

# User's Manual

# **ACTIV****FEATURECALC**

The Blob Analyzer

Microsoft, Windows, Windows NT, Windows 2000, Windows XP, Visual Basic, Microsoft .NET, Visual C++, Visual C#, and ActiveX are either trademarks or registered trademarks of Microsoft Corporation.

All other nationally and internationally recognized trademarks and tradenames are hereby recognized.

Copyright © 2001-2008 by MVTec Software GmbH, München, Germany



Edition 1	September 2001	(ActivVisionTools 2.0)
Edition 2	November 2002	(ActivVisionTools 2.1)
Edition 3	January 2005	(ActivVisionTools 3.0)
Edition 4	February 2006	(ActivVisionTools 3.1)
Edition 5	May 2008	(ActivVisionTools 3.2)

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission of the publisher.

More information about ActivVisionTools can be found at:

<http://www.activ-vision-tools.com>

# How to Read This Manual

---

This manual explains how to use `ActivFeatureCalc` to analyze the objects extracted by `ActivBlobFinder`. It describes the functionality of `ActivFeatureCalc` and its cooperation with other `ActivVisionTools` with Visual Basic examples. Before reading this manual, we recommend to read the manual [Getting Started with ActivVisionTools](#), which introduces the basic concepts of `ActivVisionTools`, the [User's Manual for ActivView](#) to learn how to load and display images, and the [User's Manual for ActivBlobFinder](#), which explains how to extract the objects you want to analyze.

To follow the examples actively, first install and configure `ActivVisionTools` as described in the manual [Getting Started with ActivVisionTools](#). For each example in this manual, there is a corresponding Visual Basic project; these projects can be found in the subdirectory `examples\manuals\activfeaturecalc` of the `ActivVisionTools` base directory you specified during installation (default: `C:\Program Files\MVTec\ActivVisionTools`). Of course, you can also create your own Visual Basic projects from scratch.

We recommend to **create a private copy of the example projects** because by experimenting with the projects, you also change their *state*, which is then automatically stored in the so-called description files (extension `.dsc`) by `ActivVisionTools`. Of course, you can restore the state of a project by retrieving the corresponding description file from the CD.





# Contents

---

<b>1</b>	<b>About ActivFeatureCalc</b>	<b>1</b>
1.1	Introducing ActivFeatureCalc . . . . .	2
1.2	The Sub-Tools of ActivFeatureCalc . . . . .	15
<b>2</b>	<b>Using ActivFeatureCalc</b>	<b>17</b>
2.1	Extracting Objects . . . . .	18
2.2	Selecting Features . . . . .	20
<b>3</b>	<b>Combining ActivFeatureCalc with other ActivVisionTools</b>	<b>23</b>
3.1	Converting Results to Other Units . . . . .	24
3.2	Evaluating Results . . . . .	26
3.3	Output of Results . . . . .	28
<b>4</b>	<b>Tips &amp; Tricks</b>	<b>31</b>
4.1	Customizing the Two Execution Modes . . . . .	32
4.2	Accessing Results Via the Programming Interface . . . . .	34
<b>A</b>	<b>Mathematical Background of the Features</b>	<b>47</b>
A.1	Basic Moments . . . . .	48
A.2	The Equivalent Ellipse . . . . .	48
A.3	Complex Moments . . . . .	49
A.4	Gray Value Moments . . . . .	49
A.5	Gray Value Features . . . . .	50

A.6 Texture Features . . . . .	50
--------------------------------	----

# Chapter 1

## About ActivFeatureCalc

---

This chapter will introduce you to the features and the basic concepts of ActivFeatureCalc. It gives an overview about ActivFeatureCalc's *master tool* and its *support tools*, which are described in more detail in [chapter 2](#) on page 17.

<b>1.1</b>	<b>Introducing ActivFeatureCalc</b>	<b>2</b>
1.1.1	The World of Features	2
1.1.2	Converting Features to Other Units	13
1.1.3	ActivFeatureCalc and ActivAlignment	13
<b>1.2</b>	<b>The Sub-Tools of ActivFeatureCalc</b>	<b>15</b>

## 1.1 Introducing ActivFeatureCalc

With the help of ActivFeatureCalc you can inspect the blobs extracted by ActivBlobFinder by analyzing various *shape and gray value features* of the blobs. These features describe the appearance of a blob, and thereby the corresponding object, from various “point of views”: For example, features like Mean or Contrast describe the texture of an object, while features like Center Row and Center Column describe its position.

The currently available features are described in [section 1.1.1](#); [section 1.1.2](#) on page 13 shows which features can be converted to other units besides pixels, while [section 1.1.3](#) on page 13 takes a closer look at what happens if you use ActivFeatureCalc together with ActivAlignment.

The results of ActivFeatureCalc, i.e., the values of the calculated features for each blob can be displayed using ActivDataView or ActivFeatureHistogram and then further evaluated using ActivDecision, before you output them via ActivFile, ActivSerial, or ActivDigitalIO; furthermore, you can access results via the programming interface (see [section 4.2](#) on page 34).

### 1.1.1 The World of Features

#### Position

In ActivFeatureCalc, the position of a blob can be calculated by determining the *center of gravity* of the corresponding area in the image, i.e., the features Center Column and Center Row. In [figure 1.1](#) the position of nine elements of a ball grid array (*BGA*) is calculated; the wrong position of the ball in the lower right corner clearly shows up in the column coordinate. Please note that the accuracy of the calculated center of gravity depends on the number of pixels of the blob; see below for similar features which can be used for small objects.

#### Size and Extent

The size or Area of a blob is typically calculated by determining the number of pixels within the blob region; if the blob has holes, the corresponding pixels do not contribute to the overall area. For the feature Area Convex the area of its *convex hull* around the blob is determined, including all holes. See [figure 1.2](#) for an example.

ActivFeatureCalc allows to measure the extent of a blob in different ways: The features Height and Width correspond to the extent along the image axes (see [figure 1.3a](#) on page 4). These features are easy to calculate, but are affected by the orientation of the blob. In other words, if the inspected object rotates slightly, Height and Width may change significantly.

Alternatively, ActivFeatureCalc calculates the features Major Extent and Minor Extent, which correspond to the extent of the smallest rectangle surrounding the blob (see [figure 1.3b](#)

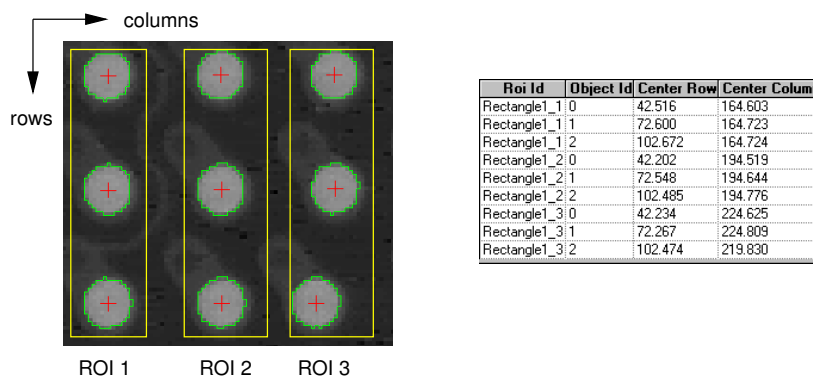


Figure 1.1: The center of gravity.

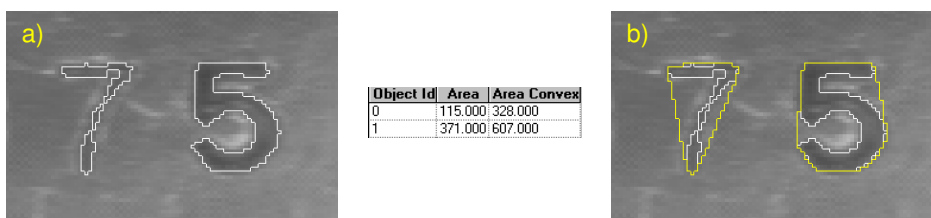


Figure 1.2: The (a) area and (b) convex area of a blob.

on page 4). These features are more robust against changes of blob orientation. However, small changes in the distribution of pixels (like small “hairs”) may “tilt” the rectangle; for example, the rectangles surrounding the two zeroes in [figure 1.3b](#) on page 4 have a different orientation though the zeroes look similar to the human eye.

A third method to calculate the extent first determines the *equivalent ellipse* and returns its radii in the features Major Radius and Minor Radius. The equivalent ellipse has the same *moments of inertia* as the blob, i.e., offers the same resistance to being rotated around the two principal axes; its center is the center of mass (see [appendix A](#) on page 47 for the mathematical background). These features are robust against changes of the blob orientation and of the distribution of pixels (compare the zeroes in [figure 1.3b](#) and [figure 1.3c](#)). Please note that the accuracy of the ellipse parameters depends on the number of pixels of the blob; see below for similar features which can be used for small objects.

Finally, ActivFeatureCalc offers to calculate the radius of the smallest surrounding circle (Radius Outer Circle) and of the largest circle fitting inside the blob



Figure 1.3: The extent of a blob: (a) along the image axes; (b) the extent of its smallest surrounding rectangle; (c) the radii of its equivalent ellipse; (d) its inner and outer circle.

(Radius Inner Circle), as depicted in [figure 1.3d](#).

## Orientation

In fact, a blob by itself has no orientation; however, ActivFeatureCalc lets you determine the orientation with the help of two intermediate constructs: the *smallest surrounding rectangle* and the *equivalent ellipse*. In [figure 1.4](#), these two methods are used to determine the orientation (Orientation Ellipse and Orientation Rectangle, respectively) of characters on the circular rim of a part. In both methods, the orientation is 0 if the major axis of the rectangle or

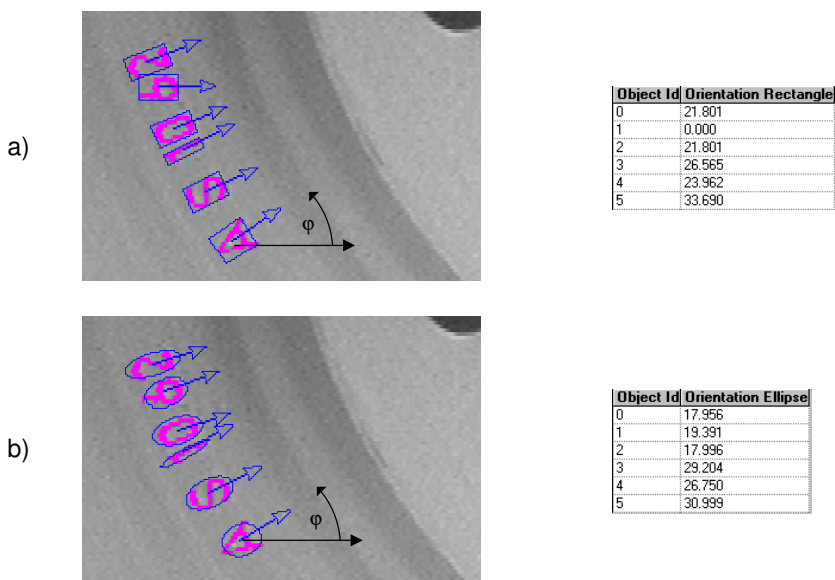


Figure 1.4: The orientation of (a) the smallest surrounding rectangle and of (b) the equivalent ellipse.

ellipse is parallel to the horizontal axis.

The example in [figure 1.4](#) was chosen on purpose to show that neither method is “perfect” in the sense that it yields the desired results in all cases. Thus, Orientation Rectangle “fails” for the digit 6, Orientation Ellipse for the digit 1. The reason for this behavior is (obviously) not that the methods are poorly implemented, but more fundamental: The digits in question are not described well by the smallest surrounding rectangle or equivalent ellipse, respectively.

Because of their symmetry, both the smallest surrounding rectangle and the equivalent ellipse only calculate angles in the range  $\pm 90^\circ$ . In contrast, the feature Orientation allows to calculate angles in the range  $\pm 180^\circ$ . For this, the underlying algorithm computes the equivalent ellipse; furthermore, it determines the point on the contour with the biggest distance to the center of gravity and uses it as a sort of reference point. Note that this method only works well for objects which are roughly symmetrical to their major axis but not symmetrical to their minor axis, like the plugs examined in [figure 1.5](#).

## Contour

To describe the contour of a blob, i.e., its boundary, ActivFeatureCalc calculates the Con-

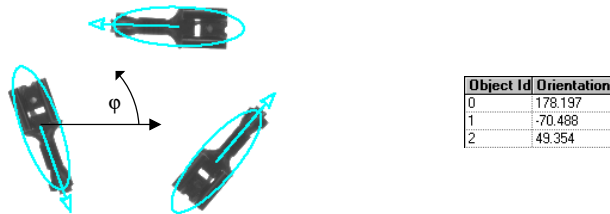


Figure 1.5: Calculating the orientation in the range  $\pm 180^\circ$ .

Contour Length. The distance between two neighboring contour points parallel to the coordinate axes is rated as 1, the distance along the diagonal as  $\sqrt{2}$ . Note that only the “outer” boundary is analyzed; holes within the blob are disregarded. In figure 1.6, the feature Contour Length is used to analyze fuse wires in different types of car fuses.

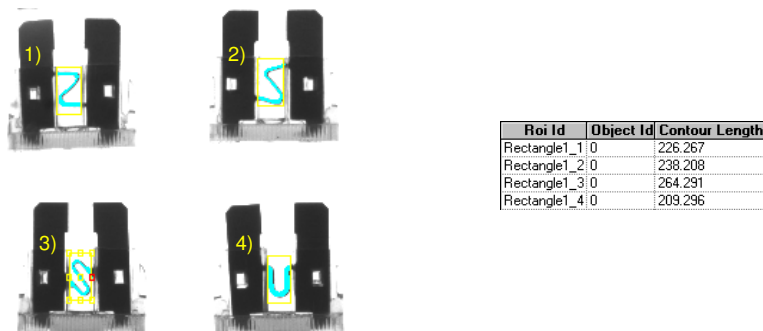


Figure 1.6: The length of the contour.

## Shape

ActivFeatureCalc provides several features which describe the shape of the blob. The features Compactness, Circularity, Anisometry, and Bulkiness have in common that they describe how similar the blob is to a circle. The feature Convexity describes how convex a blob is. Figure 1.7 shows the resulting values for an example image of pills.

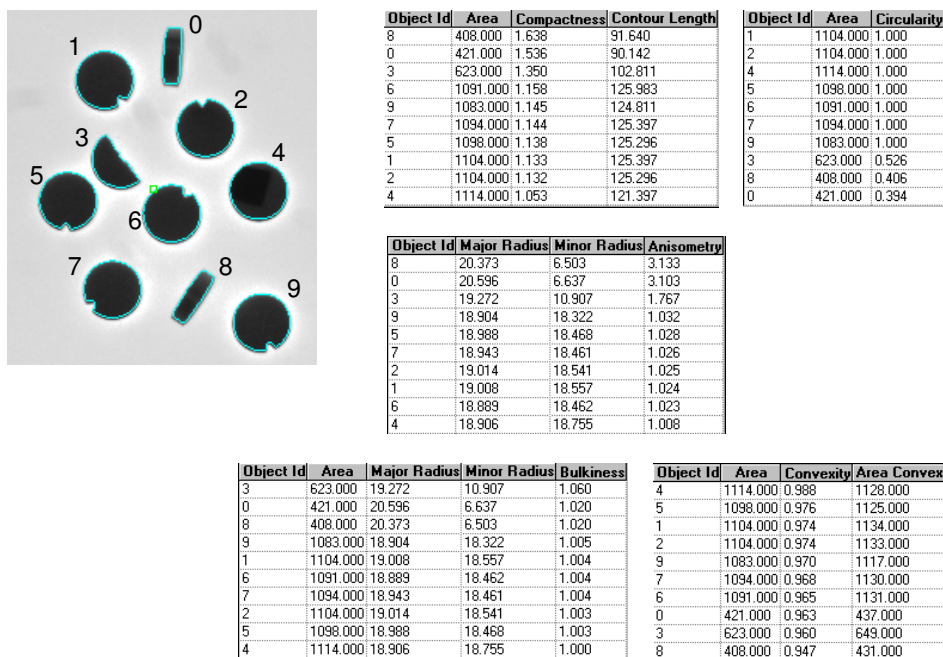


Figure 1.7: Calculating different shape features.

The Compactness of a blob is calculated as

$$\text{Compactness} = \frac{\text{Contour Length}^2}{4\pi\text{Area}}$$

For a perfect circle it becomes 1, while getting  $> 1$  for other. In the example, pill no. 4 is most similar to a circle as it has no “dent”, the two pills lying on their edge (no. 0 and 8) are the most dissimilar.

The Circularity of a blob is calculated as

$$\text{Circularity} = \frac{\text{Area}}{R_{max}^2\pi}$$

with  $R_{max}$  being the largest distance from the center of mass to the contour (not to be confused with Major Radius!). Thus, for a perfect circle it becomes 1, while getting  $< 1$  for other shapes. In the example, all pills except no. 0, 3, and 8 have a value of 1. Because only the largest distance to the contour is examined, the small “dents” do not influence the result in this example.

The Anisometry of a blob is calculated as

$$\text{Anisometry} = \frac{\text{Major Radius}}{\text{Minor Radius}}$$

It describes how “elongated” a blob is. For a perfect circle it becomes 1, while getting  $> 1$  for other shapes; the more “elongated” a blob is the larger its Anisometry. In the example, the blobs are ordered similarly to the feature Compactness.

The Bulkiness of a blob is calculated as

$$\text{Bulkiness} = \frac{\pi \cdot \text{Major Radius} \cdot \text{Minor Radius}}{\text{Area}}$$

It describes how much the blob “bulges”. For a perfect circle or ellipse it becomes 1, while getting  $> 1$  for other shapes. In the example, the blobs are ordered similarly to before, however without large difference, as none of the blobs “bulges”.

The Convexity of a blob is calculated as

$$\text{Convexity} = \frac{\text{Area}}{\text{Area Convex}}$$

It describes how much the blob differs from its convex hull. For a perfectly convex shape, e.g., a circle or polygon, it becomes 1, while getting  $< 1$  for concave shapes or blobs with holes. In the example, pill no. 4 is the most convex, the “dents” in the pills no. 1, 2, 5, 6, 7, and 9 show up as concavities.

In [figure 1.8](#), some of the pills have holes, which is signalled by the feature Number of Holes. Note that pill no. 7 has no hole but an indentation. In the following, the effect of holes on the different shape features is described.

The Compactness of a blob increases with holes as its area decreases. Thus, pills no. 2, 5, 6, and 8 have a significantly larger compactness in comparison with [figure 1.7](#) on page 7; pill no. 7 has no hole, but a new concavity, thus the contour length is increased as well. The position of the hole has no influence.

The Circularity of a blob decreases with holes as its area decreases. Pill no. 7 shows the same decrease as the others, as holes and concavities have the same influence on the Circularity. The position of the hole has no influence.

The Anisometry of a blob is only influenced by holes if they affect the features Minor Radius and Major Radius in a different way. This is the case for pills no. 7 and 2; their Minor Radius decreases significantly, while the Major Radius remains almost constant.

The Bulkiness is affected more by a hole than the Anisometry, as it depends not only on Minor Radius and Major Radius but also on the blob area which decreases with each hole. Another difference is that the influence of a hole is larger if it affects the features Minor Radius and Major Radius in a similar way. Therefore, pills no. 5 and 6 are affected most because the hole is in the middle, leading to an increase both of Minor Radius and of Major Radius.

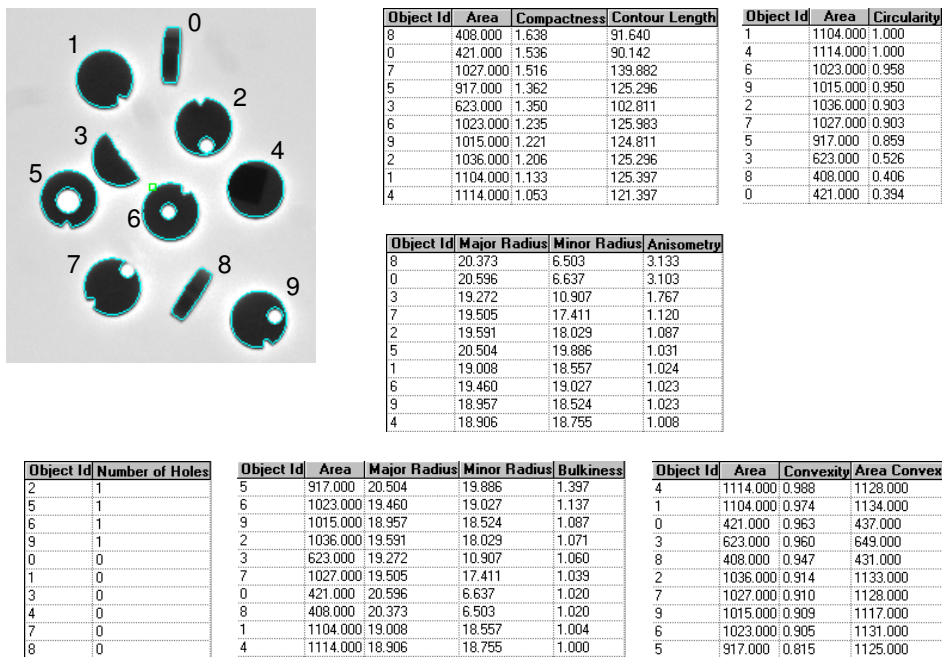


Figure 1.8: How holes affect the shape features.

The Convexity of a blob decreases with holes as with concavities.

Figure 1.9 shows how an irregular contour affects the shape features. In the example, this was provoked by using a low threshold for the blob extraction (compare with figure 1.7 on page 7). The blobs are ordered according to their Anisometry.

The Compactness increases quadratically with the length of its contour, therefore especially pills no. 9 and 6 show a marked increase; as a consequence, the “bad” pills (no. 0, 3, and 8) cannot be distinguished by this feature anymore.

The Convexity decreases as the irregular contour forms concavities. “Good” and “bad” pills cannot be distinguished by this feature anymore.

The Anisometry rises for most pills, but the “bad” pills still show significantly higher values.

The Bulkiness increases for most pills as the irregular contour forms “bulges”. “Good” and “bad” pills cannot be distinguished by this feature anymore.

The Circularity is affected most by the irregular contour. Only pill no. 4 still appears as being circular. “Good” and “bad” pills cannot be distinguished by this feature anymore.

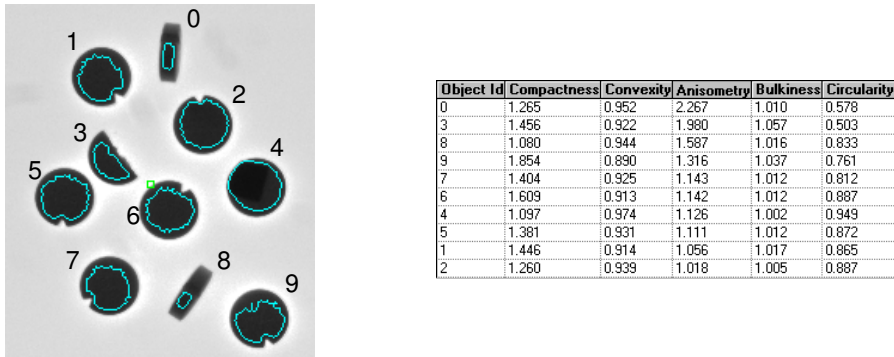


Figure 1.9: How irregular contours affect the shape features.

## Moments

ActivFeatureCalc provides various features based on moments: The normalized 2nd moments 2nd Moments 02 and 2nd Moments 20, the normalized 3rd moments 3rd Moments 30, 3rd Moments 21, 3rd Moments 12, and 3rd Moments 03, and the more complex moments 2nd Moments Phi 1, 2nd Moments Phi 2, Central Moments Psi 1, Central Moments Psi 2, Central Moments Psi 3, and Central Moments Psi 4. These features can be used in classification applications. The mathematical background is given in [appendix A](#) on page 47.

## Gray Values

The features described so far are based on spatial information about the blob, i.e., only the (relative) position of the pixels of a blob counts, not their gray values. In contrast, this section and the following one describe features which evaluate the distribution of the gray values occurring in the blob.

ActivFeatureCalc provides 6 basic gray value features: The smallest and the largest occurring gray value (Min, Max) and the corresponding gray value Range, the Mean gray value and the Deviation from it (see [appendix A.5](#) on page 50 for the mathematical background), and the so-called Median which we will focus on below. In [figure 1.10](#), these features are calculated for two types of capacitors. All capacitors have an identical smallest gray value because this value was used as threshold for the blob extraction. As you can see, the two capacitor types (0 & 1 vs. 2 & 3) can be easily distinguished by their mean gray value; furthermore, capacitors 0 and 1 have a larger range and deviation of occurring gray values.

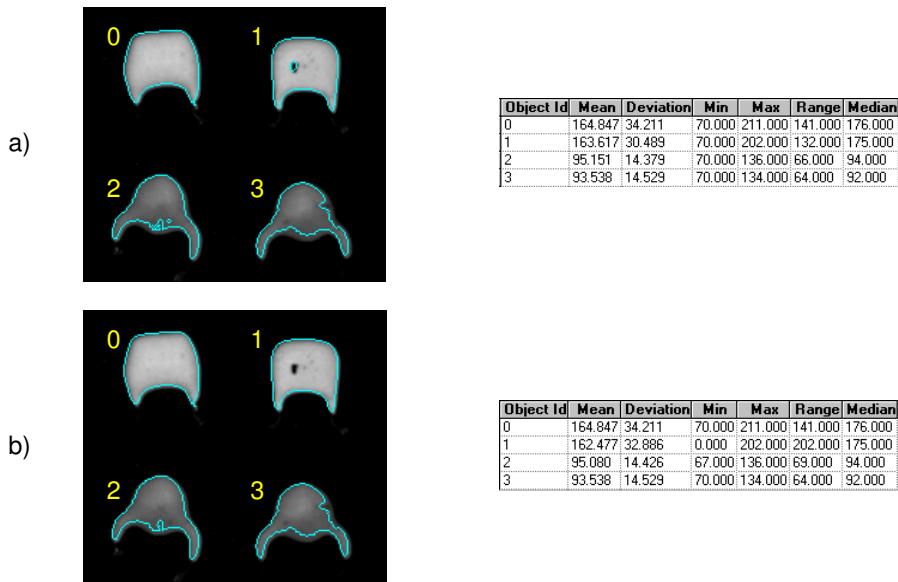


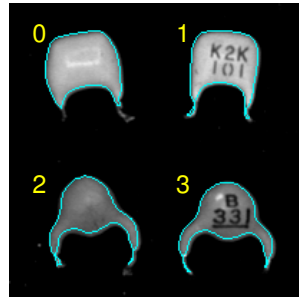
Figure 1.10: Basic gray value features; (a) with hole, (b) without hole.

In [figure 1.10b](#), the hole corresponding to the dark “stain” on capacitor no. 1 was closed, resulting in a clear change in the feature *Min*. Furthermore, the feature *Mean* decreases by more than one gray value, even though only a small number of pixels have been added. This shows that the feature *Mean* is sensitive to small disturbances if their gray value differs significantly because each pixel contributes equally. In contrast, the *Median* is calculated by sorting the gray values of all pixels in ascending order and then determining the gray value of the element in the exact middle of the list. As you can see in [figure 1.10](#), the *Median* stays constant with or without the hole.

## Texture

ActivFeatureCalc provides six gray level features describing the *texture* of a blob. [Figure 1.11](#) shows their values together with the basic gray level features for two types of capacitors, each one once with print and once without. For the mathematical background please refer to [appendix A](#) on page 47.

The feature *Entropy* is a measure for the “disorder” of the gray values in the blob. *Entropy* is high if many different gray values occur with a similar frequency; it becomes 0 if only one gray value occurs and 8 if all (256) gray values occur with the same frequency. In [figure 1.11](#), it



Object Id	Energy	Correlation	Homogeneity	Contrast	Mean	Deviation	Min	Max	Range	Median	Entropy	Anisotropy
0	0.008	0.916	0.405	14.658	164.653	34.585	58.000	238.000	180.000	174.000	6.892	-0.547
1	0.004	0.772	0.283	49.736	165.522	37.651	63.000	227.000	164.000	178.000	6.919	-0.550
2	0.012	0.795	0.499	8.609	101.460	16.277	61.000	202.000	141.000	100.000	5.967	-0.491
3	0.005	0.759	0.329	30.067	99.238	30.404	9.000	233.000	224.000	101.000	6.715	-0.515

Figure 1.11: Gray value features describing the texture.

is lowest for capacitor no. 2, which is quite homogeneous; its Range and Deviation are low as well.

The feature Anisotropy, which lies between  $-1$  and  $0$ , describes how much of the disorder stems from the darker half of the blob's pixels. It becomes  $-0.5$  if dark and light pixels are equally "disorderly"; values between  $-0.5$  and  $0$  signal that the lighter part of the blob is more homogeneous, values smaller than  $-0.5$  that the dark part is more homogeneous. If the Entropy is  $0$ , the Anisotropy is  $0$  as well.

In the features described above, the pixels were examined one by one. The following features examine how frequently two gray values occur next to each other, the neighborhood being evaluated horizontally, vertically, and along the two diagonals. The feature Energy is high if a few gray value combinations dominate. It becomes  $1$  if there is only one combination, i.e., if the gray value is constant over the blob; the Energy of a perfect chessboard pattern is  $0.5$ . In [figure 1.11](#), capacitor no. 2 has the largest Energy as it has the most homogeneous gray value distribution.

The feature Homogeneity is large if the absolute gray value differences between neighboring pixels are small. It becomes  $1$  for blobs with a constant gray value and  $0.5$  if the average gray value difference is  $1$ , which is the case for capacitor no. 2.

In contrast, the feature Contrast increases quadratically with the absolute gray value differences between neighboring pixels, becoming  $0$  for blobs with a constant gray value. For [figure 1.11](#), the illumination has been chosen such that the capacitors of the same type have a similar Mean gray value with or without print; however, they can be easily distinguished by looking at their Contrast.

The feature `Correlation` is a measure of the homogeneity of all neighborhood relations, with values lying between  $-1$  and  $1$ . A large value signals that large parts of the blob have an identical gray value (each); in [figure 1.11](#) for example, this is the case for capacitor no. 0. A negative value signals that pixels have neighbors with a very different gray value. For a chessboard pattern the `Correlation` becomes 0, because the (negative) values for the horizontal and vertical neighbors and the (positive) values for the diagonal neighbors “cancel out”.

## Using Gray Values for the Equivalent Ellipse

In the sections above, the equivalent ellipse was determined from the blob region alone, i.e., without using the gray values. This may lead to problems in the case of small objects like the balls in [figure 1.12](#): When using a high threshold only few pixels are extracted (see [figure 1.12a](#)); a single pixel more or less can thus influence the calculation significantly and lead to inaccurate results. By using a lower threshold as in [figure 1.12b](#) the calculations get more stable, however the extent of the equivalent ellipse is now larger than desired.

The solution is to include the gray values when calculating the center of gravity and the equivalent ellipse. The corresponding features are called `Gray Center Column`, `Gray Center Row`, `Gray Orientation`, `Gray Major Radius`, and `Gray Minor Radius`. The principal idea is that the contribution of a pixel is weighted by its gray value, i.e., bright pixels influence the result more than dark pixels (for the mathematical background see [appendix A.4](#) on page 49). As you can see in [figure 1.12c](#), the calculated ellipse fit the balls better than in [figure 1.12b](#).

Note that the feature `Gray Area` was intentionally omitted in the example because it is not a more accurate version of `Area` but something completely different: The `Gray Area` is calculated as the sum over all gray values in the blob; it can be interpreted as the gray value volume of the blob.

## 1.1.2 Converting Features to Other Units

You can convert features representing spatial information, i.e., the position and size and contour length, from pixels into other units if you calibrated your vision system using `ActivGeoCalib` or `AVTViewCalibration` (see the [User’s Manual for ActivGeoCalib](#) and [section 3.1](#) on page 24 for more information about these tools).

## 1.1.3 ActivFeatureCalc and ActivAlignment

Using `ActivAlignment` you can align the regions of interest (ROIs) of `ActivBlobFinder`/ `AFeature` to a certain part in the image. The main reason for aligning an ROI for blob extraction is to assure that all objects to be extracted and analyzed lie inside the ROI even if the inspected part moves. Most of the features are not influenced by the position or orientation of a blob; the

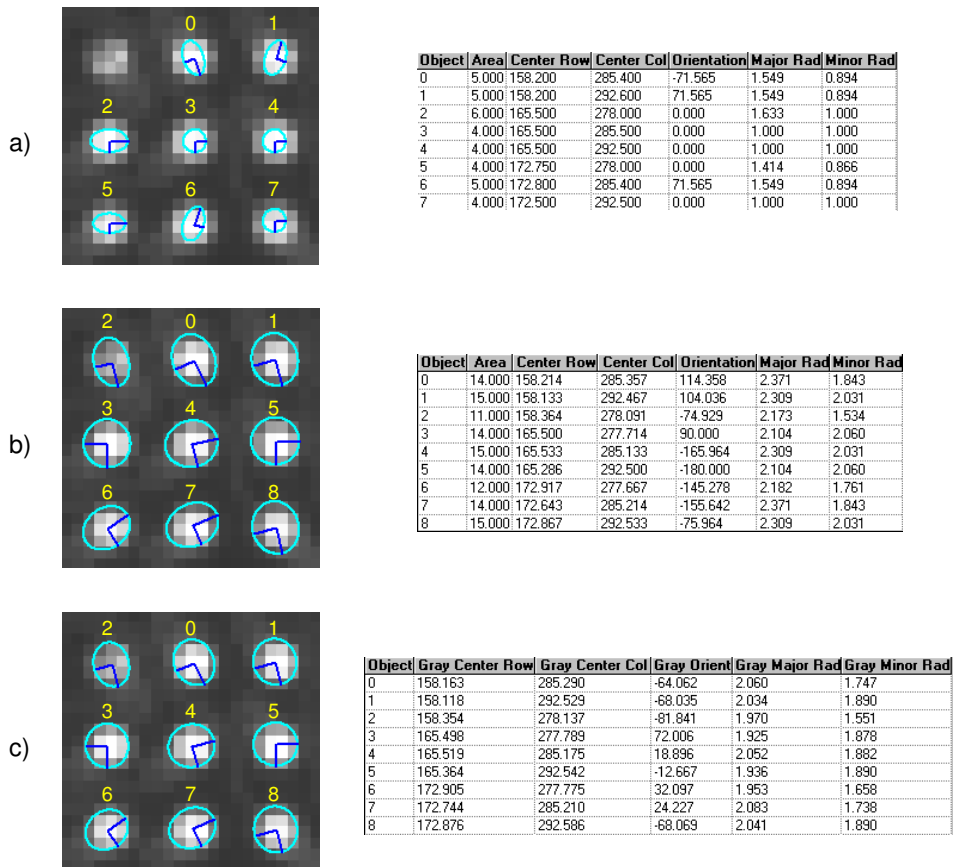


Figure 1.12: Comparing standard and gray value moments: (a) standard ellipse, high threshold; (b) standard ellipse, low threshold; c) gray value ellipse, low threshold.

exceptions are of course the position and orientation of the blob themselves. *ActivAlignment* offers to transform these features back into the so-called reference image (see the [User's Manual for ActivAlignment](#) for more information). This mechanism is useful if you want to inspect the position or orientation of blobs relative to a fixed point in the image.



Please note that **the features Height and Width cannot be transformed back into the reference image!**

## 1.2 The Sub-Tools of ActivFeatureCalc

Besides its *master tool*, ActivFeatureCalc provides 4 *support tools*. In [figure 1.13](#) on page 16 they are depicted together with other ActivVisionTools that you will use in a typical ActivFeatureCalc application.

AVTFeatureCalc is the *master tool* of ActivFeatureCalc. Note that this ActiveX control does not have a graphical user interface; thus, it is represented by its icon at design time and invisible at run time. If you forget to add AVTFeatureCalc to the form and only add the support tools, they are disabled.



AVTFeatureCalcBasic is a *support tool* of ActivFeatureCalc. It allows you to select which basic features are to be calculated.



AVTFeatureCalcShape is a *support tool* of ActivFeatureCalc. It allows you to select which shape features are to be calculated.



AVTFeatureCalcMoments is a *support tool* of ActivFeatureCalc. It allows you to select which moments are to be calculated.



AVTFeatureCalcGray is a *support tool* of ActivFeatureCalc. It allows you to select which gray value features are to be calculated.



How to use the support tools is described in [section 2.2](#) on page 20.

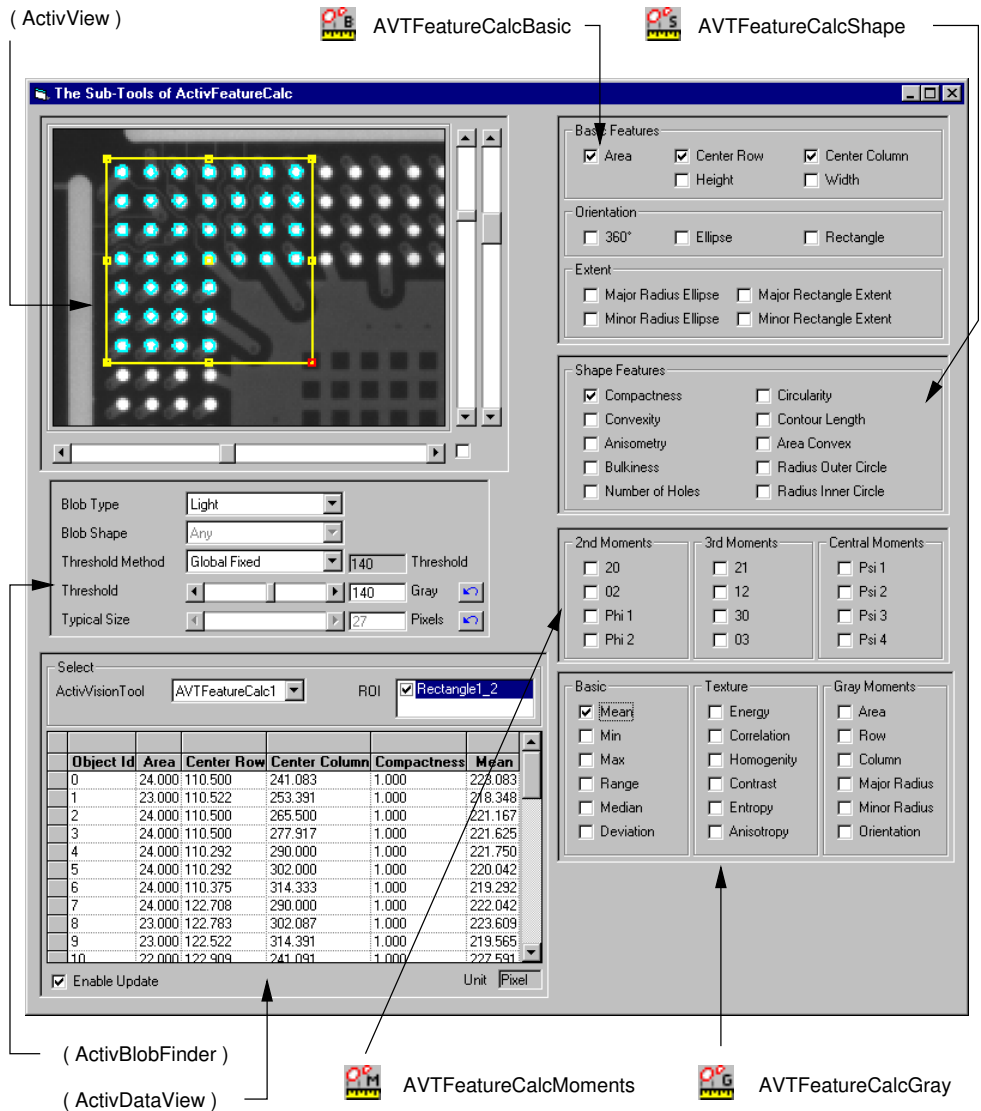


Figure 1.13: The sub-tools of ActivFeatureCalc together with suitable other tools.

## Chapter 2

# Using ActivFeatureCalc

---

This chapter explains how to select the features you want to be calculated. Before this, it briefly shows how to extract the blobs which form the input for ActivFeatureCalc.




<b>2.1</b>	<b>Extracting Objects</b> . . . . .	<b>18</b>
<b>2.2</b>	<b>Selecting Features</b> . . . . .	<b>20</b>

## 2.1 Extracting Objects Using

Before calculating blob features, the blobs must be extracted, of course. This section gives a brief overview of how to do this using AVTBlobFinder, the *master tool* of ActivBlobFinder, and AVTViewROI, which is a *support tool* of ActivView. Please consult the [User's Manual for ActivBlobFinder](#) for detailed information.

### Visual Basic Example

#### Preparation for the following example:

- Open the project `extracting\feature_extracting.vbp`. Alternatively, create a new project and place AVTView, AVTViewROI, and AVTBlobFinder on the form by double-clicking the icons , , and , respectively, with the left mouse button.
- Execute the application (Run > Start or via the corresponding button). Open AVTViewFG by clicking into AVTView with the right mouse button and selecting Image Acquisition in the popup menu. Select the image `bga\bga_01` in the combo box Input File.

The following steps are visualized in [figure 2.1](#).

- ① First, you have to tell AVTViewROI to create an ROI for ActivBlobFinder by selecting the corresponding entry in the combo box `ActivVisionTool`. In this box, all `ActivVisionTools` are listed that have been placed upon the form. By default, the tools are referenced by the name of the corresponding ActiveX control plus a counter to distinguish between multiple instances of a control on the form. In our example, there is only one item in the combo box, `AVTBlobFinder1`.
- ② Next, select the desired shape of the ROI, for example a rectangle.
- ③ To draw the ROI, move the mouse in the image while keeping the left mouse button pressed. Please experiment at this point with the different shapes.
- ④ You can now move the ROI by dragging its pick point in the middle. By dragging the outer pick points you modify its shape. Again, please experiment to get familiar with the ROIs.
- ⑤ Select the extraction method in the combo box `Threshold Method`, e.g., `'Global Fixed'` to use one global threshold.
- ⑥ In the combo box `Blob Type`, select whether the blobs to be extracted are darker or lighter than the background; to extract the balls, select `'Light'`.

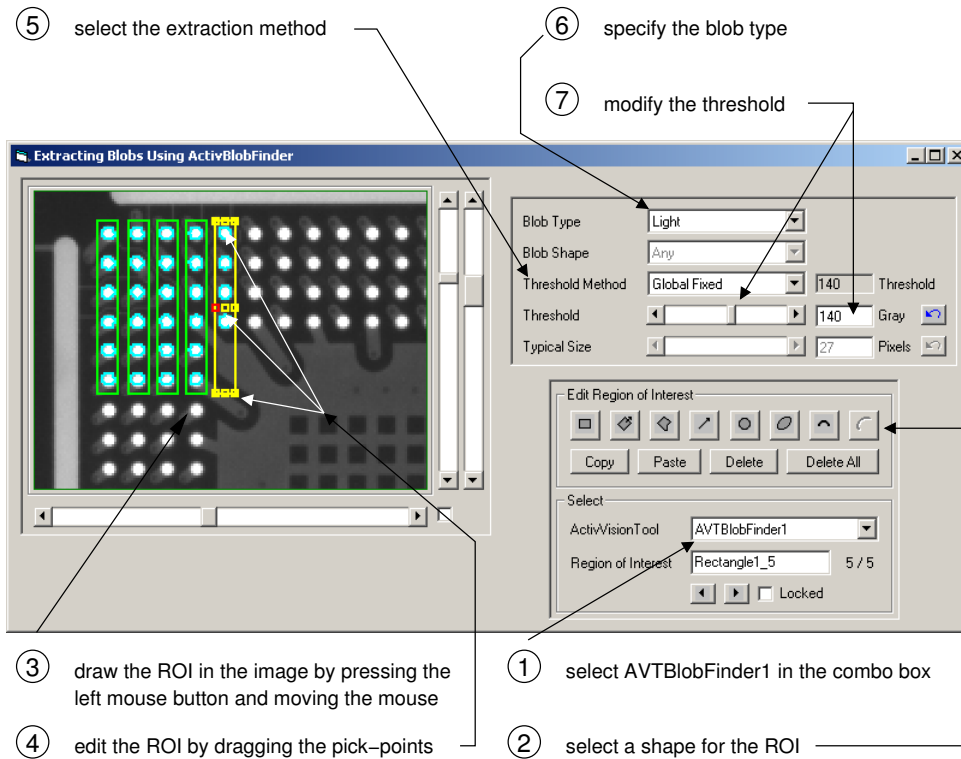


Figure 2.1: Extracting blobs.




- ⑦ As the default value for the threshold is suited to this image, the blobs corresponding to the balls in the image are already extracted, without any action on your part. Experiment with different thresholds by using the slider **Threshold** or by specifying a value in the text field beside it.

## 2.2 Selecting Features Using

You can select which features are to be calculated for each extracted blob using the *support tools* of ActivFeatureCalc. AVTFeatureCalcBasic contains basic shape features like the Area of a blob, AVTFeatureCalcShape more complex shape features like the Compactness or the Number of Holes, AVTFeatureCalcMoments higher moments, and AVTFeatureCalcGray gray value features like Mean or Contrast and mixed gray value shape features like Gray Major Radius.

### Visual Basic Example

#### Preparation for the following example:

- If you worked on the previous example, you may continue using this project. At design time, add AVTFeatureCalc, AVTFeatureCalcBasic, and AVTDataView to the form by double-clicking , , and  with the left mouse button. Note that AVTFeatureCalc is only represented by its icon on the form!  
Otherwise, open the project selecting\feature\_selecting.vbp.
- Execute the application (Run > Start or via the corresponding button) and load the image bga\bga\_01.



**There are two things to note about AVTFeatureCalc**, the master tool of ActivFeatureCalc: First, it does not have a graphical user interface; thus, it is represented by its icon at design time and invisible at run time. If you forget to add it to the form and only add the support tools, they are disabled (and do not calculate any features). Secondly, if you have more than one instance of ActivBlobFinder on the form, you can select to which one AVTFeatureCalc is to be connected via AVTViewConnections (see the User's Manual for ActivView, [section 3.5](#) on page 36).

The following steps are visualized in [figure 2.2](#).

- ① While experimenting with parameters concerning blob extraction it is useful to switch off the update of results in AVTDataView via the corresponding check box!
- ② You can select features to be calculated by checking their box.
- ③ At run time, you can open the other sub-tools of ActivFeatureCalc by clicking into AVT-View or AVTBlobFinder with the right mouse button and selecting the corresponding entry in the popup menu.
- ④ Select the ROIs whose results you want to examine in ActivDataView by checking their box. Their results are then displayed in a table, the columns corresponding to the selected features.

③ open other sub-tools of ActivFeatureCalc via the context menu of AVTView or AVTBlobFinder

② select the features to be calculated

① switch of updating while experimenting with parameters!

④ select the ROIs you are interested in

ROI Id	Object Id	Area	Center Row	Center Column	Major Radius	Minor Radius	Compactness	Contrast	Mean	Deviation
Rectangle1_1 0	25.000	110.040	255.400	3.007	2.606	232.080	33.339			
Rectangle1_1 1	26.000	122.346	255.346	3.058	2.689	1.000	332.365	225.269	39.697	
Rectangle1_1 2	24.000	134.500	255.500	2.769	2.769	1.000	336.510	234.833	28.979	
Rectangle1_1 3	28.000	146.786	255.500	3.000	2.945	1.000	349.080	223.071	43.278	
Rectangle1_2 0	26.000	110.000	267.500	3.101	2.602	1.000	325.048	231.154	34.196	
Rectangle1_2 1	27.000	122.259	267.593	3.013	2.840	1.000	329.694	226.000	39.197	
Rectangle1_2 2	27.000	134.593	267.741	3.013	2.840	1.000	339.417	226.556	39.147	
Rectangle1_2 3	27.000	146.741	267.593	3.013	2.840	1.000	344.315	224.741	41.237	
Rectangle1_3 0	24.000	110.000	279.708	2.857	2.646	1.000	362.052	238.250	32.702	
Rectangle1_3 1	26.000	122.385	279.846	2.950	2.800	1.000	370.875	230.077	39.230	
Rectangle1_3 2	26.000	134.500	280.000	3.101	2.602	1.000	318.202	226.731	38.430	
Rectangle1_3 3	24.000	146.708	280.000	2.857	2.646	1.000	319.417	236.417	32.785	
Rectangle1_4 0	22.000	109.909	291.909	2.751	2.486	1.000	348.852	245.045	23.807	
Rectangle1_4 1	24.000	122.292	292.000	2.857	2.646	1.000	367.906	236.125	32.467	
Rectangle1_4 2	26.000	134.500	292.000	3.101	2.602	1.000	324.529	226.885	37.533	
Rectangle1_4 3	24.000	146.708	292.000	2.857	2.646	1.000	310.958	236.750	29.624	
Rectangle1_5 0	23.000	110.000	304.174	2.703	2.680	1.000	333.685	240.087	28.179	
Rectangle1_5 1	23.000	122.217	304.087	2.857	2.519	1.000	345.380	237.826	29.735	

Figure 2.2: Selecting the features which are to be calculated.

You can also display the distribution of the calculated feature values in a histogram using ActivFeatureHistogram, which can be opened by clicking on AVTView with the right mouse button and selecting Feature Histogram in the appearing context menu..



## Chapter 3

# Combining ActivFeatureCalc with other ActivVisionTools

---

While the previous chapter explained how to select the features to be calculated, this chapter focuses on how to further process them by converting them into other units, and how to evaluate and output the results using other ActivVisionTools. How to access results and evaluations via the programming interface is described in [chapter 4](#) on page 31.

In the corresponding Visual Basic projects, the task is to inspect a ball grid array (*BGA*).

<b>3.1</b>	<b>Converting Results to Other Units</b>	24
<b>3.2</b>	<b>Evaluating Results</b>	26
<b>3.3</b>	<b>Output of Results</b>	28


### 3.1 Converting Results to Other Units Using

In [section 2.2](#) on page 20, positions and extents were calculated in image coordinates, i.e., pixels. Using `AVTViewCalibration`, you can convert these features into other units. Note that for a more accurate calibration you should employ `ActivGeoCalib` (and possibly rectify the image as described in User's Manual for `ActivGeoCalib`, [section 4.2](#) on page 26).


The main idea behind `AVTViewCalibration` is that the user draws a line in the image and tells `ActivVisionTools` that the length of this line is in a certain unit. From this information, `AVTViewCalibration` calculates the size of a pixel (i.e., its height as square pixels are assumed) in this unit, which in its turn can be used to convert features from pixels into the new unit. Note that this conversion only works if the measured objects lie in the same plane, i.e., at the same distance from the camera, as the line whose length was specified.

#### Visual Basic Example

##### Preparation for the following example:

- If you worked on the example in the previous chapter, you may continue using this project. At design time, add `AVTViewCalibration` to the form by double-clicking  with the left mouse button. You may remove `AVTFeatureCalcBasic` and `AVTDataView`.  
Otherwise, open the project `units\feature_units.vbp`.
- Execute the application (Run > Start or via the corresponding button) and load the image `bga\bga_01`.

The following steps are visualized in [figure 3.1](#) (not shown: `AVTBlobFinder`).

- ① You create the line of known length using `AVTViewROI`. First, select `AVTViewCalibration1` in the combo box `ActivVisionTool`.
- ② To start creating the line, click .
- ③ To draw the line, move the mouse in the image while keeping the left mouse button pressed.
- ④ You can modify the line by dragging the pick points; to position it precisely we recommend to zoom the image using the scrollbars of `AVTView`. Place the ROI as shown in the figure; this distance corresponds to 2.7 cm.
- ⑤ Now, switch your attention to `AVTViewCalibration`. Select the desired unit (cm) in the combo box `Unit`. The labels below denoting the unit will automatically switch to the selected unit.

③ to draw the line in the image, move the mouse while pressing the left button

④ edit the line by dragging the pick points

① select AVTViewCalibration1

② click to create a line-shaped ROI

⑤ select a unit in the combo box

⑥ specify the length of the line; the computed pixel size is displayed

⑦ results are converted automatically into the selected unit

Object Id	Area	Center Row	Center Column	Width	Height	Major Radius	Minor Radius	Compactness	Contour Length	Rad
0	0.0031147	2.662	0.052	0.042	0.031	0.027	1.000	0.163	0.02E	
1	0.0031275	2.662	0.052	0.052	0.032	0.028	1.000	0.166	0.02E	

Figure 3.1: Converting measurements into other units.

- ⑥ Then, specify the length of the line (2.7) in the text box Length Line and press **Enter**. AVTViewCalibration now calculates the height of a pixel in mm and displays it in the text box Pixel Height.
- ⑦ Automatically, all calculated positions and extents are converted into the selected unit.


## 3.2 Evaluating Results Using

In the former examples, you have employed ActivVisionTools to calculate features for blobs. In the following, we show how to use ActivDecision to evaluate these results by formulating *conditions* the features have to meet in order to be “okay”. For a detailed description of ActivDecision please consult the [User’s Manual for ActivDecision](#).

In the example task, the extracted balls must be circular (measured by the feature Compactness) with a radius of at least 0.25 mm (measured by the features Area, Major Radius, resultFeatureMinorElleExtent). Surface disturbances can be detected by analyzing the Mean gray value. Note that only a part of the BGA is inspected; each ROI should contain 4 balls.

### Visual Basic Example

#### Preparation for the following example:

- If you worked on the previous example, you may continue using this project. At design time, add AVTDecision to the form by double-clicking  with the left mouse button.  
Otherwise, open the project decisions\feature\_decisions.vbp.
- Execute the application and load the image sequence bga\bga1.seq.

The following steps are visualized in [figure 3.2](#) (not shown: AVTView, AVTBlobFinder).

- ① The main functionality of ActivDecision is presented by its *support tools*, which can be opened via clicking on AVTDecision with the right mouse button. The *master tool* displays the overall evaluation of the application.
- ② To view the current feature and evaluation results check  Enable Update in AVT-DecisionViewResults.
- ③ ActivDecision lets you compare the value of an individual object, ROI, or tool feature with two boundary values, a minimum and a maximum value. You can formulate conditions for features by specifying values in the columns Min and Max and selecting a comparison mode in the column Operation. If you select None, the feature is not evaluated.
- ④ Those features which meet their condition appear in green, the others in red. If at least one feature is “not okay”, the whole object, ROI, or tool is evaluated as “not okay” as well. Analogously, the overall evaluation of application, which is visualized by AVT-Decision, depends on the tool evaluations.

[Figure 3.2](#) shows suitable conditions for the example task. Step through the image sequence and examine the evaluations; in some of the images, balls are defect or missing.

① open the support tools via the context menu (right mouse click)

② first, enable the update of results

③ formulate conditions for objects, ROIs or tools

④ the evaluations are displayed immediately

⑤ specify default conditions

⑥ check this box to show the used parameters

Tool ROI Object	Name	Value	Min	Max	Interpretation	Operation
AVTFeatureCalc1	Objects Tool	19	0	10000	Number	None
	Good Objects Tool	18	0	10000	Number	None
	Bad Objects Tool	1	0	10000	Number	None
	Good ROIs	3	0	10000	Number	None
	Bad ROIs	2	0	0	Number	= Max
Rectangle1_1	Objects ROI	4	4	4	Number	= Max
	Good Objects ROI	3	0	10000	Number	None
	Bad Objects ROI	1	0	0	Number	= Max
Object 0	Area	0.261	0.215	0.305	Number	Inside
	Major Radius	0.301	0.250	0.325	Number	Inside
	Minor Radius	0.275	0.250	0.325	Number	Inside
	Compactness	1.000	1.000	1.100	Number	Inside
	Mean	226.042	200.000	255.000	Number	>= Min
Object 1	Area	0.174	0.215	0.305	Number	Inside

Tool ROI Object	Name	Min	Max	Interpretation	Operation
AVTFeatureCalc1	Objects ROI	4	4	Number	= Max
	Good Objects ROI	0	10000	Number	None
	Bad Objects ROI	0	0	Number	= Max
	Area	0.215	0.305	Number	Inside
	Major Radius	0.250	0.325	Number	Inside
	Minor Radius	0.250	0.325	Number	Inside
	Compactness	1.000	1.100	Number	Inside
	Mean	200.000	255.000	Number	>= Min

Figure 3.2: Formulating conditions to evaluate results.


- ⑤ If many similar objects are extracted which all should meet the same conditions, you can specify default conditions using `AVTDecisionViewDefaults`. Defaults can be set per tool or per ROI; ROI defaults override tool defaults, and individual conditions override defaults.
- ⑥ If you check  `Substitute Default`, the entries marked `Default` are substituted by their actual content.

### 3.3 Output of Results Using

Using ActivFile, you can write the results and the evaluations to a log file. How to access results via the programming interface is described in [section 4.2](#) on page 34, how to output them via a serial interface or a digital I/O board in the [User's Manual for ActivSerial](#) and the [User's Manual for ActivDigitalIO](#), respectively.

#### Visual Basic Example

##### Preparation for the following example:

- If you worked on the previous example, you may continue using this project. At design time, add AVTOutputFile by double-clicking  .  
Otherwise, open the project output\feature\_output.vbp.
- Execute the application and load the image sequence bga\bga1.seq.

The following steps are visualized in [figure 3.3](#) (not shown: AVTView and AVTDecision).

- ① By clicking on **Select**, you can open a file selector box to choose a file name for the log file, which will then appear in the text field beside the button. By pressing **Clear File**, you can clear the content of the selected file.
- ② By checking  **Enable Writing** you enable the writing mode.
- ③ You can open the ActivFile's two dialogs **DialogFileOptions** and **DialogOutputDataSelect** by clicking **File Options** and **Data Selection**, respectively.
- ④ In **DialogFileOptions**, you can choose between two file formats: Standard text files (suffix .txt) and the so-called *comma-separated values* (suffix .csv) which can be used as an input to Microsoft Excel. Furthermore, you can select a delimiter.
- ⑤ In the same dialog you can limit the size of the log file in form of the number of *cycles* that are to be recorded. A cycle corresponds to one processing cycle from image input to the evaluation and output of results. If you use this option, ActivFile creates two log files and switches between them, thus assuring that you can always access (at least) the results of the last  $N$  cycles,  $N$  being the specified number of cycles.
- ⑥ By pressing **Estimate**, you can let ActivFile estimate the size of one cycle. Note that you must first select the output data in order to get meaningful results!
- ⑦ In the left part of **DialogOutputDataSelect**, you can navigate through the result hierarchy similarly to **ActivDecision**.

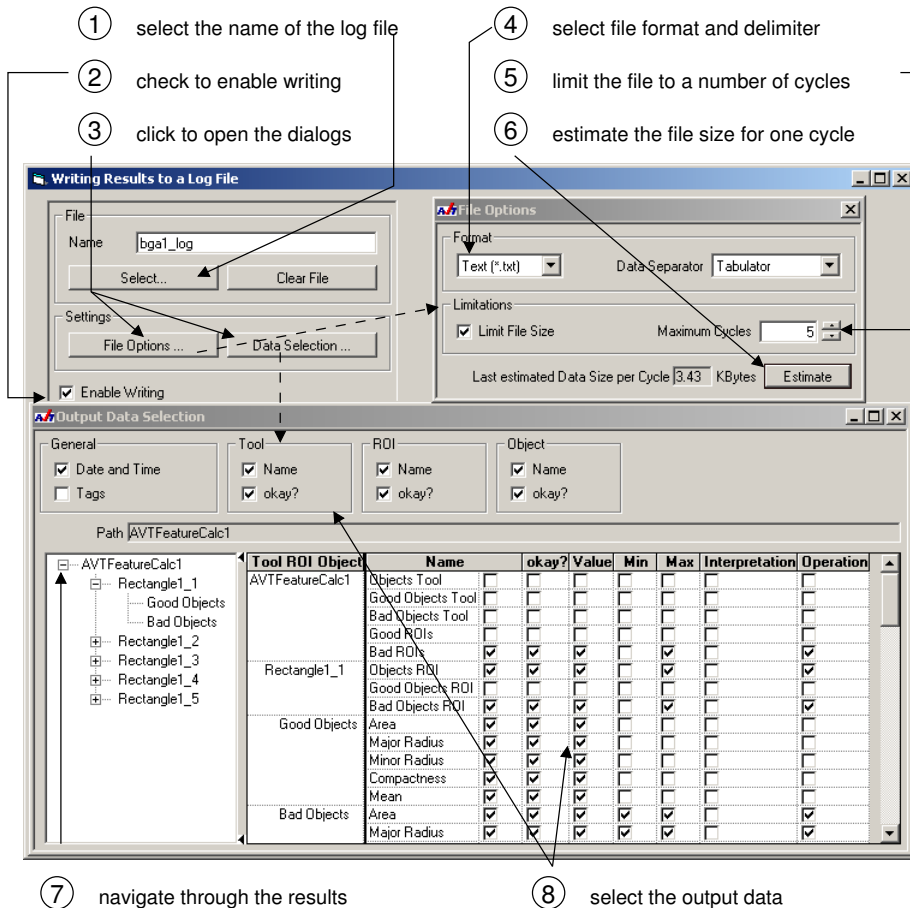


Figure 3.3: Customizing log files.

- ⑧ In the right part of DialogOutputDataSelect, choose the output data by checking the corresponding boxes. You may output different items depending on the evaluation of an object. By clicking on the column labels with the right mouse button you can check or uncheck all boxes in the column; similarly, you can check or uncheck whole rows or all rows of a certain tool.

If you now step through the image sequence by clicking `Single` in AVTviewFG, the log file is created. Figure 3.4 shows part of an example log file.

```

09/10/04 11:50:29
AVTFeatureCalc1 no
  Bad ROIs no 2 0 = Max
Rectangle1_1 no
  Objects ROI yes 4 4 = Max
  Bad Objects ROI no 1 0 = Max
0 yes
  Area yes 0.260819
  Major Radius yes 0.300935
  Minor Radius yes 0.275264
  Compactness yes 1.000000
  Mean yes 226.041667
1 no
  Area no 0.173879 0.215000 0.305000 Inside
  Major Radius yes 0.305259 0.250000 0.325000 Inside
  Minor Radius no 0.211016 0.250000 0.325000 Inside
  Compactness no 1.219212 1.000000 1.100000 Inside
  Mean no 192.750000 200.000000 255.000000 >= Min
2 yes
  Area yes 0.271686
  Major Radius yes 0.313434
  Minor Radius yes 0.271718
  Compactness yes 1.000000
  Mean yes 222.800000
3 yes
  Area yes 0.239084
  Major Radius yes 0.299839
  Minor Radius yes 0.252878
  Compactness yes 1.000000
  Mean yes 224.000000

```

Figure 3.4: Part of an example log file.

# Chapter 4

## Tips & Tricks

---

This chapter contains additional information that facilitates working with `ActivFeatureCalc`, e.g., how to customize the appearance of an `ActivFeatureCalc` application in the two execution modes. Furthermore, it shows how to access the calculated features and the evaluations directly via the programming interface of `ActivVisionTools`.

<b>4.1</b>	<b>Customizing the Two Execution Modes</b>	<b>32</b>
<b>4.2</b>	<b>Accessing Results Via the Programming Interface</b>	<b>34</b>
4.2.1	Calculated Features	35
4.2.2	Evaluation Results	41



## 4.1 Customizing the Two Execution Modes Using

In an ActivVisionTools application you can switch between two execution modes: the *configuration mode* and the *application mode*. The former should be used to setup and configure an application, the latter to run it. ActivView's *support tools* AVTViewExecute and AVTViewConfigExec allow you to switch between the two modes and to customize the behavior of an ActivVisionTools application in the two execution modes, e.g., display live images only in the *configuration mode* to setup your application, but then switch it off in the *application mode* to speed up the application. A third sub-tool, AVTViewExecuteSimple, provides a single button to start/stop the application.

With the help of AVTViewStatus, another *support tool* of ActivView, you can inspect the current status of your application.

### Visual Basic Example

#### Preparation for the following example:

- If you worked on the example in the previous chapter, you may continue using this project. At design time, add AVTViewExecuteSimple and AVTViewStatus to the form by double-clicking the icons  and .
- Otherwise, open the project usermodes\feature\_usermodes.vbp.
- Execute the application and load the image sequence bga\bga1.seq.

The following steps are visualized in [figure 4.1](#) (not shown: AVTBlobFinder).

- ① Open AVTViewExecute and AVTViewConfigExec by clicking on AVTView with the right mouse button and selecting Execution and Execution Parameters.
- ② Switch between the two execution modes via AVTViewExecute's combo box Mode.
- ③ To execute one cycle, press `Single`. With the other two buttons you can let the application run continuously and stop it again. By default, AVTViewExecuteSimple starts and stops an application; how to change its behavior to a single-step button is described in the User's Manual for ActivView, [section 3.4](#) on page 34.
- ④ For each of the two execution modes, you can choose what is to be displayed by checking the corresponding boxes in AVTViewConfigExec. Please refer to the User's Manual for ActivBlobFinder, [section 4.1](#) on page 34, for more information about adapting the display, e.g., how to choose colors for ROIs and results. Furthermore, you can specify if images can be dragged to the image window and whether ROIs can be modified in the two modes; by default, this is disabled in the *application mode* to prevent you from accidentally moving or deleting an ROI.

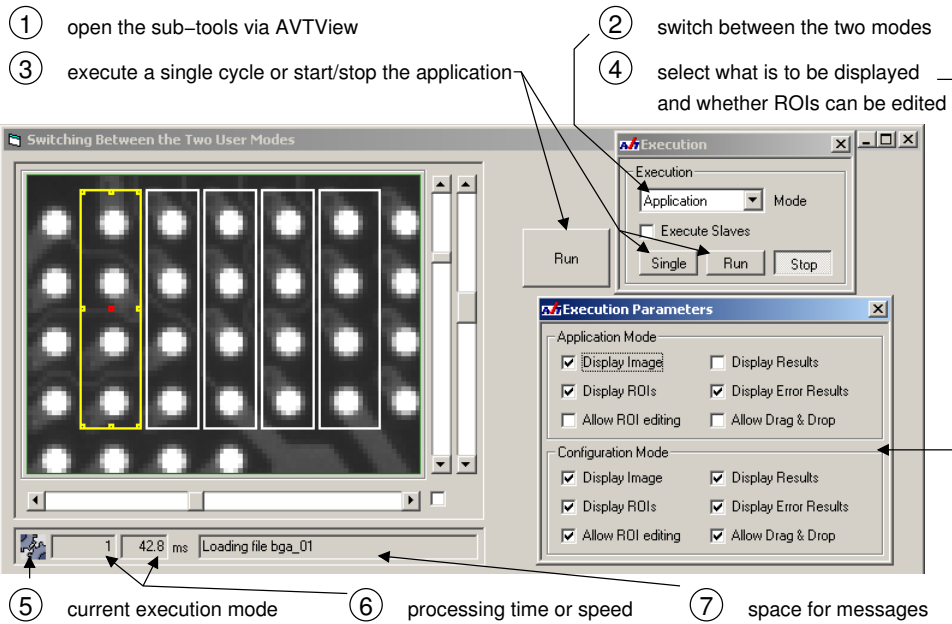



Figure 4.1: Customizing and switching between the two execution modes.

- ⑤ In AVTViewStatus, an icon indicates the current execution mode of the application. In the mode , the application does not perform any processing and waits for your interaction. If you start the continuous mode the cogwheels rotate; any interaction on your part is stored in the event queue and processed after the current cycle is finished. If the cursor gets “busy”, ActivVisionTools has started a particularly time-consuming operation, e.g., connecting to an image acquisition device. Any interaction on your part is then deferred to the end of this operation.
- ⑥ AVTViewStatus also shows the number of processed cycles and the time needed for the last processing cycle.
- ⑦ AVTViewStatus display two types of messages: Informative messages describe, e.g., what the application is doing while it is “busy”, while error messages indicate errors that prevent the application from working correctly, e.g., the failure to connect to an image acquisition device. If AVTViewStatus is not added to an application, error messages are displayed in popup dialogs.

More information about AVTViewStatus, e.g., how to modify its appearance, can be found in the User’s Manual for ActivView, [section 3.3](#) on page 32.

## 4.2 Accessing Results Via the Programming Interface

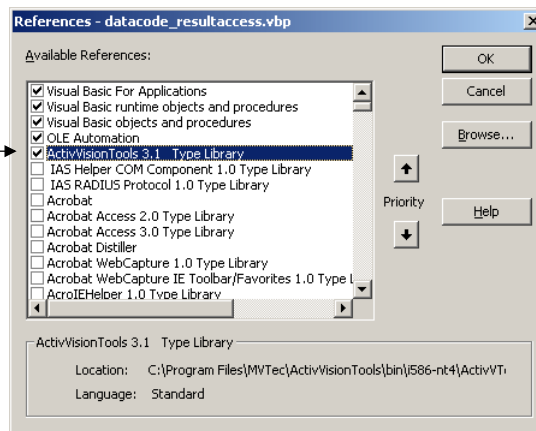
The previous chapters and sections showed how to use ActivVisionTools interactively, i.e., via the graphical user interfaces presented by the underlying ActiveX controls. In this mode, you can develop the image processing part of your machine vision application rapidly and easily, without any programming. However, there is more to ActivVisionTools than the graphical user interfaces: Because ActivVisionTools comes as a set of ActiveX controls, it provides you with an open programming interface, thereby offering full flexibility.

In this section, we show how to access the blob features and evaluation results via the programming interface. With this, you can, e.g., realize an application-specific graphical user interface, perform additional processing on the results, or send results to a special output device. Detailed information about the programming interface can be found in the **Reference Manual**.

As in the previous sections, the examples stem from Visual Basic 6.0; if the (ActivVisionTools-specific) code differs in Visual Basic .NET, the corresponding lines are also shown (for the first appearance only). For other .NET languages or C++, please refer to the Advanced User's Guide for ActivVisionTools, [section 1.2.3](#) on page 5 and [section 1.3.4](#) on page 28, respectively. Please note that we assume that readers of this part have at least a basic knowledge of Visual Basic.



To work with the programming interface, in VB 6.0 you must first **add the ActivVisionTools type library to the project's references** by checking the box labeled `ActivVisionTools` Type Library in the menu dialog `Project > References`. In Visual Basic .NET, the reference is added automatically.



### 4.2.1 Accessing Calculated Features

The principal idea behind accessing the results of an `ActivVisionTool` is quite simple: When a tool has finished its execution, it raises an event called `Finish`, sending its results as a parameter. If you want to access the results, all you have to do, therefore, is to create a corresponding event procedure which handles the event.

Within the Visual Basic environment, you can create event procedures very easily as shown in [figure 4.2](#): In the header of the form's code window there are two combo boxes. Select the instance of `AVTFeatureCalc` (by default called `AVTFeatureCalc1`) in the left combo box. The right combo box then lists all events available for this object; when you select `Finish`, the event procedure is created automatically. Within this procedure, the measurement results are now accessible via the object variable `atToolResults`.

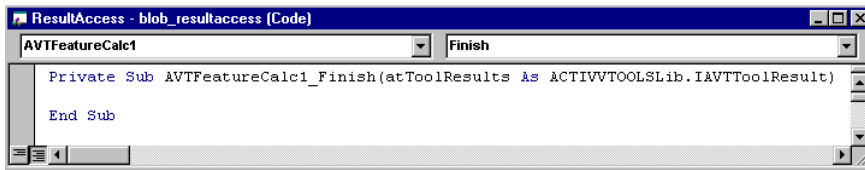


Figure 4.2: Creating a procedure to handle the event `Finish` .

`atToolResults` contains the result data for all ROIs of your instance of `AVTFeatureCalc`. The current number of ROIs can be queried via

```
Dim iNumROIs As Integer
iNumROIs = atToolResults.ROIEnum
```

In Visual Basic .NET , the event handler has a different signature:

```
Private Sub _
    AxAVTFeatureCalc1_Finish(ByVal sender As Object, _
        ByVal e As _
            AxActivVTools._AVTFeatureCalc_FinishEvent) _
    Handles AxAVTFeatureCalc1.Finish
```

A first difference is that tool names start the prefix `Ax`, i.e., `AVTFeatureCalc` becomes `AxAVTFeatureCalc`. The main difference, however, is that the tool results are not directly passed; instead, they are encapsulated in the parameter `e`. From there, they can be extracted with the following lines:

```
Dim atToolResults As AVTToolResult
atToolResults = e.atToolResults
```

To use classes like `ACTIVVTOOLSlib.AVTToolResult` without the namespace `ACTIVVTOOL-`

SLib as in the code above, you must import this namespace by inserting the following line at the very beginning of the code (more information about importing namespaces can be found in the Advanced User's Guide for ActivVisionTools in [section 1.2.4.5](#) on page 12):

```
Imports ACTIVVTOOLSlib
```

The results of a certain ROI can be accessed by specifying its name in a call to the method `ROIResult`, or by specifying its index in a call to the method `ROIResults`. The following code uses the latter method to access the first ROI of `AVTFeatureCalc1`:

```
Dim atROIResult As AVTROIRResult

Set atROIResult = atToolResults.ROIResults(0)
```

Now, we can, e.g., query the number of objects extracted in the ROI via

```
Dim iNumObjects As Integer

iNumObjects = atROIResult.ObjectNum
```

Actual results for an object, i.e., the calculated values of features like Major Extent or Area, can be accessed by specifying the feature of interest and the ID of the object in a call to the method `ObjectValue` of `ACTIVVTOOLSlib.AVTROIResult`. The feature handles are available as methods of the corresponding tool, e.g., `AVTFeatureCalc1.FeatureHandleArea` being the handle for the calculated area.

The following code fragment uses another method of `ACTIVVTOOLSlib.AVTROIResult`, `ObjectValues`, which returns the values of all objects for the specified feature in an array, to calculate the mean blob area:

```
Dim handleArea As Integer, i As Integer
Dim vAreaArray As Variant
Dim dSumArea As Double, dMeanArea As Double

handleArea = AVTFeatureCalc1.FeatureHandleArea
vAreaArray = atROIResult.ObjectValues(handleArea)
dSumArea = 0
For i = 0 To iNumObjects - 1
    dSumArea = dSumArea + vAreaArray(i)
Next
dMeanArea = dSumArea / iNumObjects
```

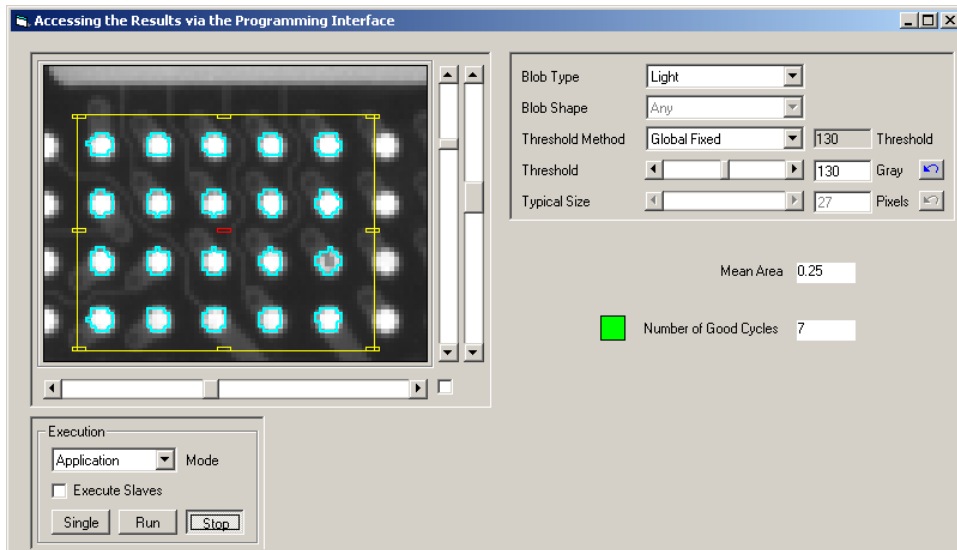


Figure 4.3: Accessing and displaying the calculated features.

A general difference **in Visual Basic .NET** is that instead of the type `Variant` you must use `Object` when accessing multiple values:

```
Dim vAreaArray As Object
```

The `ActivVisionTools` distribution includes the example Visual Basic project `resultaccess\feature_resultaccess.vbp`, which uses the methods described above to inspect BGAs similarly to the application introduced in the previous chapter. The task is to calculate the mean area of all balls; on contrast to the previous chapter only one ROI is used here (see [figure 4.3](#)). The example project is already configured, just start it and click the button `Run` in `AVTViewExecute`.

Besides accessing the calculated features, the project code contains additional functionality which is explained briefly in the following. Note that the code is only shown for Visual Basic 6.0; a Visual Basic .NET application with result access can be found in the directory `examples\dotnet\vb/blob_results`.

First of all, the calculated area is only to be read and displayed if all balls have the correct size and extent:

```

For i = 0 To iNumObjects - 1
    dArea = atROIResult.ObjectValue(handleArea, i)
    dMajorRadius = atROIResult.ObjectValue(handleMajorRadius, i)
    dMinorRadius = atROIResult.ObjectValue(handleMinorRadius, i)
    If dArea >= 0.215 And dArea <= 0.305 And _
        dMajorRadius >= 0.25 And dMajorRadius <= 0.325 And _
        dMinorRadius >= 0.25 And dMinorRadius <= 0.325 Then
        ' sum Area
        dSumArea = dSumArea + dArea
    Else
        TextMeanArea.Caption = "---"
        Call SetAlarm
        Exit Sub
    End If
Next
dMeanArea = dSumArea / iNumObjects
TextMeanArea.Caption = Format(dMeanArea, "Fixed")

```

If the conditions are not met, the function `SetAlarm` stops the application by setting `AVTView`'s property `RunState` to 'False' and switches the color of the element beside the number of good cycles to red. The function `ClearAlarm` resets the color to green.

```

Private bIsError As Boolean

Private Function SetAlarm()
    AVTView1.RunState = False
    Light.BackColor = vbRed
    bIsError = True
End Function

Private Function ClearAlarm()
    Light.BackColor = vbGreen
    bIsError = False
End Function

```

As a further condition, the correct number of balls must be present.

```

iNumObjects = atROIResult.ObjectNum
If Not (iNumObjects = 5 * 4) Then
    TextMeanArea.Caption = "---"
    Call SetAlarm
    Exit Sub
End If

```

Below the mean area, the number of BGAs which passed the inspection is to be displayed. For this one has to keep in mind that `AVTFeatureCalc` is executed not only when the next image is grabbed but also whenever you modify its ROI (s) or parameters. To distinguish the two cases

an event raised by AVTView at the start of each execution cycle can be used to set a variable called `bIsNewCycle`:

```
Private bIsNewCycle As Boolean
Private Sub AVTView1_CycleStart()
    bIsNewCycle = True
End Sub
```

Before increasing the counter of good cycles within the handler for AVTFeatureCalc's event `Finish`, this variable is checked (and immediately reset). You can test this behavior by modifying the blob extraction parameters of AVTBlobFinder or AVTBlobFinderProcess: The new mean area is displayed, while the number of good cycles remains constant.

```
If bIsNewCycle = True Then
    iNumGoodCycles = iNumGoodCycles + 1
    bIsNewCycle = False
    TextNumCycles.Caption = iNumGoodCycles
End If
```

When using the programming interface of ActivVisionTools, you leave the safe world of the graphical user interfaces where all input is checked for validity automatically. In contrast, if you try to access a non-existent object or result via the programming interface, a run-time error is caused which terminates your application! To avoid this, you can use the Visual Basic error handling mechanisms, i.e., set up an error handler which examines any occurring error and reacts in a suitable way. In the example project, if an error is caused by the result access, a dialog with the error description pops up and the function `SetAlarm` is called.

```
Private Sub AVTFeatureCalc1_Finish(atToolResults As _
                                ACTIVVTOOLSlib.IAVTTToolResult)
    ' variable declarations
On Error GoTo ErrorHandler
    ' procedure body
Exit Sub

ErrorHandler:
    Dim sTitle As String
    If Left(Err.Source, 11) = "ActivVTools" Then
        sTitle = "ActivVisionTools Error"
    Else
        sTitle = "Runtime Error " & CStr(Err.Number)
    End If
    Call MsgBox(Err.Description, vbExclamation, sTitle)
    Call SetAlarm
End Sub
```

To view the effect of the error handler, de-select the feature Area in AVTFeatureCalcBasic.

By placing the following code at the beginning of `AVTFeatureCalc1_Finish`, the actual result access is restricted to the *application mode*. With this mechanism you can setup the vision part of your application in the configuration mode without having to worry about run-time errors.

```
If Not AVTView1.ExecutionMode = eApplicationMode Then
    TextMeanArea.Caption = "---"
    Exit Sub
End If
```

### 4.2.2 Accessing Evaluation Results

The evaluation results can be accessed similarly to the measurement results; in fact, they are even stored in the same object. However, to access the evaluation results you now have to wait for `ActivDecision` to finish, i.e., create the following event procedure:

```
Private Sub AVTDecision1_Finish(atToolResults As Collection)

End Sub
```

Note that **you will get a run-time error if you try to access evaluation results before `ActivDecision` has finished** (e.g., in the handler for `AVTFeatureCalc`'s event `Finish`!



Because `ActivDecision` can evaluate the results of more than one tool, the event handler provides you with a `Collection` of tool results. The following code fragment searches the collection for the results of `AVTFeatureCalc1` and “stores” them in `atFeatureResult`, or exits if no results are found:

```
Dim atToolResult As AVTToolResult
Dim atFeatureResult As AVTToolResult
Dim bFeatureResultsFound As Boolean

bFeatureResultsFound = False
For Each atToolResult In atToolResults
    If atToolResult.Name = "AVTFeatureCalc1" Then
        Set atFeatureResult = atToolResult
        bFeatureResultsFound = True
    End If
Next
If bFeatureResultsFound = False Then
    Exit Sub
End If
```

In Visual Basic .NET, the event procedure has the following signature:

```
Private Sub AxAVTDecision1_Finish(ByVal sender As System.Object, _
    ByVal e As _
        AxActivVTools._AVTDecision_FinishEvent) _
    Handles AxAVTDecision1.Finish
```

Again, the tool results are encapsulated in the parameter `e`. They can be extracted as follows; note the use of `VBA.Collection` instead of `Collection`!

```
Dim atToolResults As VBA.Collection
atToolResults = e.atToolResults
```

As already remarked in the previous section, tool names are prefixed with `Ax`, thus you must

search for the results of AxAVTFeatureCalc1:

```
If atToolResult.name = "AxAVTFeatureCalc1" Then
```

You can query the overall evaluations at different levels, tool, ROI, or object:

```
Dim atROIResult As AVTROIRResult
Dim bToolIsOK As Boolean, bROIIsOK As Boolean, bObjIsOK As Boolean

Set atROIResult = atFeatureResult.ROIResults(0)

bToolIsOK = atFeatureResult.Evaluation
bROIIsOK = atROIResult.Evaluation
bObjIsOK = atROIResult.ObjEvaluation(0)
```

Furthermore, you can access the evaluation of individual features like the measured area of the blob with the ID 0, but also of tool features (e.g., the evaluation of the number of “bad” ROIs) or of ROI features (e.g., the evaluation of the number of objects) via the corresponding feature handle. In contrast to object features, the handles for tool and ROI features are available as properties of ACTIVVTOOLSlib.AVTToolResult.

```
Dim handleArea As Integer, handleBadROIs As Integer
Dim handleObjectsInROI As Integer
Dim bAreaIsOK As Boolean, bBadROIsIsOK As Boolean
Dim bObjectsInROIIsOK As Boolean

Debug.Print "Object 0 OK? " & bObjIsOK

handleArea = AVTFeatureCalc1.FeatureHandleArea
handleBadROIs = atFeatureResult.FeatureHandleBadROIs
handleObjectsInROI = atFeatureResult.FeatureHandleObjectsROI

bAreaIsOK = atROIResult.ObjFeatureEvaluation(handleArea, 0)
bBadROIsIsOK = atFeatureResult.FeatureEvaluation(handleBadROIs)
bObjectsInROIIsOK = atROIResult.FeatureEvaluation(handleObjectsInROI)
```

The ActivVisionTools distribution includes the example Visual Basic project evalaccess\feature\_evalaccess.vbp, which uses these methods to extend the application described in the previous section. Now, ActivDecision is used to check the extent of the extracted balls as described in [section 3.2](#) on page 26. The task is to stop the application in case the overall evaluation shows an error and to display the cause of the error (see [figure 4.4](#)). The example project is already configured, just start it and click the button **Run** in AVTViewExecute.

Besides accessing the evaluation results, the project code contains additional functionality which is explained briefly in the following (again only for Visual Basic 6.0!). First of all, the main If - Then - Else clause around the display of calculated features now tests the overall evaluation

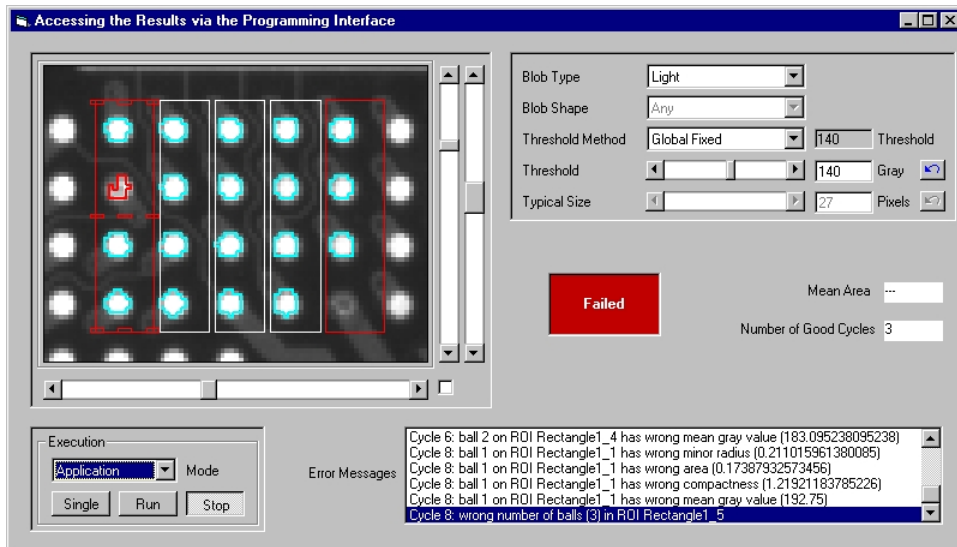


Figure 4.4: Accessing and displaying the evaluation results.

of AVTFeatureCalc1:

```
If atFeatureResult.Evaluation = True Then
    ' access and display the mean area
Else
    ' examine cause of error more closely
End If
```

First, the program checks whether the application contains 5 ROIs. Otherwise, the application is stopped and an error message is displayed. Test this behavior by creating an additional ROI.

```
iNumROI = atFeatureResult.ROIEnum
If Not iNumROI = 5 Then
    TextMeanArea.Caption = "----"
    sErrMsg = "Cycle " & iNumCycles & ": Corrupt application!"
    Call DisplayMessage(sErrMsg)
    ' evaluation of Activfeature corrupt -> exit
    Call SetAlarm
    Exit Sub
End If
```

If the overall evaluation of AVTFeatureCalc1 is “not okay”, the cause of error is investigated by checking the evaluations in a loop over all ROIs:

```
Dim atROIResult(5) As AVTROIRResult

For i = 0 To 4
    Set atROIResult(i) = atFeatureResult.ROIResults(i)
Next

For i = 0 To 4
    If atROIResult(i).Evaluation = False Then
        ' check cause of error
    End If
Next
```

The first possible cause of an error is that the ROI contains the wrong number of balls:

```
If atROIResult(i).FeatureEvaluation(handleObjectsInROI) = False Then
    sErrMsg = "Cycle " & iNumCycles & _
              ": wrong number of balls (" & _
              atROIResult(i).Value(handleObjectsInROI) & _
              ") in ROI " & sROINames(i)
    Call DisplayMessage(sErrMsg)
End If
```

Independently of this possible cause of error, features of the individual objects can have been evaluated as “not okay” (only one feature shown below). Note how you can access the indices of all objects evaluated as “not okay”:

```
Dim badObject As Variant

For Each badObject In atROIResult(i).BadObjectIndices
    ' check evaluation of features
    If atROIResult(i).ObjFeatureEvaluation(handleMinorRadius, _
                                           badObject) = False Then
        sErrMsg = "Cycle " & iNumCycles & _
                  ": ball " & badObject & " on ROI " & sROINames(i) & _
                  " has wrong minor radius (" & _
                  atROIResult(i).ObjectValue(handleMinorRadius, _
                  badObject) & ")"
        Call DisplayMessage(sErrMsg)
    End If
    ' check other features
Next
```

The error message also contains the number of the cycle in which the error occurred. The corresponding counter is incremented in the handler for AVTView’s event CycleStart which was introduced already in the previous section:

```
Private iNumCycles As Integer

Private Sub AVTView1_CycleStart()
    iNumCycles = iNumCycles + 1
End Sub
```



# Appendix A

## Mathematical Background of the Features

---

This chapter contains the equations for the more complex features.

<b>A.1 Basic Moments</b> . . . . .	48
<b>A.2 The Equivalent Ellipse</b> . . . . .	48
<b>A.3 Complex Moments</b> . . . . .	49
<b>A.4 Gray Value Moments</b> . . . . .	49
<b>A.5 Gray Value Features</b> . . . . .	50
<b>A.6 Texture Features</b> . . . . .	50

## A.1 Basic Moments

The (j,k)th moment of a blob  $B$ , denoted  $M_{jk}$  is defined as

$$M_{jk} = \sum_{(r,c) \in B} r^j c^k \quad (\text{A.1})$$

with  $r$  and  $c$  being the row and column coordinates of the pixels. The moment  $M_{00}(B)$  is identical with the *area*  $A$  of the blob  $B$ :

$$A = M_{00} = \sum_{(r,c) \in B} 1 \quad (\text{A.2})$$

The *center of gravity*  $(\bar{r}, \bar{c})$  is calculated from the zeroeth and first moments:

$$\bar{r} = \frac{M_{10}}{M_{00}} \quad \bar{c} = \frac{M_{01}}{M_{00}} \quad (\text{A.3})$$

Typically, higher moments are calculated relative to the center of gravity to make them translation invariant; these moments are then called *centralized moments*. The centralized (j,k)th moment of a blob  $B$ , denoted  $m_{jk}$  is defined as

$$m_{jk} = \sum_{(r,c) \in B} (r - \bar{r})^j \cdot (c - \bar{c})^k \quad (\text{A.4})$$

To make moments scale invariant as well they can be divided by the blob area  $A$  and are then called *normalized moments*. The normalized (j,k)th moment of a blob  $B$ , denoted as  $\mu_{jk}$  is defined as

$$\mu_{jk} = \frac{1}{A^{j+k}} \cdot m_{jk} = \frac{1}{A^{j+k}} \cdot \sum_{(r,c) \in B} (r - \bar{r})^j \cdot (c - \bar{c})^k \quad (\text{A.5})$$

## A.2 The Equivalent Ellipse

The parameters of the equivalent ellipse, i.e., the major radius  $R_{maj}$ , the minor radius  $R_{min}$ , and the orientation  $\varphi_E$ , are calculated from the normalized central moments as follows:

$$R_{maj} = \sqrt{2A \cdot (\mu_{20} + \mu_{02} + \sqrt{(\mu_{20} - \mu_{02})^2 + 4 \cdot \mu_{11}^2})} \quad (\text{A.6})$$

$$R_{min} = \sqrt{2A \cdot (\mu_{20} + \mu_{02} - \sqrt{(\mu_{20} - \mu_{02})^2 + 4 \cdot \mu_{11}^2})} \quad (\text{A.7})$$

$$\varphi_E = \arctan \left( \frac{-2\mu_{11}}{\mu_{20} - \mu_{02} + \sqrt{(\mu_{20} - \mu_{02})^2 + 4 \cdot \mu_{11}^2}} \right) \quad (\text{A.8})$$

## A.3 Complex Moments

Based on the basic moments, ActivFeatureCalc provides more complex moments which are invariant under other affine transformations, e.g., rotation or stretching.

$$\text{2nd Moments Phi 1} = \mu_{20} + \mu_{02} \quad (\text{A.9})$$

$$\text{2nd Moments Phi 2} = (\mu_{20} - \mu_{02})^2 + \mu_{11} \quad (\text{A.10})$$

$$\text{Central Moments Psi 1} = \mu_{20} \cdot \mu_{02} - \mu_{11}^2 \quad (\text{A.11})$$

$$\begin{aligned} \text{Central Moments Psi 2} = & A^2 \cdot ((\mu_{30}\mu_{03} - \mu_{21}\mu_{12})^2 - \\ & 4 \cdot (\mu_{30}\mu_{12} - \mu_{21}^2)(\mu_{21}\mu_{03} - \mu_{12}^2)) \end{aligned} \quad (\text{A.12})$$

$$\begin{aligned} \text{Central Moments Psi 3} = & A \cdot (\mu_{20} \cdot (\mu_{21}\mu_{03} - \mu_{12}^2) - \mu_{11} \cdot (\mu_{30}\mu_{03} - \\ & \mu_{21}\mu_{12}) + \mu_{02} \cdot (\mu_{30}\mu_{12} - \mu_{21}^2)) \end{aligned} \quad (\text{A.13})$$

$$\begin{aligned} \text{Central Moments Psi 4} = & A \cdot (\mu_{30}^2\mu_{02}^3 - 6\mu_{30}\mu_{21}\mu_{11}\mu_{02}^2 + \\ & 6\mu_{30}\mu_{12}\mu_{02}(2\mu_{11}^2 - \mu_{20}\mu_{02}) + \mu_{30}\mu_{03}(6\mu_{20}\mu_{11}\mu_{02} - \\ & 8\mu_{11}^3) + 9\mu_{21}^2\mu_{20}\mu_{02}^2 - 18\mu_{21}\mu_{12}\mu_{20}\mu_{11}\mu_{02} + \\ & 6\mu_{21}\mu_{03}\mu_{20}(2\mu_{11}^2 - \mu_{20}\mu_{02}) + 9\mu_{12}^2\mu_{20}^2\mu_{02} - \\ & 6\mu_{12}\mu_{03}\mu_{11}\mu_{20}^2 + \mu_{03}^2\mu_{20}^3) \end{aligned} \quad (\text{A.14})$$

## A.4 Gray Value Moments

The moments mixing spatial and gray value information, called *gray value moments* are defined in analogy to the basic moments in [appendix A.1](#) on page 48, but now the gray value of a pixel, denoted as  $g(r, c)$ , is included in the calculation. To avoid a confusion with their “standard” counterparts, the gray value moments and the derived features are marked with a small ‘g’ in the upper left corner.

$${}^gM_{jk} = \sum_{(r,c) \in B} r^j c^k \cdot g(r, c) \quad (\text{A.15})$$

Therefore, the *gray value area*  ${}^gA$  of the blob  $B$  is calculated as:

$${}^gA = {}^gM_{00} = \sum_{(r,c) \in B} g(r, c) \quad (\text{A.16})$$

and the *gray value center of gravity*  $({}^g\bar{r}, {}^g\bar{c})$  as:

$${}^g\bar{r} = \frac{{}^gM_{10}}{{}^gM_{00}} \quad {}^g\bar{c} = \frac{{}^gM_{01}}{{}^gM_{00}} \quad (\text{A.17})$$

The normalized (j,k)th gray value moment of a blob  $B$ , denoted as  ${}^g\mu_{jk}$  is defined as

$${}^g\mu_{jk} = \frac{1}{{}^gA} \cdot \sum_{(r,c) \in B} (r - {}^g\bar{r})^j \cdot (c - {}^g\bar{c})^k \cdot g(r, c) \quad (\text{A.18})$$

From this, the parameters of the equivalent ellipse are calculated as follows:

$${}^gR_{maj} = \sqrt{2 \cdot ({}^g\mu_{20} + {}^g\mu_{02} + \sqrt{({}^g\mu_{20} - {}^g\mu_{02})^2 + 4 \cdot ({}^g\mu_{11})^2})} \quad (\text{A.19})$$

$${}^gR_{min} = \sqrt{2 \cdot ({}^g\mu_{20} + {}^g\mu_{02} - \sqrt{({}^g\mu_{20} - {}^g\mu_{02})^2 + 4 \cdot ({}^g\mu_{11})^2})} \quad (\text{A.20})$$

$${}^g\varphi_E = \arctan\left(\frac{-2 \cdot {}^g\mu_{11}}{{}^g\mu_{20} - {}^g\mu_{02} + \sqrt{({}^g\mu_{20} - {}^g\mu_{02})^2 + 4 \cdot ({}^g\mu_{11})^2}}\right) \quad (\text{A.21})$$

## A.5 Gray Value Features

Gray value features evaluate the gray value of the pixels of a blob  $B$ , denoted as  $g(r, c)$ , with  $r$  and  $c$  being the row and column coordinates of the pixels. The *mean* gray value  $\bar{g}$  and the *deviation*  $\sigma$  from it are calculated as follows:

$$\bar{g} = \frac{1}{A} \cdot \sum_{(r,c) \in B} g(r, c) \quad (\text{A.22})$$

$$\sigma = \sqrt{\frac{1}{A} \cdot \sum_{(r,c) \in B} (g(r, c) - \bar{g})^2} \quad (\text{A.23})$$

## A.6 Texture Features

The texture features Entropy and Anisotropy are determined from the *histogram* of gray value frequencies  $h(f(g))$  as follows:

$$\text{Entropy} = - \sum_{g=0}^{255} h(f(g)) \cdot \log_2(h(f(g))) \quad (\text{A.24})$$

$$\text{Anisotropy} = \frac{\sum_{g=0}^k h(f(g)) \cdot \log_2(h(f(g)))}{\text{Entropy}} \quad (\text{A.25})$$

with  $k$  being the smallest gray value for which  $\sum_{g=0}^k h(f(g)) \geq 0.5$ .

The texture features Energy, Correlation, Homogeneity, and Contrast are determined from the *gray level co-occurrence matrix*  $Cooc(g_1, g_2)$ . This matrix stores the relative frequencies with which two gray values occur as neighbors in the blob  $B$ . The neighborhood is evaluated in the directions  $0^\circ$ ,  $45^\circ$ ,  $90^\circ$ , and  $135^\circ$ .

$$\text{Energy} = \sum_{(g_1, g_2) \in Cooc} Cooc(g_1, g_2)^2 \quad (\text{A.26})$$

$$\text{Correlation} = \frac{\sum_{(g_1, g_2) \in Cooc} (g_1 - \bar{g}) \cdot (g_2 - \bar{g}) \cdot Cooc(g_1, g_2)}{\sigma^2} \quad (\text{A.27})$$

$$\text{Homogeneity} = \sum_{(g_1, g_2) \in Cooc} \frac{Cooc(g_1, g_2)}{1 + (g_1 - g_2)^2} \quad (\text{A.28})$$

$$\text{Contrast} = \sum_{(g_1, g_2) \in Cooc} (g_1 - g_2)^2 \cdot Cooc(g_1, g_2) \quad (\text{A.29})$$