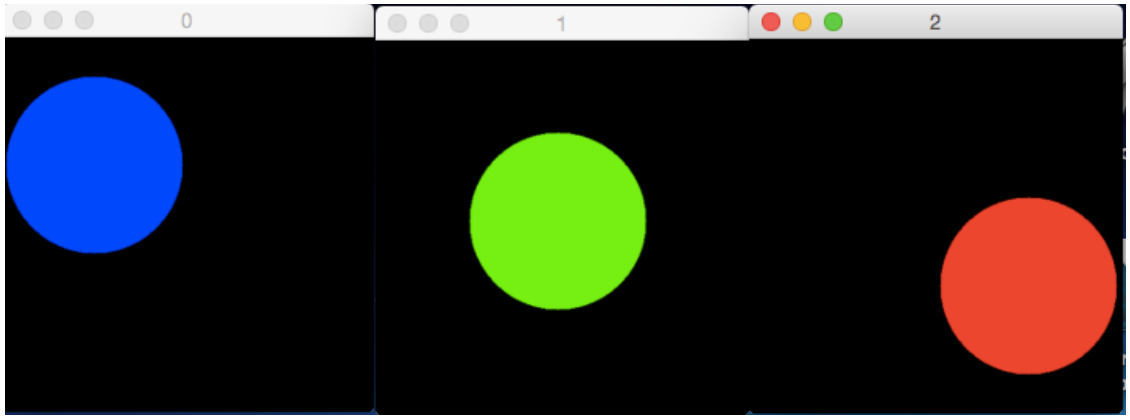




Deuxième solution : mode RGB

```
public static void main(String[] args) {
    System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
    Mat m = utils.LectureImage("bgr.png");
    Vector<Mat> channels = new Vector<Mat>();
    Core.split(m, channels);
    // BGR order
    Mat dst = Mat.zeros(m.size(), m.type());
    Vector<Mat> chans = new Vector<Mat>();
    Mat empty = Mat.zeros(m.size(), CvType.CV_8UC1);
    for (int i=0; i < channels.size();i++) {
        utils.ImShow(Integer.toString(i), channels.get(i));
        chans.removeAllElements();
        for (int j=0; j<channels.size();j++) {
            if (j != i) {
                chans.add(empty);
            }else{
                chans.add(channels.get(i));
            }
        }
        Core.merge(chans, dst);
        utils.ImShow(Integer.toString(i), dst);
    }
}
```





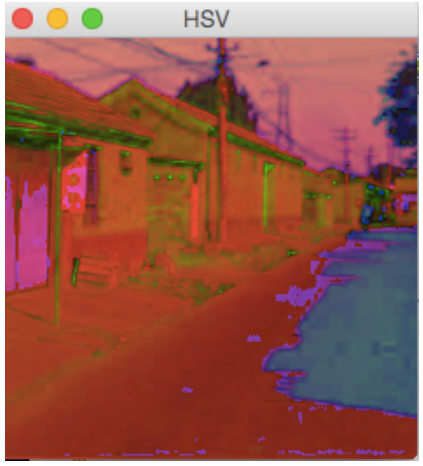
Changer l’espace de couleurs

- Classe Imgproc
- Méthode : cvtColor
- Transformation de couleurs : Imgproc.COLOR_XXX2XXX
  - COLOR_BGR2HSV
  - COLOR_BGR2RGB
  - COLOR_BGR2GRAY
  - ….
- Plages de valeurs HSV :
  - H : 0 .. 179
  - S : 0 .. 255
  - V : 0 .. 255

Exercice : passage de BGR à HSV

- Transposer l'image *hsv.png* en espace HSV
- Afficher le résultat dans une fenêtre graphique
- Extraire les 3 canaux HSV en 3 fenêtres différentes

Décomposition en niveaux de gris



Décomposition en niveaux HSV

Solution

```
Mat m = utils.LectureImage("hsv.png");
Mat output = Mat.zeros(m.size(),m.type());
Imgproc.cvtColor(m, output, Imgproc.COLOR_BGR2HSV);
utils.ImShow("HSV", output);
Vector<Mat> channels = new Vector<Mat>();
Core.split(output, channels);
double hsv_values[][] = {{1, 255, 255}, {179, 1, 255}, {179, 0, 1}};
for (int i=0; i< 3; i++) {
    utils.ImShow(Integer.toString(i)+"-HSV",channels.get(i));
    Mat chans[] = new Mat[3];
    for (int j=0; j < 3; j++) {
        Mat empty = Mat.ones(m.size(), CvType.CV_8UC1);
        Mat comp = Mat.ones(m.size(),CvType.CV_8UC1);
        Scalar v = new Scalar(hsv_values[i][j]);
        Core.multiply(empty,v,comp);
        chans[j] = comp;
    }
    chans[i] = channels.get(i);
    Mat dst = Mat.zeros(output.size(), output.type());
    Mat res = Mat.ones(dst.size(), dst.type());
    Core.merge(Arrays.asList(chans), dst);
    Imgproc.cvtColor(dst, res, Imgproc.COLOR_HSV2BGR);
    utils.ImShow(Integer.toString(i), res);
```

Seuillage d'une image par couleur

- Application d'un masque couleur sur une image
- Méthode OpenCV : Core.*inRange*
- Étapes:
  - Convertir l'image en HSV
  - Identifier les seuils de la couleur à extraire
  - Appliquer Core.inRange pour construire un ou plusieurs masques d'extraction de la couleur
  - Combiner les masquer avec l'opérateur binaire OU (Core.bitwise_OR)
  - Appliquer un filtre gaussien pour lisser les masques

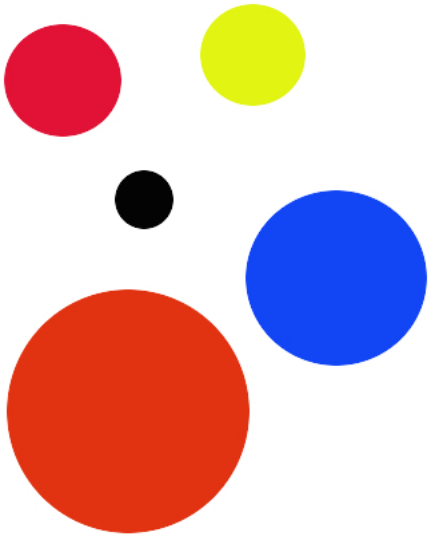
Exercice: seuillage

- Extraire les cercles rouge de l'image circles.jpg
- Première Solution : un seul seuil

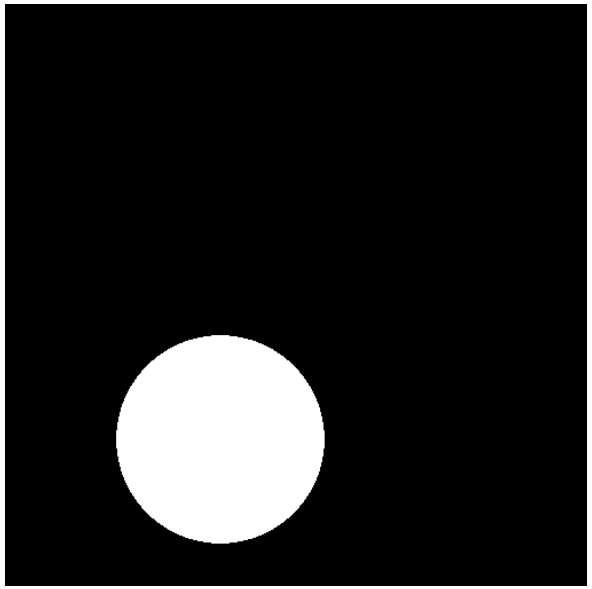
```
public static void main(String[] args) {
    System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
    Mat m = utils.LectureImage("circles.jpg");
    Mat hsv_image = Mat.zeros(m.size(),m.type());
    Imgproc.cvtColor(m, hsv_image, Imgproc.COLOR_BGR2HSV);
    Mat threshold_img = new Mat();
    Core.inRange(hsv_image, new Scalar(0,100,100), new Scalar(10,255,255), threshold_img);
    Imgproc.GaussianBlur(threshold_img, threshold_img, new Size(9, 9), 2, 2);
    utils.ImShow("Cercles rouge", threshold_img);

}
```

Borne inférieure

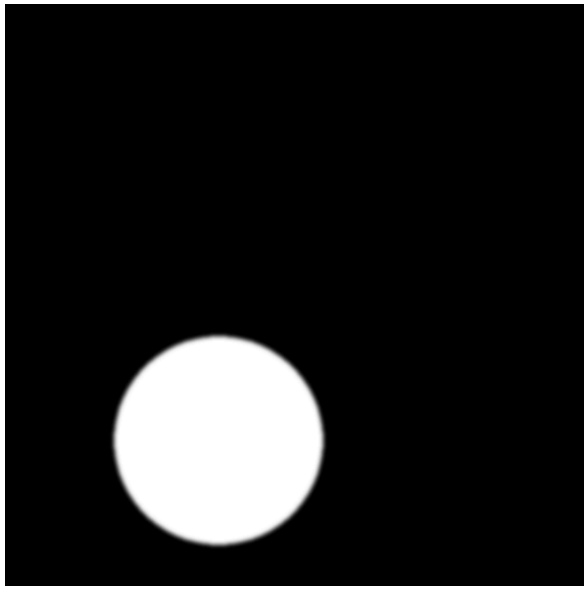
Borne supérieure



Sans lissage



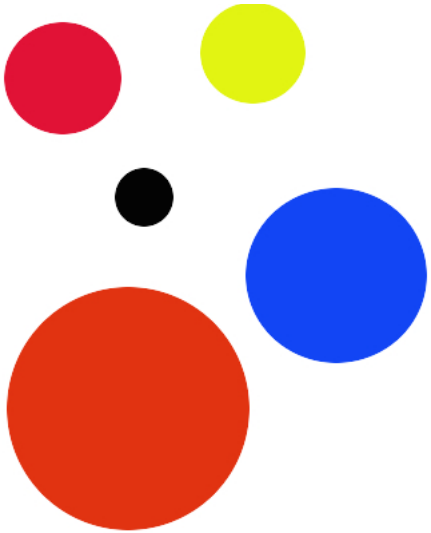
Avec lissage

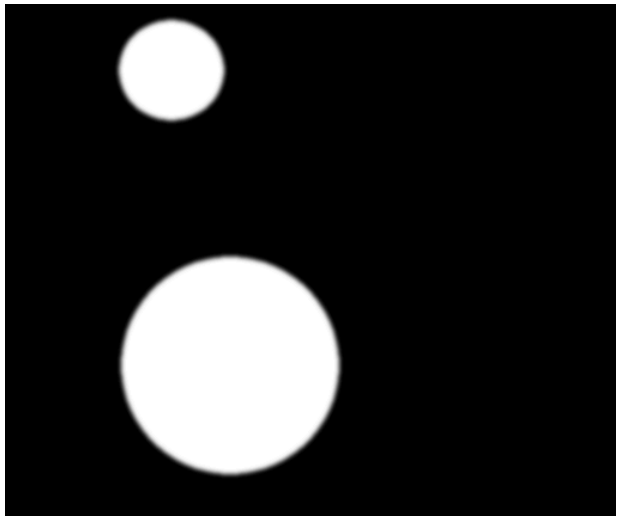


Exercice: seuillage

- Extraire les cercles rouge de l'image circles.jpg
- Deuxième Solution : plusieurs seuils

```
public static void main(String[] args) {
    System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
    Mat m = utils.LectureImage("circles.jpg");
    Mat hsv_image = Mat.zeros(m.size(),m.type());
    Imgproc.cvtColor(m, hsv_image, Imgproc.COLOR_BGR2HSV);
    Mat threshold_img1 = new Mat();
    Mat threshold_img2 = new Mat();
    Mat threshold_img = new Mat();
    Core.inRange(hsv_image, new Scalar(0,100,100), new Scalar(10,255,255), threshold_img1);
    Core.inRange(hsv_image, new Scalar(160,100,100), new Scalar(179,255,255), threshold_img2);
    Core.bitwise_or(threshold_img1, threshold_img2, threshold_img);
    Imgproc.GaussianBlur(threshold_img, threshold_img, new Size(9, 9), 2, 2);
    utils.ImShow("Cercles rouge", threshold_img);
}
```



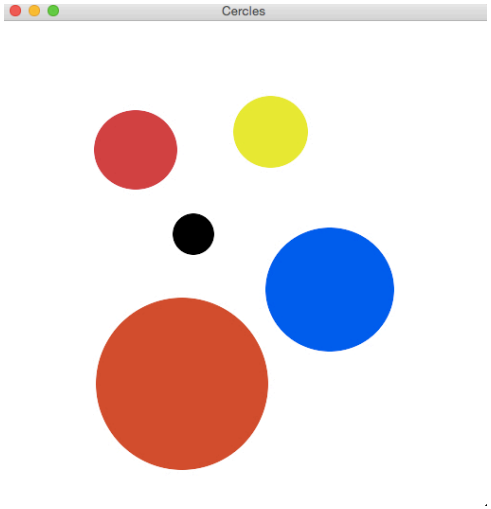


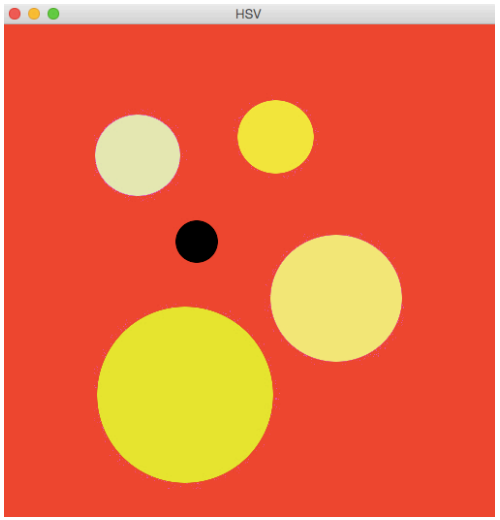
Détection de contours

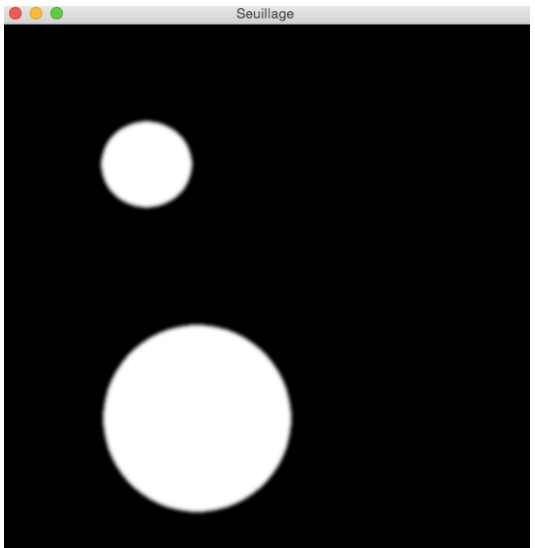
- Un contour est une frontière entre deux objets dans une image
- Méthode : filtre de Canny
  - Imgproc.Canny : détection de segments dans une image
  - Imgproc.findContours : détection de contours
  - Imgproc.drawContours : affichage de contours
- Technique
  - Transformer l'image en HSV
  - Extraire l'objet par un seuillage basée sur une couleur
  - Appliquer le filtre de Canny pour détecter les contours
  - Extraire une hiérarchie de contours

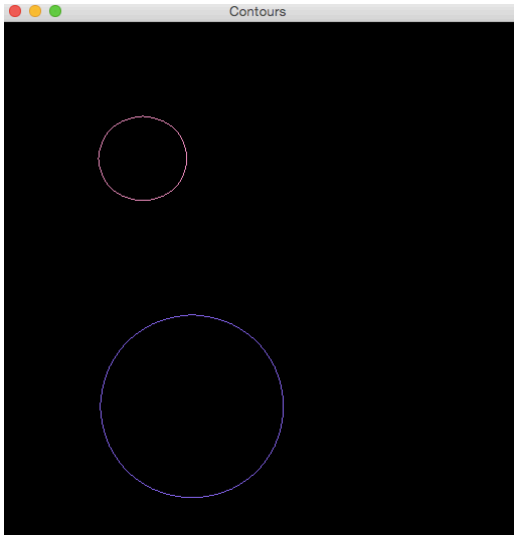
Exercice : Extraire les contours des cercles

- Extraire les contours des cercles rouge de l'image circles.jpg









Solution

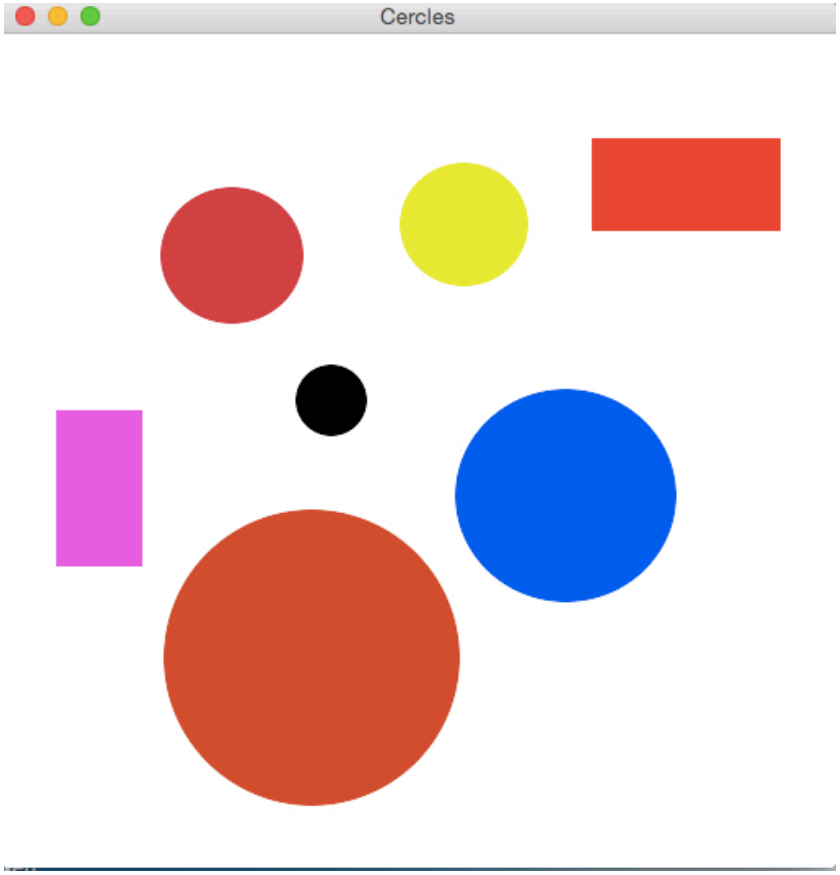
```
Mat m = utils.LectureImage("circles.jpg");
utils.ImShow("Cercles", m);
Mat hsv_image = Mat.zeros(m.size(),m.type());
Imgproc.cvtColor(m, hsv_image, Imgproc.COLOR_BGR2HSV);
utils.ImShow("HSV", hsv_image);
Mat threshold_img = utils.DetecterCercles(hsv_image);
utils.ImShow("Seuillage", threshold_img);
int thresh = 100;
Mat canny_output = new Mat();
List<MatOfPoint> contours = new ArrayList<MatOfPoint>();
MatOfInt4 hierarchy = new MatOfInt4();
Imgproc.Canny( threshold_img, canny_output, thresh, thresh*2);
Imgproc.findContours( canny_output, contours, hierarchy,Imgproc.RETR_EXTERNAL,
                      Imgproc.CHAIN_APPROX_SIMPLE);
Mat drawing = Mat.zeros( canny_output.size(), CvType.CV_8UC3 );
Random rand = new Random();
for( int i = 0; i< contours.size(); i++ )
{
    Scalar color = new Scalar( rand.nextInt(255 - 0 + 1) , rand.nextInt(255 - 0 + 1),
                               rand.nextInt(255 - 0 + 1) );
    Imgproc.drawContours( drawing, contours, i, color, 1, 8, hierarchy, 0, new Point() );
}
utils.ImShow("Contours",drawing);
```

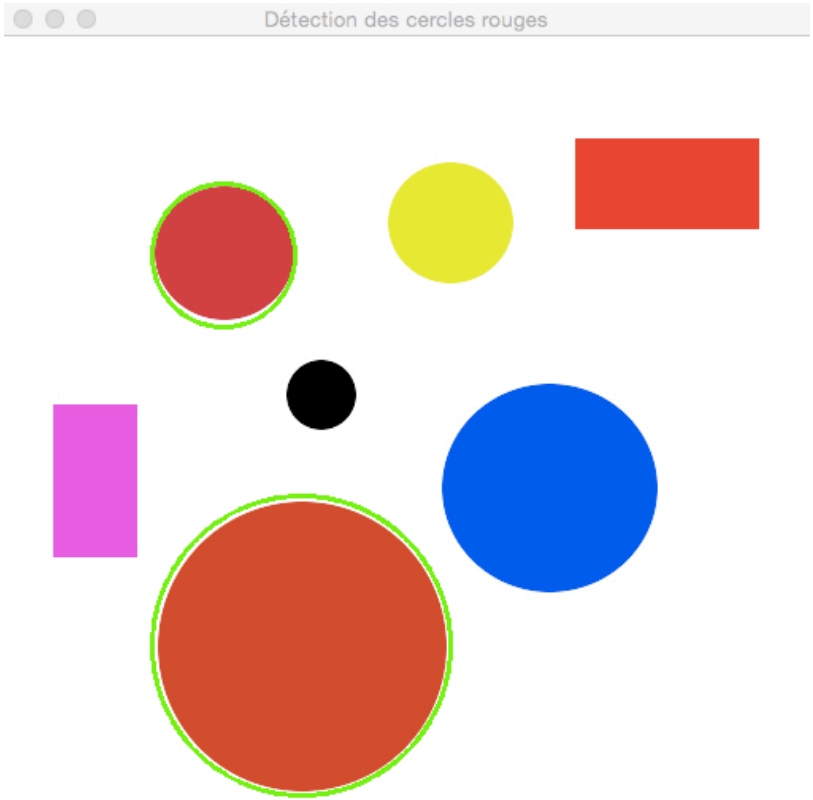
Reconnaître les formes de contours

- Un contour est une structure de type MatOfPoint
  - Il existe aussi la structure MatOfPoint2f pour les nombres réels (32-bit float)
- Algorithme
  - Calculer la surface de l'espace délimité par le contour : Imgproc.contourArea
  - Calculer le plus petit cercle qui enveloppe le contour avec la fonction Imgproc.minEnclosingCircle
  - Si Surface est proche de $PI*R^2$ alors c'est un cercle
  - Sinon calculer le polygone approximatif du contour
    - Si nombre de segments est 3 alors c'est un triangle
    - Si nombre de segments est >= 4 et <= 6 alors il s'agit d'un rectangle, ou d'un carré ou d'un polygone.

Exercice : Reconnaissance de cercles rouges

- Dans l'image circles_rectangels.jpg détecter et entourer les cercles rouges.





Solution

```
public static void main(String[] args) {
    System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
    Mat m = utils.LectureImage("circles_rectangles.jpg");
    utils.ImShow("Cercles", m);
    Mat hsv_image = Mat.zeros(m.size(),m.type());
    Imgproc.cvtColor(m, hsv_image, Imgproc.COLOR_BGR2HSV);
    utils.ImShow("HSV", hsv_image);
    Mat threshold_img = utils.DetecterCercles(hsv_image);
    utils.ImShow("Seuillage", threshold_img);
    List<MatOfPoint> contours = utils.DetecterContours(threshold_img);

    MatOfPoint2f matOfPoint2f = new MatOfPoint2f();
    float[] radius = new float[1];
    Point center = new Point();
    for (int c=0; c < contours.size();c++) {
        MatOfPoint contour = contours.get(c);
        double contourArea = Imgproc.contourArea(contour);
        matOfPoint2f.fromList(contour.toList());
        Imgproc.minEnclosingCircle(matOfPoint2f, center, radius);
        if ((contourArea/(Math.PI*radius[0]*radius[0])) >=0.8) {
            Core.circle(m, center, (int)radius[0], new Scalar(0, 255, 0), 2);
        }
    }
    utils.ImShow("Détection des cercles rouges", m);
}
```

Template matching

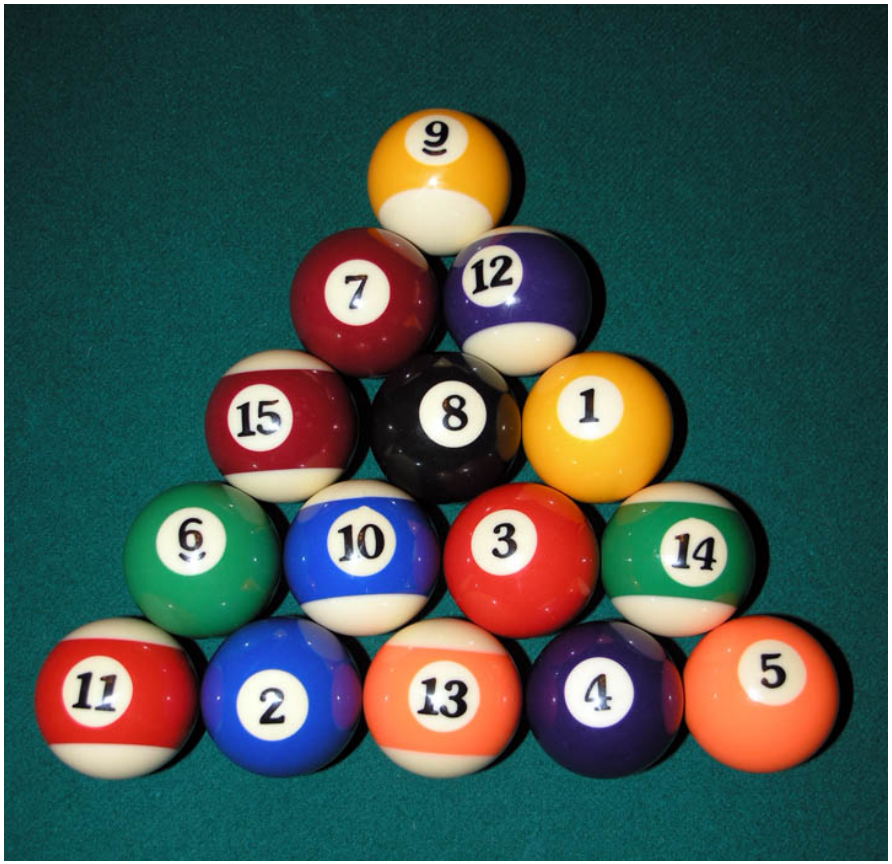
- Comparer un objet avec plusieurs templates
- Choisir le template avec le meilleur score
- Technique
  - Extraire de l'objet depuis l'image
  - Convertir l'objet en niveaux de gris
  - Extraire les caractéristiques et les points d'interêts de l'objet
  - Pour chaque template i:
    - Extraire les caractéristiques et les points d'interêts du template i
    - Calculer la correspondance entre le template i et l'objet
  - Choisir le template j tel que j = Min(score(i))

Template matching

- Classes et méthodes OpenCV
  - Extraction de caractéristiques : FeatureDetector.*create(FeatureDetector.**XXX)***
  - Extraction de point d'intérêt : DescriptorExtractor.*create(DescriptorExtractor.**XXX)***
  - Algorithmes :
    - ORB (Oriented FAST Rotated BRIEF)
    - BRISK
    - FAST
    - SIFT
    - SURF

Exercice : trouver la balle 3

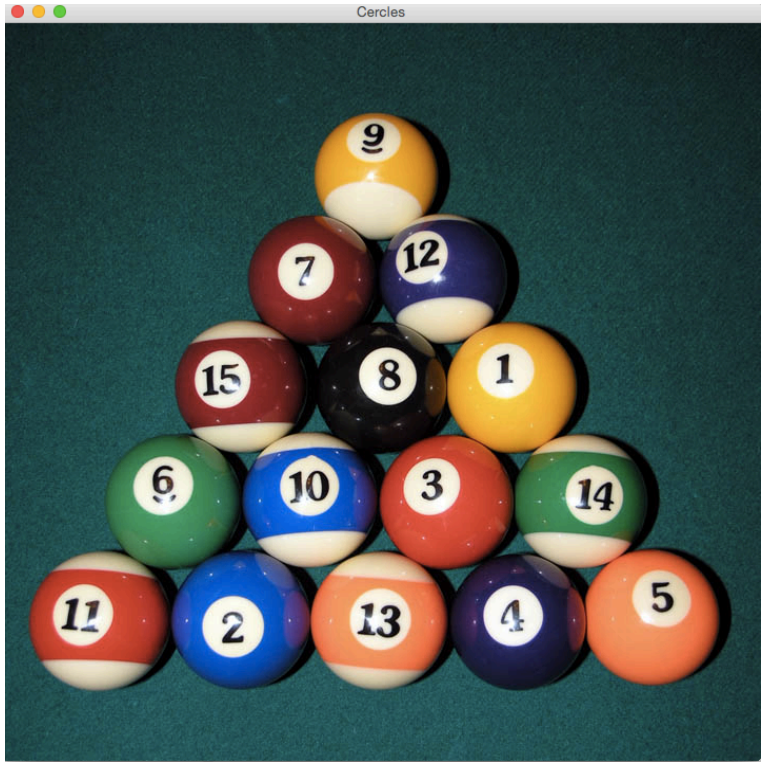
- Dans l'image suivante on vous demande d'écrire un programme pour reconnaître la balle numéroté 3
- Les images sont disponibles sous arche dans l'archive balls.zip

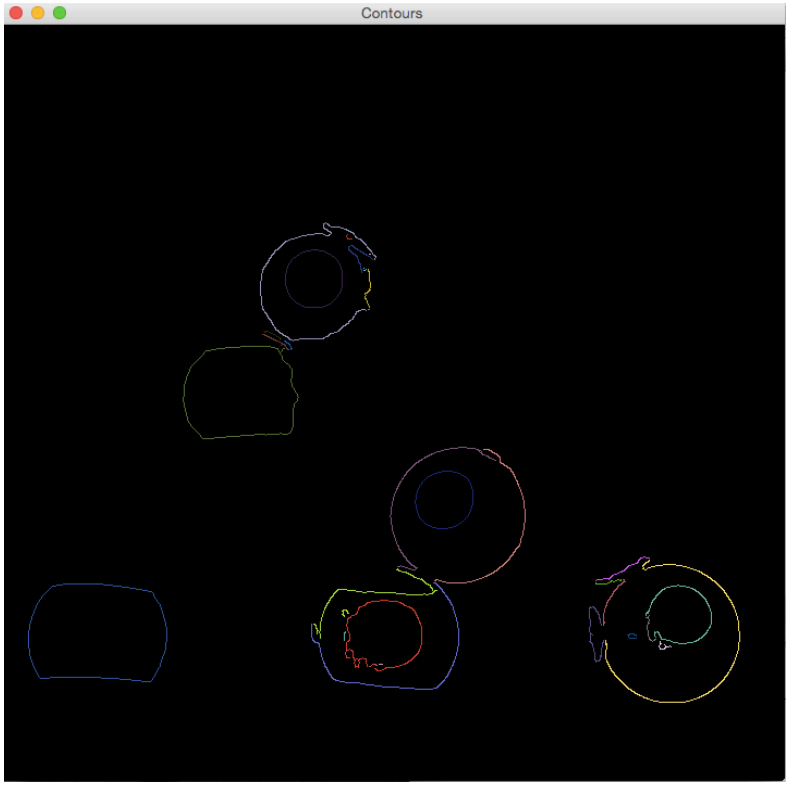


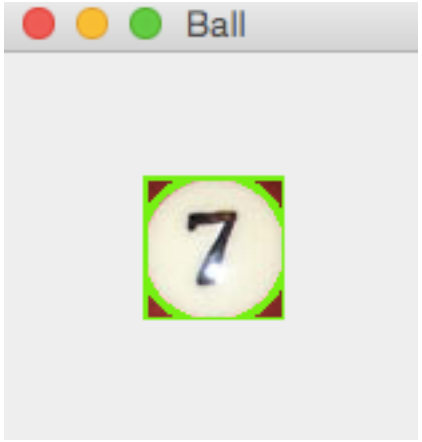
Solution : extraire les balles rouges

```
MatOfPoint2f matOfPoint2f = new MatOfPoint2f();
float[] radius = new float[1];
Point center = new Point();
for (int c=0; c < contours.size();c++) {
    MatOfPoint contour = contours.get(c);
    double contourArea = Imgproc.contourArea(contour);
    matOfPoint2f.fromList(contour.toList());
    Imgproc.minEnclosingCircle(matOfPoint2f, center, radius);
    if ((contourArea/(Math.PI*radius[0]*radius[0])) >=0.8) {
        Core.circle(m, center, (int)radius[0], new Scalar(0, 255, 0), 2);
        Rect rect = Imgproc.boundingRect(contour);
        Core.rectangle(m, new Point(rect.x,rect.y),
                new Point(rect.x+rect.width,rect.y+rect.height),
                new Scalar (0, 255, 0), 2);
        Mat tmp = m.submat(rect.y,rect.y+rect.height,rect.x,rect.x+rect.width);
        Mat ball = Mat.zeros(tmp.size(),tmp.type());
        tmp.copyTo(ball);
        utils.ImShow("Ball",ball);
    }
}
```

Résultat







La mise à l'échelle

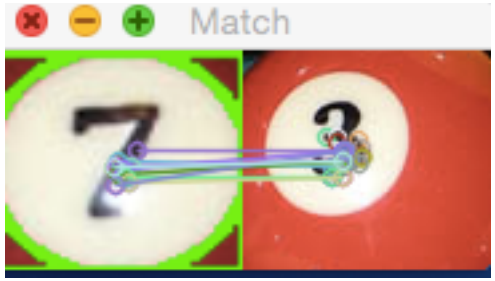
```
Mat sroadSign = Highgui.imread(objectfile);
Mat sObject = new Mat();
Imgproc.resize(object, sObject, sroadSign.size());
Mat grayObject = new Mat(sObject.rows(), sObject.cols(), sObject.type());
Imgproc.cvtColor(sObject, grayObject, Imgproc.COLOR_BGRA2GRAY);
Core.normalize(grayObject, grayObject, 0, 255, Core.NORM_MINMAX);

Mat graySign = new Mat(sroadSign.rows(), sroadSign.cols(), sroadSign.type());
Imgproc.cvtColor(sroadSign, graySign, Imgproc.COLOR_BGRA2GRAY);
Core.normalize(graySign, graySign, 0, 255, Core.NORM_MINMAX);
```

Extraction des caractéristiques

```
// Extraction des descripteurs et keypoints
FeatureDetector orbDetector = FeatureDetector.create(FeatureDetector.ORB);
DescriptorExtractor orbExtractor = DescriptorExtractor.create(DescriptorExtractor.OR

MatOfKeyPoint objectKeypoints = new MatOfKeyPoint();
orbDetector.detect(grayObject, objectKeypoints);

MatOfKeyPoint signKeypoints = new MatOfKeyPoint();
orbDetector.detect(graySign, signKeypoints);

Mat objectDescriptor = new Mat(object.rows(), object.cols(), object.type());
orbExtractor.compute(grayObject, objectKeypoints, objectDescriptor);

Mat signDescriptor = new Mat(sroadSign.rows(), sroadSign.cols(), sroadSign.type());
orbExtractor.compute(graySign, signKeypoints, signDescriptor);
```

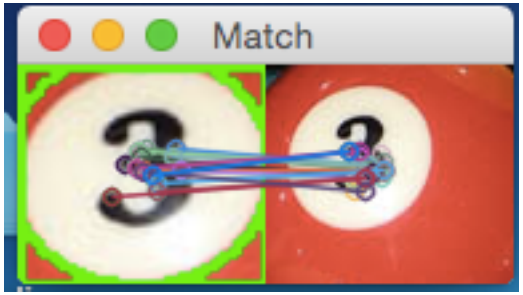
Le matching

```
// Faire le matching
MatOfDMatch matchs = new MatOfDMatch();
DescriptorMatcher matcher = DescriptorMatcher.create(DescriptorMatcher.BRUTEFORCE);
matcher.match(objectDescriptor, signDescriptor, matchs);
System.out.println(matchs.dump());
Mat matchedImage = new Mat(sroadSign.rows(), sroadSign.cols()*2, sroadSign.type());
Features2d.drawMatches(sObject, objectKeypoints, sroadSign, signKeypoints,
        matchs, matchedImage);
```



[0, 14, 0, 352.20734;
1, 10, 0, 373.21979;
2, 16, 0, 313.84232;
3, 9, 0, 404.18683;
4, 14, 0, 342.62369;
5, 16, 0, 384.25253;
6, 12, 0, 370.20264;
7, 13, 0, 453.70917;
8, 5, 0, 407.3598;
9, 16, 0, 329.95453]

Query index, train index,
Image index, distance



[0, 6, 0, 412.87772;
1, 5, 0, 499.45572;
2, 5, 0, 468.49118;
3, 9, 0, 393.14502;
4, 5, 0, 482.05084;
5, 8, 0, 392.08801;
6, 10, 0, 434.13361;
7, 7, 0, 489.32095;
8, 9, 0, 435.43311;
9, 0, 0, 488.32263;
10, 7, 0, 470.31479;
11, 9, 0, 474.04535;
12, 9, 0, 307.0863;
13, 1, 0, 516.54718;
14, 13, 0, 432.784;
15, 0, 0, 486.48227]