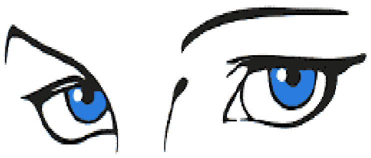
PROFI



Project number:	FP6-511572
Project acronym:	PROFI
Project Title:	Perceptually-Relevant Retrieval of Figurative Images

Deliverable No: D8.1
Title: Algorithms to calculate features from components.
Title: ***Objective 1:*** To identify the optimum feature set to describe the components and their inter-relationships

Short description:

During this stage, the set of features to describe the perceptually segments components will be identified. Consequently, each component will be represented by a point in the multi-dimensional space formed by the chosen features. It is expected that similar components will be positioned in close proximity in this space. Such discriminatory capabilities can be maximised by manipulating the feature space in order to maximise the ratio of between-class to within-class variances (Webb, 2002). This work will investigate different methods to achieve this objective including Fisher linear discriminants (Fisher, 1936).

Due month:	15
Delivery month:	15
Partners owning:	UoY
Partners contributed:	UoY
Classification:	Restricted

Information Society Technologies

Project funded by the European Community under the "Information Society Technologies" Programme

PROFI

Perceptually-Relevant Retrieval of Figurative Images

Deliverable 8.1

Internal Project Report

Title: Algorithms to calculate features from components.

Authors: Dr Victoria Hodge, Prof. John Eakins and Prof. Jim Austin.

Advanced Computer Architectures Group,
Department of Computer Science,
University of York
York
UK

31st March 2006.

Work package 8

Objective 1:

The following is the original objective of Work Package 8.1 as given in the project proposal document:

> *To identify the feature set to describe the components and their inter-relationships.*
>
> *During this stage, the set of features to describe the perceptually segments components will be identified. Consequently, each component will be represented by a point in the multi-dimensional space formed by the chosen features. It is expected that similar components will be positioned in close proximity in this space. Such discriminatory capabilities can be maximised by manipulating the feature space in order to maximise the ratio of between-class to within-class variances (Webb, 2002). This work will investigate different methods to achieve this objective including Fisher linear discriminants (Fisher, 1936).*

Deliverables: D8.1 Algorithms to calculate features from components.

Overview

The aim of this work has been to characterise components of the trademarks (produced from earlier processing) into a feature vector. The main aim has been to achieve this optimally; however, this requires the system to be in place. In practice the features generated from the system provided here will be selected by an optimisation method. The software package developed in the work provides a large set of measures that can be used in many applications, not just the trademark data of interest here.

The report starts by explaining the motivation for the selection of features, following this the basic methods are described. Appendix A describes the call structures for the software.

Approach

This work takes its cue from the features elicited by Alwis [A99], Chan & King [CK99] and Antani et al. [A03] and the ARTISAN [ERE03] and QBIC [N93, F95] systems. All features implemented in the deliverable are component-based (not whole-image based [ERE03]) and are intended for use with boundary points only as the closed shape identification algorithm (produced in work package 2.3) which uses the features, outputs the closed paths it identifies within each image as a set of boundary points. Some feature calculators (most notably Hu Moments, triangularity and ellipticity) may also be used with region points (the full set of fill area points). Note that, if there are gaps in between the end-points of the segments included in the list of boundary points then these are **NOT** filled by the C++ methods and remain during calculations for the Features class. The implementation of the closed shape identification algorithm outputs paths with gaps and with gaps filled so we recommend using the latter for all feature calculations.

The features have not yet been evaluated to identify the optimal set as this will vary with work package. Selecting the best subsets of features from a feature set is a combinatorial optimisation problem and, as noted in [ERE03] the recall and precision results from different sets of features are very similar. The authors compared retrieval results for different sets of 6 or 7 features and found the recall and precision results very similar for the different sets. Therefore, the work will choose a small number of subsets of features each containing a broad cross-section of features and evaluate these. A subset of the features will be used for assessing the similarity of closed shapes and for assessing the global goodness (perceptual relevance) of closed shapes within the closed shape identification algorithm (Work Package 2.3). Assessing the similarity of closed paths output by the closed shapes identification algorithm will need human visual inspection to identify which set of features produce similar results on similar closed paths and different results on different closed shapes. To identify the features that allow the perceptually relevant closed paths output from the closed shape identification algorithm to be sorted from the perceptually irrelevant closed paths, the perceptual breakdowns from the human perception experiments may be used. The experiments were performed in Work Package 3.1 and are described in [HEA06] & [HHEA06]. The experiments produced a large body of ground truth representing the image components of 84 images that are perpetually relevant to human subjects. By calculating feature values for each closed shape identified by the closed shape identification algorithm and classifying each shape with one of two classes, relevant or irrelevant, a supervised feature selector may be used to select the feature set. This process also allows any shapes that are missed, i.e., human breakdowns not captured by the closed shape identification algorithm to be pinpointed.

The original proposal aimed to use Fisher linear discriminant analysis to optimize the features. Unfortunately, Fisher linear discriminant may not be used to derive the optimal set of features as there are no component-based (closed path or region-based) similarity data available. This aspect of the objective will be integrated within the future work of work packages 2, 3 and 8 when the system parts are integrated to produce the overall system.

Alwis [A99] has produced a trademark retrieval system, with many similarities to the work here, so the features he used will be particularly relevant for the work here. He uses circularity, aspect ratio, stuffedness, right angled-ness, sharpness, complexity, directness and straightness. The first three of these features have been implemented as these are observed to perform the best.

Chan & King [CK99] propose a method for feature weight assignment in a trademark system using Genetic algorithms to determine the weights for the set of features. Therefore, the features used by Chan & King will be useful to the work here. They use invariant moments, Euler number, eccentricity, and circularity in their evaluations. All of these features bar the Euler number have been implemented as the closed paths output by the closed path identification algorithm do not contain ‘holes’.

Antani et al. [A03] have implemented a modular prototype system for content-based image retrieval of vertebral X-rays. The shapes within the x-ray are segmented using active contour and human assisted segmentation. The authors then compare various features to assess each feature's effectiveness in retrieving vertebral shapes. The authors' desiderata for a suitable feature set are: invariance, uniqueness, stability, efficiency, ease of implementation, computation of shape properties, geometric invariance, compact representation, fast matching speed, and high quality image retrieval. The authors compare polygon approximation, scale space filtering, Fourier descriptors and invariant moments coupled with a set of geometric properties: they use centre of gravity (centroid), area, perimeter, convex perimeter, major axis length and angle, minor axis length and angle, compactness, roughness and elongation. The authors find that for their retrieval aim of matching very similar shapes, none of the evaluated feature has sufficiently high performance. However, they note that: invariant moments coupled with a set of geometric properties work well in separating shapes that are significantly different. All of these geometric properties have been implemented in this work except the three: major axis length and angle, minor axis length and angle and elongation as all three may be derived from the height, width and angle of the minimum area bounding box.

ARTISAN [ERE03] is "regarded as one of the most comprehensive trademark retrieval systems in the current literature" [JNT06]. It uses the shape's aspect ratio, circularity, convexity, the 8 Fourier descriptors, the shape's area and the three 'natural' shape measures defined by Rosin [R00]: rectangularity, triangularity and ellipticity for trademark retrieval. The PROFI planned system will match trademark at the component level and ARTISAN uses component-based matching so the features used by ARTISAN should be relevant for our system. The authors identified that component-based matching outperforms whole image matching by a significant margin (70% recall for component-based matching versus 53% recall for the best performing whole image matching feature: angular-radial transform). Therefore, all of these component-based features have been implemented except Rosin's rectangularity as stuffedness (which is implemented) is very similar.

The IBM QBIC [N93, F95] system is one of the most ubiquitous image retrieval systems developed and has been used widely so the features used for matching should be relevant to our developmental system. The shape features used in QBIC consist of shape area, circularity, eccentricity, major axis orientation and a set of algebraic moment invariants. For the database images, these shape features are extracted for all the object contours. All of these features have been implemented bar the major axis orientations as this is, in effect, covered by the minimum area bounding box.

Features

Various feature calculation functions have been implemented in C++. These are in the files:

- **Features.h –** Features class header.
- **Features.cpp–** Features class source.
- **2dch.h –** File containing the convex hull code (called from Features.cpp).

The C++ code has been tested as far as possible on a range of closed paths output by the closed path identification algorithm produced in work package 2.3 from a range of images. However, *The software is provided "AS IS" but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.*

The API for the Feature class is provided in the Appendix.

All feature methods are also available for Work Package 2.3 – the closed shape identification algorithm

This class provides the feature calculation methods. The outline of each method is given in the following description, most are standard well known methods with many details available in the literature:

Area Moment: (invariant to rotation & translation)

This method calculates the area moment according to:

Area Moment = $\frac{1}{2}\left((x_0 y_{n-1} - x_{n-1} y_0) + \sum_{i=0}^{N-1}(x_i y_{i+1} - x_{i+1} y_i)\right)$

Convex Hull: (not invariant)

This method calculates the convex hull of a set of points X i.e., the smallest convex set containing the points X. If not all points are on the same line, then their convex hull is a convex polygon. Its most common representation is the list of its vertices ordered along its boundary clockwise or anti-clockwise. The C++ code in Features.cpp calls the file 2dch.h which is adapted from:

- Ken Clarkson, “A Short, Complete Planar Convex Hull Code", 1996. Available: http://cm.bell-labs.com/who/clarkson/2dch.c

Centroid: (invariant to scale)

This method calculates the centroid according to:

- CentroidX = $\frac{1}{6A}\left(((x_0 + x_{n-1})(x_0 y_{n-1} - x_{n-1} y_0)) + \sum_{i=0}^{N-1}(x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i)\right)$
- CentroidY = $\frac{1}{6A}\left(((y_0 + y_{n-1})(x_0 y_{n-1} - x_{n-1} y_0)) + \sum_{i=0}^{N-1}(y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i)\right)$

Perimeter: (invariant to rotation & translation)

This method calculates the perimeter according to:

Perimeter = $\sqrt{(x_0 - x_{n-1})^2 + (y_0 - y_{n-1})^2} + \sum_{i=0}^{N-1}\sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}$

Axis-aligned bounding box: (not invariant)

This method calculates the axis-aligned bounding box (rectangle) according to:

- Left = leftmost y coordinate

- Right = rightmost y coordinate
- Top = topmost x coordinate
- Bottom = bottommost x coordinate

Minimum Area Rectangle: (not invariant)

This method calculates the minimum area bounding box that encloses the convex hull of the shape using the “rotating callipers” technique.
The C++ code is adapted from:

```
//                    Intel License Agreement
//              For Open Source Computer Vision Library
//
// Copyright (C) 2000, Intel Corporation, all rights reserved.
```

There are two additional methods related to the minimum area bounding box:

Minimum Area Rectangle Perimeter:

This method returns the perimeter of the minimum area bounding rectangle.

Minimum Area Rectangle Area:

This method returns the area of the minimum area bounding rectangle.

Areas Ratio: (invariant to rotation, scale & translation)

This method calculates the areas ratio according to:

Area ratio = Area Moment / Convex Hull Area Moment

Roughness: (invariant to rotation, scale & translation)

Roughness = Perimeter / Convex Hull Perimeter

Aspect Ratio: (invariant to rotation, scale & translation)

This method calculates the aspect ratio according to:

Aspect Ratio = Perimeter / Minimum Bounding Box Perimeter

Stuffedness Ratio: (invariant to rotation, scale & translation)

This method calculates the stuffedness according to:

Stuffedness ratio = Area Moment / Minimum Bounding Box Area Moment

Hu Moments [Hu62] (from boundary points):

The first six Hu moments are translation invariant coupled with invariance to rotation. Moment 7 is also skew invariant as its sign can be used to detect mirror images. Hu derived these seven moments from algebraic invariants applied to the moment generating function under a rotation transformation. They consist of groups of nonlinear centralised moment expressions. The result is a set of absolute orthogonal (i.e. rotation) moment invariants, which can be used for scale, position, and rotation invariant pattern identification. These were used in a simple pattern recognition experiment to successfully identify various typed characters. They are computed from normalised centralised moments up to order three.

Normalisation (for scale invariance):

$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{r}}$ where, $r = \frac{p+q}{2} + 1$ and where $\mu_{pq} = \sum_x \sum_y (x_i - C_x)^p \times (y_i - C_y)^q \times I(x_i, y_i)$

where C_x is the x coordinate of the centroid and similarly, C_y is the y coordinate of the centroid.

The 7 Hu moments are given below (we note that the 7 expressions vary in the literature but we use the seven given below):

$\phi_0 = \eta_{20} + \eta_{02}$

$\phi_1 = (\eta_{20} - \eta_{02})^2 + 4\eta^2_{11}$

$\phi_2 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2$

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

$\phi_4 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] +$

$(3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03}) \times [3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$

$\phi_5 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11} \times (\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})$

Finally a skew invariant, to help distinguish mirror images, is:

$\phi_6 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] -$

$(\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03}) \times [3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$

Note that only boundary points are used in the calculations for the seven values this allows the moments to focus on the shape's external geometric properties (boundary variations) rather than the general overall shape characteristics when the fill area is included in the Hu moment calculations [TTT98]. Using the boundary points only, in essence, treats the contour as a "thin" region.

Eccentricity: (invariant to rotation, scale & translation)

This method calculates the eccentricity from the ratio principal second order moments.

- $Ecc = \frac{(M_{20} - M_{02})^2 + 4M_{11}^2}{(M_{20} + M_{02})^2}$ where M is calculated as **Error! Reference source not found.**.

Ellipticity: (invariant to rotation, scale & translation)

This method calculates Rosin's [R00] ellipticity feature where:

I_1 is the simplest Flusser & Suk affine moment invariant [R00] of the circle to characterise ellipses.

$I_1 = \frac{\mu_{20}\mu_{02} - \mu_{11}^2}{\mu_{00}^4}$

$$\text{Ellipticity} = \begin{cases} 16\pi^2 I_1 & \text{if} \quad I_1 \leq \dfrac{1}{16\pi^2} \\ \dfrac{1}{16\pi^2 I_1} & \text{otherwise} \end{cases}$$

Triangularity: (invariant to rotation, scale & translation)

This method calculates Rosin's [R00] triangularity feature where:
I_1 is again the simplest Flusser & Suk affine moment invariant [R00].

$$I_1 = \frac{\mu_{20}\mu_{02} - \mu_{11}^2}{\mu_{00}^4}$$

$$\text{Triangularity} = \begin{cases} 108 I_1 & \text{if} \quad I_1 \leq \dfrac{1}{108} \\ \dfrac{1}{108 I_1} & \text{otherwise} \end{cases}$$

Circularity: (invariant to rotation, scale & translation)

This method calculates the circularity according to:

Circularity = (Perimeter^2/area)

Circularity4Pi: (invariant to rotation, scale & translation)

This method calculates the normalised circularity according to:

Circularity4Pi = 4π* Area/ Perimeter^2

Maximum Gap Ratio: (invariant to rotation, scale & translation)

This method calculates the ratio of the maximum gap between pixels in the shape's contour and the perimeter of the shape according to:

Max gap ratio = Maximum Gap in Perimeter / Perimeter

Fourier coefficients (from boundary points): (invariant to rotation & translation)

The Fourier coefficients are the amplitude of the Fourier expansion of the cumulative angular bend around the shape's boundary points. Thus:

$$c[k] = \frac{\sqrt{\left(\sum_{0}^{n-1}\left(\theta_j \cos\left(\frac{-2\pi jk}{n}\right)\right)\right)^2 + \left(\sum_{0}^{n-1}\left(\theta_j \sin\left(\frac{-2\pi jk}{n}\right)\right)\right)^2}}{n}$$

Where θj is the cumulative angular bend at point ***j*** and ***n*** is the number of points in the boundary. If there are gaps in the perimeter these are not accounted for – they are missed.

The C++ code is adapted from:

> * *IMAGE EXTRACTION LIBRARY*
> *
> * *DISCOVIR - Distributed Content-based Visual Information Retrieval System on Peer-to-Peer(P2P) Network*
> * *http://www.cse.cuhk.edu.hk/~miplab/discovir*
> *

Conclusion

A set of 20 features (counting the Hu Moments and Fourier Coefficients each as one feature) has been produced. The set of features will allow a framework for assessing the similarity and relevancy of image components (global goodness) to be introduced. At this stage the work has not identified the optimal feature set for this task. However, a large body of ground truth data was produced from human perceptual experiments in Work Package 3.1 described in [HEA06] & [HHEA06]. This data illustrates the perceptually relevant image components for 84 images and will provide the basis for us to identify the optimal feature set for perceptual relevance. As stated previously, there is no similarity data available to us for image components so the proposal to investigate different methods for similarity assessment has not been implemented yet. This process will need further investigation and will be incorporated with the system integration process.

References

[A99] S. Alwis
Content-Based Retrieval of Trademark Images,
PhD Thesis, Dept. of Computer Science, University of York, UK, 1999

[A03] S . Antani, L.R. Long, D.J. Lee, and G.R. Thoma
Evaluation of Shape Indexing Methods for Content-Based Retrieval of X-Ray Images.
In Proceedings of IS&T/SPIE Electronic Imaging Science and Technology, Storage and Retrieval for Media Databases. 2003 Jan; 5021: 405-16.

[CK99] D. Y-M. Chan, and I. King
Genetic Algorithm for Weights Assignment in Dissimilarity Function for Trademark Retrieval
In Third International Conf. on Visual Information and Information Systems (VISUAL'99), volume 1614 of Lecture Notes in Computer Science, pages 557--565, Amsterdam, The Netherlands, 1999. Springer Verlag.

[ERE03] J. P. Eakins, K. J. Riley, and J. D. Edwards.
Shape Feature Matching for Trademark Image Retrieval,
In Image and Video Retrieval: Second International Conference, CIVR 2003. Lecture Notes in Computer Science, Volume 2728, Jan 2003, Pages 28 – 38.

[F95] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker,
Query by Image and Video Content: The QBIC System,
Computer, vol. 28, no. 9, pp. 23-32, Sept., 1995.

[HHEA06] Hodge, V.J., Hollier, G., Eakins, J. & Austin, J.
Eliciting Perceptual Ground Truth for Image Segmentation.
To appear, International Conference on Image and Video Retrieval (CIVR2006). Tempe, Arizona, July 13-15, 2006.

[HEA06] Hodge, V.J., Eakins, J. & Austin, J.
Eliciting Perceptual Ground Truth for Image Segmentation.
Technical Report YCS 401(2006), Department of Computer Science, University of York.

[Hu62] M.-K. Hu,
Visual pattern Recognition by Moment Invariants.
IRE Transactions on Information Theory, IT-8:179-187, 1962.

[JNT06] H. Jiang, C.-W. Ngo, and H-K Tan.
Gestalt-based feature similarity measure in trademark database
Pattern Recognition, 39: pp. 988 – 1001, 2006.

[N93] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, P. Yanker, C. Faloutsos, and G. Taubin.
The QBIC project: Querying images by content using color, texture, and shape.
In Proceedings of the SPIE Conference on Storage and Retrieval for Image and Video Databases, 2-3 February '93, San Jose, CA, pages 173-187, 1993.

[R00] P. L. Rosin,
Measuring Shape: Ellipticity, Rectangularity, and Triangularity,
In Proceedings of 15th International Conference on Pattern Recognition (ICPR'00) - Volume 1, 2000.

[TTT98] G. Tzanetakis, M. Traka, and G. Tziritas
Motion estimation based on affine moment invariants.
In Proc. European Signal Processing Conference (Euspico), Rhodes, Greece, 1998

Appendix A

Features class API (from features.h)

In the following `point` is defined as:

```
struct point{
int x;
int y;
};
```

Which is defined in the file EndPoints.h produced as part of deliverable 2.3.

`Std::vector<point>` is a vector (from the Standard Template library) of points

The methods available in the class Features are:

`float getArea(const std::vector<point>& pointsVector);`

Return the area moment as a floating point value where the area is calculated from the boundary defined by the listing of points in the parameter pointsVector.

`std::vector<point> getHull(const std::vector<point>& allPointsVector);`

Return a vector of points representing the points in the convex hull calculated from the boundary defined by the listing of points in the parameter pointsVector. The code for calculating the hull is in the file 2dch.h.

`void getCentroid(const std::vector<point>& pointsVector, float & centroidX, float & centroidY);`

The centroid is calculated from the boundary defined by the listing of points in the parameter pointsVector and returned as two float parameters where centroidX is the x-coordinate and centroidY is the y-coordinate.

`void getAreaAndCentroid(const std::vector<point>& pointsVector, float & areaMoment, float & centroidX, float & centroidY);`

The centroid and area are calculated from the boundary defined by the listing of points in the parameter pointsVector. The area moment is returned as a float value in areaMoment. The centroid is returned as two float parameters where centroidX is the x-coordinate and centroidY is the y-coordinate.

`float getPerimeter(const std::vector<point>& pointsVector);`

Return the perimeter as a floating point value where the perimeter is calculated from the boundary defined by the listing of points in the parameter pointsVector

`int boundingBox(const std::vector<point>& pointsVector, int & left, int& right, int& top, int& bottom);`

The method takes the boundary defined by the listing of points in the parameter pointsVector and calculates the minimum axis-aligned bounding box returning the leftmost coordinate in the parameter left, the rightmost coordinate in the parameter right, the topmost coordinate in the parameter top and the bottommost coordinate on the parameter bottom. It also returns the area of the bounding box as an integer.

`std::vector<point> cvMinAreaRect2( const std::vector<point>& pointsVector );`

The method takes the boundary defined by the listing of points in the parameter pointsVector and calculates the minimum area bounding box returning the four corner points of the box as a vector of points.

float cvMinAreaRectPerim(const std::vector<point>& pointsVector);
The method takes the boundary defined by the listing of points in the parameter pointsVector and calculates the perimeter of the minimum area bounding box returning the perimeter as a float.

float cvMinAreaRectArea(const std::vector<point>& pointsVector);
The method takes the boundary defined by the listing of points in the parameter pointsVector and calculates the area of the minimum area bounding box returning the area as a float.

float areasRatio(const std::vector<point>& pointsVector);
The convex hull is calculated from the boundary defined by the listing of points in the parameter pointsVector and the area of this hull and the closed path are calculated. The method returns a float representing the area moment of the closed shape divided by the area moment of the convex hull.

float roughnessRatio(const std::vector<point>& pointsVector);
The convex hull is calculated from the boundary defined by the listing of points in the parameter allPointsVector and the perimeter of this hull and the closed path are calculated. The method returns a float representing the perimeter of the closed shape divided by the perimeter of the convex hull.

float aspectRatio(const std::vector<point>& pointsVector);
The minimum area bounding box is calculated from the boundary defined by the listing of points in the parameter pointsVector and the perimeter of this box and the closed path are calculated. The method returns a float representing the perimeter of the closed shape divided by the perimeter of the minimum area bounding box.

float stuffednessRatio(const std::vector<point>& pointsVector);
The minimum area bounding box is calculated from the boundary defined by the listing of points in the parameter pointsVector and the area of this box and the area moment of the closed path are calculated. The method returns a float representing the area moment of the closed shape divided by the area of the minimum area bounding box.

float * momentInvariantHu7Boundary(const std::vector<point>& pointsVector);
This method returns a 7-dimensioanl array of floats representing the seven Hu moments as calculated from the boundary defined by the listing of boundary points in the parameter pointsVector

float momentEccentricityBoundary(const std::vector<point>& pointsVector);
This method returns a float representing the eccentricity moment as calculated from the boundary defined by the listing of boundary points in the parameter pointsVector

float ellipticityBoundary(const std::vector<point>& pointsVector);
This method returns a float representing Rosin’s ellipticity as calculated from the boundary defined by the listing of boundary points in the parameter pointsVector

float triangularityBoundary(const std::vector<point>& pointsVector);
This method returns a float representing the Rosin’s triangularity as calculated from the boundary defined by the listing of boundary points in the parameter pointsVector

float circularity(const std::vector<point>& pointsVector);

This method returns a float representing the circularity as calculated from the boundary defined by the listing of boundary points in the parameter pointsVector

`float circularity4Pi(const std::vector<point>& pointsVector);`
This method returns a float representing the normalised circularity as calculated from the boundary defined by the listing of boundary points in the parameter pointsVector

`float getMaxGap(const std::vector<point>& pointsVector);`
This method returns a float representing the maximum gap between two adjacent points in the boundary as calculated from the boundary defined by the listing of boundary points in the parameter pointsVector

`float getGapRatio(const std::vector<point>& pointsVector);`
This method returns a float representing the ratio of the maximum gap between two adjacent points in the boundary divided by the perimeter as calculated from the boundary defined by the listing of boundary points in the parameter pointsVector

`float * FourierCoefficient(const std::vector<point>& pointsVector, int m);`
This method returns an m-dimensional array of floats representing the m Fourier coefficients as calculated from the boundary defined by the listing of boundary points in the parameter pointsVector where m is an integer-valued parameter.

`private:`

`float momentCentralGeometricBoundary(const std::vector<point>& pointsVector, float cX, float cY, int p, int q);`
This method is called by the Hu moments, eccentricity and affine Flusser & Suk moments methods.

`float momentAffineFlusserSukBoundary(const std::vector<point>& pointsVector);`
This method is called by the ellipticity and triangularity methods.

`float angularDirection(point p1, point p2);`
This method is called by the Fourier coefficients method.

`float * cumulativeAngularBend(const std::vector<point>& pointsVector);`
This method is called by the Fourier coefficients method.

`void icvRotatingCalipers( const std::vector<point>& pointsVector, int mode, point* out );`
This method is called by the minimum area rectangle, area of the minimum area rectangle and perimeter of the minimum area rectangle methods.