Robotics: Principles and Practice

Module 5: Robot Vision

Lecture 6: Image analysis; feature extraction

David Vernon
Carnegie Mellon University Africa

www.vernon.eu

Image Analysis

- Reminder: automatically extracting useful information from an image
- We can also classify the types of analysis we wish to perform according to function
  - Inspection

    Is the visual appearance of objects as it should be?
  - Location

    requires the specification of both position and orientation in either 2D or 3D
    - image frame of reference (pixels)
    - real world frame of reference (e.g. millimetres) … calibration required
  - Identification of object type

Image Analysis

- Find objects within the image and identify or classify those objects
- Central assumption:
  - the image depicts one or more objects
  - each object belongs to one of several distinct and exclusive predetermined classes
- We know what objects exist and an object can only have one particular type or label

Image Analysis

Three components of this type of pattern recognition process:

- an object isolation module
    - Produces a representation of the object (segmentation)
- a feature extraction module
    - Abstracts one or more characteristic features and produces a feature vector
    - Selection of features is crucial
- a classification module
    - The feature vector is used by the classification module to identify and label each object

Features

Features should be

- Independent
  a change in one feature should not change significantly the value of another feature

- Discriminatory
  Each feature should have a significantly different value for each different object

- Reliable
  Features should have the same value for all objects in the same class/group

- The computational complexity of pattern recognition increases rapidly as the number of features increases

- Hence it is desirable to use the fewest number of features possible, while ensuring a minimal number of errors

Features

Simple features

- Many features are either based on the size of the object or on its shape
- Area of the object
  - simply the number of pixels comprising the object multiplies by the area of a single pixel (frequently assumed to be a single unit)
  - can also be computed from the boundary contour (see later)

Features

Simple features

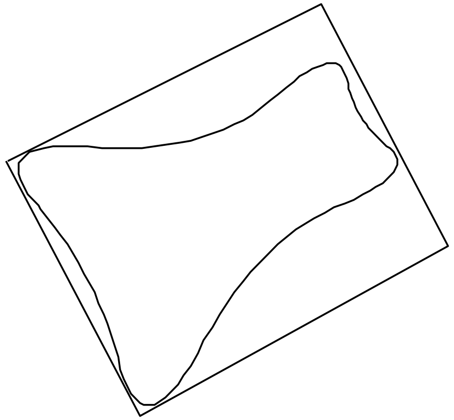
- The length and the width of an object describe
- If its orientation is not known
  - we may have to first compute its orientation before evaluating the minimum and maximum extent of its boundary
- These measures should always be made with respect to some rotation-invariant datum line in the object, *e.g.*, its major or minor axis

Features

Simple features

Minimum Bounding Rectangle

- The smallest rectangle which can completely enclose the object
- The main axis of this rectangle is the principle axis of the object
- Hence, the dimensions of the minimum bounding rectangle correspond to the features of length and width

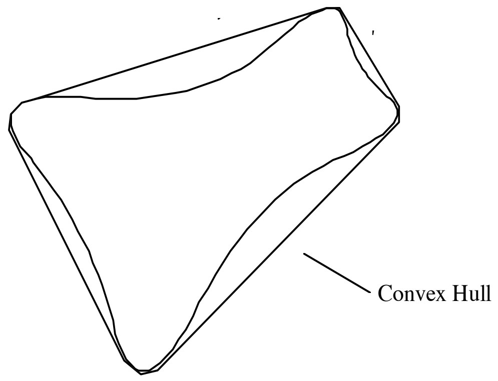


Features

Simple features

Convex Hull

– The smallest convex boundary which can completely enclose the object



Features

Simple features

- The distance around the perimeter of the object
- Depending on how the object is represented, it can be quite trivial to compute the length of the perimeter
- This makes it an attractive feature for industrial vision applications

Features

Simple features

Rectangularity

$$R = \frac{A_{\text{object}}}{A_{\text{min. bound. rectangle}}}$$

- Minimum value of 1 for a perfect rectangular shape
- Tends toward zero for thin curvy objects

Rectangularity : Aspect Ratio

$$\text{Aspect Ratio} = \frac{W_{\text{min. bounding rectangle}}}{L_{\text{min. bounding rectangle}}}$$

Features

Simple features

Elongatedness

$$\frac{A_{object}}{(2d)^2}$$

- Ratio of object area to square of its “thickness” d
- “Thickness” can be estimated by
  - the number of iterations of an erosion operator to remove the object
  - the number of iterations of a thinning operator

Features

Simple features

Circularity

$$C = \frac{A_{object}}{P^2_{object}}$$

- Maximum value for discs
- Tends toward zero for irregular shapes with ragged boundaries

Features

Moment features

– Method of Moments

– The standard two-dimensional moments m_{uv} of an image intensity function $g(x, y)$

$$m_{uv} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} g(x,y)x^u y^u dxdy \qquad u, v = 0, 1, 2, 3\ldots$$

$$m_{uv} = \sum_x \sum_y g(x,y)x^u y^u \qquad u, v = 0, 1, 2, 3\ldots$$

summed over the entire sub-image within which the shape lies

Features

Moment features

- Method of Moments

- However, these moments will vary for a given shape depending on where the shape is positioned, *i.e.*, they are position-dependent

- Instead, use the central moments

$$\mu_{uv} = \sum_x \sum_y g(x,y)(x-\bar{x})^u (y-\bar{y})^v$$

where $\bar{x} = \frac{m_{10}}{m_{00}}$ and $\bar{y} = \frac{m_{01}}{m_{00}}$
i.e. the coordinates of the centroid of the shape

- This renders the moments position invariant

Features

Moment features

- Method of Moments

- Assuming that the intensity function $g(x, y)$ has a value of 1 everywhere in the object (*i.e.* we are dealing with a simple segmented binary image), the computation of m_{00} is simply a summation yielding the total number of pixels within the shape

- If we also assume that a pixel is one unit area, then m_{00} is equivalent to the area of the shape

- Similarly, m_{10} is effectively the summation of all the x co-ordinates of pixels in the shape and m_{01} is the summation of all the y co-ordinates of pixels in the shape; hence

  m_{10}/m_{00} is the average x co-ordinate

  m_{01}/m_{00} is the average y co-ordinate

  i.e. the co-ordinates of the centroid.

Features

Moment features

Central Moments

$$\mu_{00} = m_{00}$$
$$\mu_{10} = 0$$
$$\mu_{01} = 0$$
$$\mu_{20} = m_{20} - \bar{x}m_{10}$$
$$\mu_{02} = m_{02} - \bar{y}m_{01}$$
$$\mu_{11} = m_{11} - \bar{y}m_{10}$$
$$\mu_{30} = m_{30} - 3\bar{x}m_{20} + 2\bar{x}^2 m_{10}$$
$$\mu_{03} = m_{03} - 3\bar{y}m_{02} + 2\bar{y}^2 m_{01}$$
$$\mu_{12} = m_{12} - 2\bar{y}m_{11} - \bar{x}m_{01} + 2\bar{y}^2 m_{10}$$
$$\mu_{21} = m_{21} - 2\bar{x}m_{11} - \bar{y}m_{20} + 2\bar{x}^2 m_{01}$$

Features

Moment features

Normalized Central Moments

$$\eta_{ij} = \frac{\mu_{ij}}{(\mu_{00})^k}$$

where $k = ((i + j) / 2) + 1 \quad | \, i + j >= 2$

Features

Moment features

Moment Invariants

- Linear combinations of normalized central moments
- More frequently used for shape description because they generate values which are invariant with position, orientation and scale changes
- Also known as Hu moment invariants or Hu moments

Features

Moment features

Moment Invariants

$$\phi_1 = \eta_{20} + \eta_{02}$$

$$\phi_2 = \left(\eta_{20} + \eta_{02}\right)^2 + 4\eta_{11}^{\;2}$$

$$\phi_3 = \left(\eta_{30} + 3\eta_{12}\right)^2 + \left(3\eta_{21} + \eta_{03}\right)^2$$

$$\phi_4 = \left(\eta_{30} + \eta_{12}\right)^2 + \left(\eta_{21} + \eta_{03}\right)^2$$

$$\phi_5 = \left(\eta_{30} + 3\eta_{12}\right)\left(\eta_{30} + \eta_{12}\right)\left\{\left(\eta_{30} + \eta_{12}\right)^2 - 3\left(\eta_{21} + \eta_{03}\right)^2\right\} + \left(3\eta_{21} - \eta_{03}\right)\left(\eta_{21} + \eta_{03}\right)\left\{3\left(\eta_{30} + \eta_{12}\right)^2 - \left(\eta_{21} + \eta_{03}\right)^2\right\}$$

$$\phi_6 = \left(\eta_{20} + 3\eta_{12}\right)\left\{\left(\eta_{30} + \eta_{12}\right) - \left(\eta_{21} + \eta_{03}\right)^2\right\} + 4\eta_{11}\left(\eta_{30} - \eta_{12}\right)\left(\eta_{21} + \eta_{03}\right)$$

$$\phi_7 = \left(3\eta_{21} - \eta_{03}\right)\left(\eta_{30} + \eta_{12}\right)\left\{\left(\eta_{30} + \eta_{12}\right)^2 - 3\left(\eta_{21} + \eta_{03}\right)^2\right\} - \left(3\eta_{12} - \eta_{30}\right)\left(\eta_{21} + \eta_{03}\right)\left\{3\left(\eta_{30} + \eta_{12}\right)^2 - \left(\eta_{21} + \eta_{03}\right)^2\right\}$$

Reading

D. Vernon, Machine Vision, Prentice-Hall International

Section 6.3 Decision-theoretic Techniques

Demo

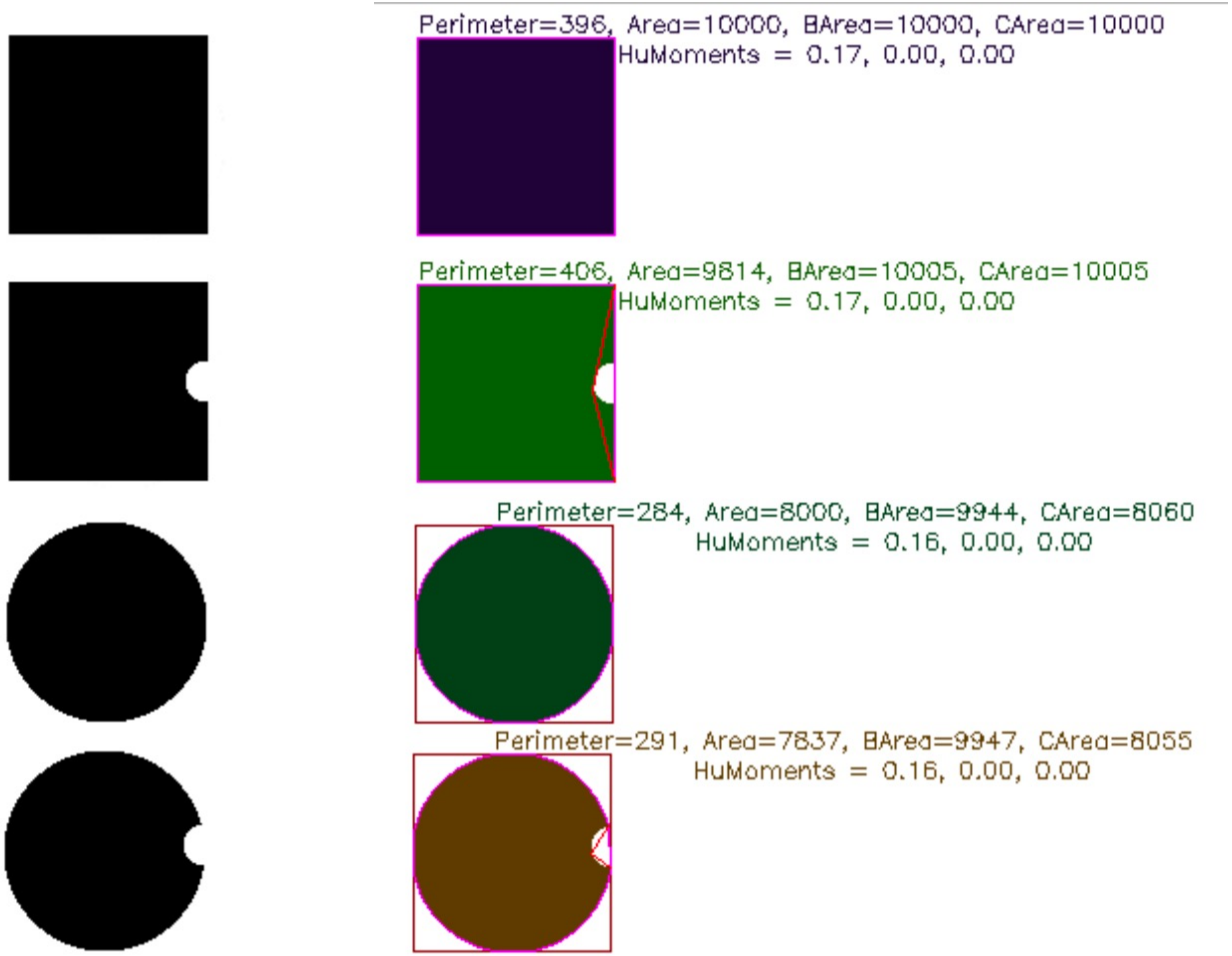
The following code is taken from the featureExtraction example application

See:

```
featureExtraction.h
featureExtractionImplementation.cpp
featureExtractionApplication.cpp
```

To run the example:

Ubuntu 16.04: `rosrun module5 featureExtraction`


Perimeter=396, Area=10000, BArea=10000, CArea=10000
HuMoments = 0.17, 0.00, 0.00
Perimeter=406, Area=9814, BArea=10005, CArea=10005
HuMoments = 0.17, 0.00, 0.00
Perimeter=284, Area=8000, BArea=9944, CArea=8060
HuMoments = 0.16, 0.00, 0.00
Perimeter=291, Area=7837, BArea=9947, CArea=8055
HuMoments = 0.16, 0.00, 0.00

```
/*
  Example use of openCV to perform 2D feature extraction
  ------------------------------------------------------
  Implementation file

  David Vernon
  18 June 2017
*/

#include "featureExtraction.h"

void featureExtraction(char *filename, FILE *fp_out) {

   char inputWindowName[MAX_STRING_LENGTH]              = "Input Image";
   char outputWindowName[MAX_STRING_LENGTH]             = "Contour Image";

   Mat inputImage;

   namedWindow(inputWindowName,   CV_WINDOW_AUTOSIZE);
   namedWindow(outputWindowName,  CV_WINDOW_AUTOSIZE);

   inputImage = imread(filename, CV_LOAD_IMAGE_COLOR); // Read the file

   if (!inputImage.data) {                            // Check for invalid input
      printf("Error: failed to read image %s\n",filename);
      prompt_and_exit(-1);
   }

   printf("Press any key to continue ...\n");

   fprintf(fp_out,"%s \n",filename);  // file write added by David Vernon
```

```
/*
 * The following is based on code provided as part of "A Practical Introduction to Computer Vision with OpenCV"
 * by Kenneth Dawson-Howe © Wiley & Sons Inc. 2014.  All rights reserved.
 */

 /* convert the input image to a binary image */
  Mat gray;
  Mat binary;

  cvtColor(inputImage, gray, CV_BGR2GRAY);
  //threshold(gray,binary,128,255,THRESH_BINARY_INV);
  threshold(gray,binary,128, 255,THRESH_BINARY_INV  | THRESH_OTSU); // David Vernon: substituted in automatic threshold selection

  /* extract the contours of the objects in the binary image */
  vector<vector<Point>> contours;
  vector<Vec4i>         hierarchy;

  /* David Vernon: see http://docs.opencv.org/2.4.10/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html#findcontours */
  findContours(binary,contours,hierarchy,CV_RETR_TREE,CV_CHAIN_APPROX_NONE);

  /* extract features from the contours */
  Mat contours_image = Mat::zeros(binary.size(), CV_8UC3);
  contours_image = Scalar(255,255,255);

  //binary.copyTo(contours_image,binary);

   /* Prepare to do some processing on all contours (objects and holes!) by declaring appropriate data-structures */
   vector<RotatedRect>   min_bounding_rectangle(contours.size());   // bounding rectangles
   vector<vector<Point>> hulls(contours.size());                    // convex hulls
   vector<vector<int>>   hull_indices(contours.size());             // indices of convex hulls
   vector<vector<Vec4i>> convexity_defects(contours.size());        // convex cavities
   vector<Moments>       contour_moments(contours.size());          // moments
```

Each contour is a vector of points

The hierarchical organization of contours, i.e. contours inside contours, inside contours, ... is captured in this vector

This function builds the list of contours and determines their hierarchical structure

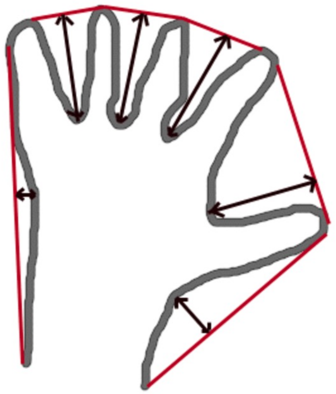
Minimum bounding rectangles: perfect for capturing the required information about bricks

```
for (int contour_number=0; (contour_number<(int)contours.size()); contour_number++) {
   if (contours[contour_number].size() > 10) { // only consider contours of appreciable length

      /* David Vernon: see http://docs.opencv.org/2.4.10/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html#boundingrec
      min_bounding_rectangle[contour_number] = minAreaRect(contours[contour_number]);

      /* David Vernon: see http://docs.opencv.org/2.4.10/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html#convexhull
      convexHull(contours[contour_number], hulls[contour_number]);
      convexHull(contours[contour_number], hull_indices[contour_number]);

      /* David Vernon: see http://docs.opencv.org/2.4.10/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html#convexityde
      convexityDefects(contours[contour_number], hull_indices[contour_number], convexity_defects[contour_number]);

      /* David Vernon: see http://docs.opencv.org/2.4.10/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html#moments*/
      contour_moments[contour_number] = moments( contours[contour_number] );
   }
}
```

Minimum bounding rectangles: perfect for capturing the required information about bricks

```
struct CvConvexityDefect
{
   CvPoint* start; // point of the contour where the defect begins
   CvPoint* end; // point of the contour where the defect ends
   CvPoint* depth_point; // the farthest from the convex hull point within the defect
   float depth; // distance between the farthest point and the convex hull
};
```

The figure below displays convexity defects of a hand contour:



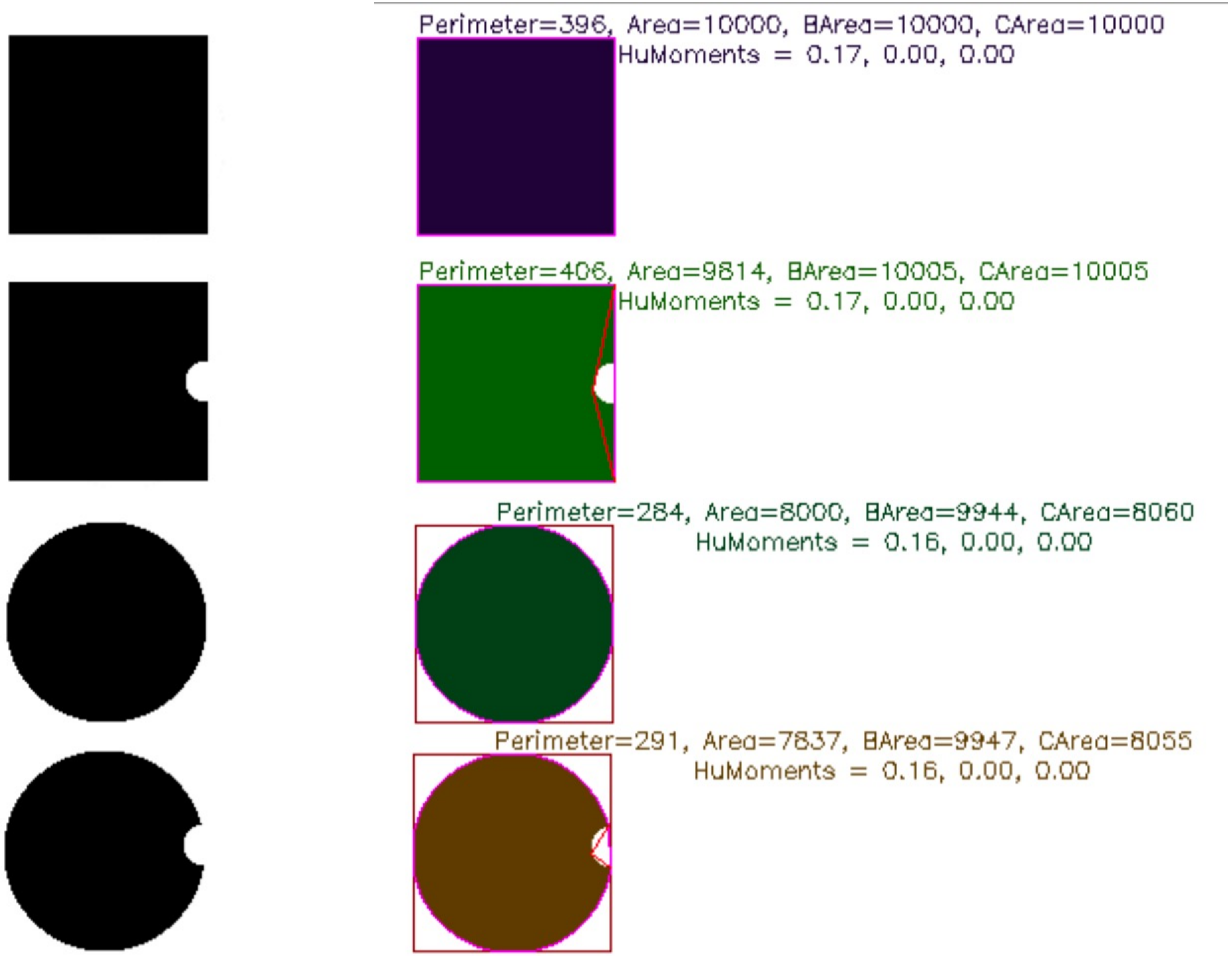
http://docs.opencv.org/2.4.10/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html#convexitydefects

```
/* for all contours */
for (int contour_number=0; (contour_number>=0); contour_number=hierarchy[contour_number][0]) {

   /* only consider contours of appreciable length */
   if (contours[contour_number].size() > 10) {
      Scalar colour(rand()&0x7F, rand()&0x7F, rand()&0x7F );                                   // generate a random colour
      drawContours(contours_image, contours, contour_number, colour, CV_FILLED, 8, hierarchy ); // draw the contour

      char output[500];

      // David Vernon: Ken Dawson-Howe adjusts area as it seems to be underestimated by half the number of pixels on the perimeter
      double area = contourArea(contours[contour_number]) + contours[contour_number].size()/2 + 1;

      // Process any holes (removing the area from the area of the enclosing contour)
      for (int hole_number=hierarchy[contour_number][2]; (hole_number>=0); hole_number=hierarchy[hole_number][0]) {

         // David Vernon: Ken Dawson-Howe adjusts area as it seems to be underestimated by half the number of pixels on the perimeter
         area -= (contourArea(contours[hole_number]) - contours[hole_number].size()/2 + 1);

         Scalar colour( rand()&0x7F, rand()&0x7F, rand()&0x7F );
         drawContours( contours_image, contours, hole_number, colour, CV_FILLED, 8, hierarchy );

         sprintf(output,"Area=%.0f", contourArea(contours[hole_number]) -contours[hole_number].size()/2+1);

         /* write to file added by David Vernon */
         fprintf(fp_out,"Object %d, Hole %d: Area = %.0f\n",
                 contour_number, hole_number, contourArea(contours[hole_number]) - contours[hole_number].size()/2 + 1);

         Point location( contours[hole_number][0].x + 20, contours[hole_number][0].y + 5 );
         putText( contours_image, output, location, FONT_HERSHEY_SIMPLEX, 0.4, colour );
      }
```

Minimum bounding rectangles: perfect for capturing the required information about bricks

```
/* Draw the minimum bounding rectangle */
Point2f bounding_rect_points[4];
min_bounding_rectangle[contour_number].points(bounding_rect_points);
line(contours_image, bounding_rect_points[0], bounding_rect_points[1], Scalar(0, 0, 127));
line(contours_image, bounding_rect_points[1], bounding_rect_points[2], Scalar(0, 0, 127));
line(contours_image, bounding_rect_points[2], bounding_rect_points[3], Scalar(0, 0, 127));
line(contours_image, bounding_rect_points[3], bounding_rect_points[0], Scalar(0, 0, 127));

float bounding_rectangle_area = min_bounding_rectangle[contour_number].size.area();

/* Draw the convex hull */
drawContours(contours_image, hulls, contour_number, Scalar(255,0,255) );  // purple

/* Highlight any convexities */
int largest_convexity_depth=0;

for (int convexity_index=0; convexity_index < (int)convexity_defects[contour_number].size(); convexity_index++) {
   if (convexity_defects[contour_number][convexity_index][3] > largest_convexity_depth)
      largest_convexity_depth = convexity_defects[contour_number][convexity_index][3];

   if (convexity_defects[contour_number][convexity_index][3] > 256*2) {
      line( contours_image, contours[contour_number][convexity_defects[contour_number][convexity_index][0]],
                            contours[contour_number][convexity_defects[contour_number][convexity_index][2]], Scalar(0,0, 255));
      line( contours_image, contours[contour_number][convexity_defects[contour_number][convexity_index][1]],
                            contours[contour_number][convexity_defects[contour_number][convexity_index][2]], Scalar(0,0, 255));
   }
}
```

```
        //sprintf(output,"Perimeter=%d, Area=%.0f, BArea=%.0f, CArea=%.0f", contours[contour_number].size(),area,min_bounding_rectangle[
        /* David Vernon: alternative as area seems to be underestimated by half the number of pixels on the perimeter */
        sprintf(output,"Perimeter=%d, Area=%.0f, BArea=%.0f, CArea=%.0f", contours[contour_number].size(),
                                                                  area,
                                                                  min_bounding_rectangle[contour_number].size.area() + contours[
                                                                  contourArea(hulls[contour_number])+ contours[contour_number].s

        /* file write added by David Vernon */
        /* David Vernon: area seems to be underestimated by half the number of pixels on the perimeter */
        fprintf(fp_out,"Object %d: perimeter = %d, object area = %.0f, bounding rectangle area = %.0f, convex hull area = %.0f \n",
                contour_number,
                contours[contour_number].size(),
                area,
                min_bounding_rectangle[contour_number].size.area() + contours[contour_number].size()/2 + 1,
                contourArea(hulls[contour_number]) + contours[contour_number].size()/2 + 1);

        Point location( contours[contour_number][0].x, contours[contour_number][0].y-3 );
        putText(contours_image, output, location, FONT_HERSHEY_SIMPLEX, 0.4, colour );

        /* David Vernon: see http://docs.opencv.org/2.4.10/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html#humoments 
        double hu_moments[7];
        HuMoments(contour_moments[contour_number], hu_moments );

        sprintf(output,"HuMoments = %.2f, %.2f, %.2f", hu_moments[0],hu_moments[1],hu_moments[2]);
        Point location2( contours[contour_number][0].x+100, contours[contour_number][0].y-3+15 );
        putText(contours_image, output, location2, FONT_HERSHEY_SIMPLEX, 0.4, colour );

        /* filewrite added by David Vernon */
        fprintf(fp_out,"Object %d: HuMoments = %.2f, %.2f, %.2f \n\n", contour_number, hu_moments[0],hu_moments[1],hu_moments[2]);
     }
     fprintf(fp_out,"\n"); //file write added by David Vernon
  }

  imshow(inputWindowName,  inputImage );
  imshow(outputWindowName, contours_image);

  do{
     waitKey(30);
  } while (!_kbhit());

  getchar(); // flush the buffer from the keyboard hit

  destroyWindow(inputWindowName);
  destroyWindow(outputWindowName);
}
```


Perimeter=396, Area=10000, BArea=10000, CArea=10000
HuMoments = 0.17, 0.00, 0.00
Perimeter=406, Area=9814, BArea=10005, CArea=10005
HuMoments = 0.17, 0.00, 0.00
Perimeter=284, Area=8000, BArea=9944, CArea=8060
HuMoments = 0.16, 0.00, 0.00
Perimeter=291, Area=7837, BArea=9947, CArea=8055
HuMoments = 0.16, 0.00, 0.00


0
2
4
6
8
Perimeter=80, Area=409, BArea=692, CArea=596
Area=172
HuMoments = 0.17, 0.00, 0.00
Perimeter=111, Area=347, BArea=626, CArea=584
HuMoments = 0.42, 0.07, 0.00
Perimeter=86, Area=350, BArea=636, CArea=488
Area=59
HuMoments = 0.20, 0.01, 0.00
Perimeter=101, Area=395, BArea=671, CArea=587
HuMoments = 0.23, 0.01, 0.00
Area=77
Perimeter=85, Area=443, BArea=694, CArea=615
Area=45
Area=59
HuMoments = 0.20, 0.01, 0.00
Perimeter=101, Area=275, BArea=573, CArea=480
HuMoments = 0.46, 0.13, 0.01
Perimeter=121, Area=353, BArea=650, CArea=592
HuMoments = 0.40, 0.06, 0.00
Perimeter=121, Area=333, BArea=631, CArea=565
HuMoments = 0.42, 0.06, 0.00
Perimeter=94, Area=269, BArea=599, CArea=442
HuMoments = 0.46, 0.09, 0.04
Perimeter=101, Area=394, BArea=640, CArea=575
Area=70
HuMoments = 0.23, 0.01, 0.00