
Vision User Manual



Mess- und Prüftechnik. Die Experten.

**Ihr Ansprechpartner /
Your Partner:**

dataTec AG
E-Mail: info@datatec.eu
>>> www.datatec.eu



Authorized
Distributor

2025-12-21



Contents

Vision User Manual	8
Vision Overview	9
Digital Images	10
Definition of a Digital Image	10
Properties of a Digitized Image	10
Image Types	12
Image Files	14
Internal Representation of a Vision Image	15
Image Borders	16
Image Masks	19
Display	21
Image Display	21
Palettes	24
Regions of Interest	28
Nondestructive Overlay	29
Setting Up Your Imaging System	31
Spatial Calibration	36
When to Use	37
Concepts	40
In-Depth Discussion	51
Image Processing and Analysis	61
Image Analysis	61
Histogram	61
Line Profile	66
Intensity Measurements	66
Structural Similarity Index	67
Feature Extraction	69
Image Processing	72
Lookup Tables	72
Convolution Kernels	81
Spatial Filtering	83
Grayscale Morphology	107

Operators	114
Introduction	114
When to Use	114
Concepts	114
Frequency Domain Analysis	122
Introduction	122
When to Use	124
Concepts	124
In-Depth Discussion	133
Texture Defect Detection	135
Introduction	135
When to Use	136
What to Expect from Texture Defect Detection	136
In-Depth Discussion	138
Flat Field Correction	145
Introduction to Flat Field Correction	145
When to Use Flat Field Correction	146
Flat Field Correction Concepts	147
Flat Field Correction In-Depth Discussion	148
Particle Analysis	151
Image Segmentation	153
Thresholding	154
Morphological Segmentation	171
Binary Morphology	176
Introduction	176
Structuring Elements	176
Connectivity	182
Primary Morphology Operations	185
Advanced Morphology Operations	196
Morphological Reconstruction	205
Particle Measurements	210
Introduction	211
Particle Measurements	212
Machine Vision	229
Edge Detection	229
Introduction	229

When to Use.	229
Concepts	233
Pattern Matching	250
Introduction	250
When to Use.	251
Limitations.	252
What to Expect from a Pattern Matching Tool	253
Pattern Matching Techniques	255
In-Depth Discussion.	269
Advanced Pattern Matching Concepts.	270
Object Tracking.	282
Introduction	282
When to Use.	283
What to Expect from Object Tracking	283
Object Tracking Techniques.	283
In-Depth Discussion.	288
Geometric Matching	290
Introduction	290
When to Use.	291
What to Expect from a Geometric Matching Tool	293
Geometric Matching Technique	296
Geometric Matching Using Calibrated Images	306
In-Depth Discussion.	307
Dimensional Measurements	310
Introduction	311
When to Use.	311
Concepts	311
Coordinate System	313
Finding Features or Measurement Points.	319
Making Measurements on the Image.	321
Contour Analysis	328
Introduction	328
When to Use.	328
Contour Analysis Concepts	329
In-Depth Discussion.	335
Color Inspection.	336

Color Spaces	336
Color Spectrum	342
Color Matching	346
Color Location.....	351
Color Pattern Matching.....	356
Color Segmentation.....	370
Introduction	373
Model Importer Techniques.....	374
In-Depth Discussion.....	378
Classification.....	381
Introduction	381
Training the Classifier	382
Binary Particle Classification.....	383
Color Classification	388
Nearest Neighbor.....	393
Support Vector Machines.....	396
Custom Classification	403
In-Depth Discussion.....	403
Defect Inspection.....	410
When to Use Defect Maps.....	410
Defect Map Concepts.....	410
Golden Template Comparison	412
Introduction	412
When to Use.....	413
Concepts	413
Optical Character Recognition	418
Introduction	418
When to Use.....	419
Training Characters	419
Reading Characters	421
OCR Session	422
Concepts and Terminology	423
Instrument Readers.....	434
Introduction	434
Meter Functions	434
LCD Functions.....	435

Barcode Functions	436
2D Code Recognition	438
What to Expect from 2D Code Recognition	438
Data Matrix Concepts	439
Quality Grading	440
Decode	440
Symbol Contrast	440
Print Growth	441
Axial Nonuniformity	442
Unused Error Correction	443
Overall Symbol Grade	445
ISO 15415 Grading Standard Concepts	445
Modulation	446
Fixed Pattern Damage	447
Grid Nonuniformity	448
Scan Grade	450
Overall Symbol Grade	450
AIM DPM Grading Standard Concepts	450
Cell Contrast	451
Cell Modulation	452
Minimum Reflectance	453
PDF417 Concepts	454
QR Code Concepts	455
Micro QR Code Concepts	455
Stereo Vision	456
Stereo Vision in NI Vision	456
When to Use Stereo Vision	456
Stereo Vision in Navigation Applications	457
Stereo Vision in Robotic Applications	458
Stereo Vision in Machine Vision Applications	458
Stereo Vision in Surveillance Applications	458
What to Expect from a Stereo Vision System	459
Stereo Vision Concepts	462
Parts of a Stereo Vision System	463
Stereo Calibration	465
Maximum Projection Error and Calibration Quality Metric	466

Stereo Image Rectification	466
Maximum Rectification Error and Rectification Quality Metric	468
Stereo Image Correspondence	469
Confidence Score Image	470
Depth Computation	470
In-Depth Discussion of Stereo Vision Concepts	472
Stereo Calibration In-Depth	472
Stereo Image Rectification In-Depth	474
Stereo Image Correspondence In-Depth	475
Pre-Filtering for Stereo Image Correspondence	475
Block Matching Algorithm	476
Semi-Global Matching Algorithm	477
Post-Filtering for Stereo Image Correspondence	479
Depth Computation In-Depth	480
Error Mapping for Depth Computation	481
Feature Detection and Matching	481
Introduction to Feature Detection and Matching	482
When to Use Feature Detection and Matching	482
Feature Detection and Matching Concepts	483
Kernels	486
Gradient Kernels	486
Laplacian Kernels	491
Smoothing Kernels	493
Gaussian Kernels	494

Vision User Manual

The Vision User Manual provides detailed descriptions of the product functionality and the step by step processes for use.

Looking for Something Else?

For information not found in the User Manual for your product, such as specifications and API reference, browse ***Related Information***.

Related information:

- [Hardware and Software Operating System Compatibility](#)
- [License Setup and Activation](#)

Vision Overview

The ***Vision Concepts Help*** describes the basic concepts of machine vision and image processing for users with little or no imaging experience. This document also contains in-depth discussions on machine vision and image processing functions for advanced users.

Digital Images

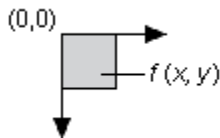
This section describes conceptual information about digital images, image display, and system calibration.

Also contains information about the properties of digital images, image types, file formats, the internal representation of images in NI Vision, image borders, and image masks.

Definition of a Digital Image

An image is a 2D array of values representing light intensity. For the purposes of image processing, the term image refers to a digital image. An image is a function of the light intensity $f(x, y)$ where f is the brightness of the point (x, y) , and x and y represent the spatial coordinates of a picture element, or pixel.

By convention, the spatial reference of the pixel with the coordinates $(0, 0)$ is located at the top, left corner of the image. Notice in the following figure that the value of x increases moving from left to right, and the value of y increases from top to bottom.



In digital image processing, an imaging sensor converts an image into a discrete number of pixels. The imaging sensor assigns to each pixel a numeric location and a gray level or color value that specifies the brightness or color of the pixel.

Properties of a Digitized Image

A digitized image has three basic properties: resolution, definition, and number of planes.

Image Resolution

The spatial resolution of an image is determined by its number of rows and columns of pixels. An image composed of m columns and n rows has a resolution of $m \times n$. This image has m pixels along its horizontal axis and n pixels along its vertical axis.

Image Definition

The definition of an image indicates the number of shades that you can see in the image. The bit depth of an image is the number of bits used to encode the value of a pixel. For a given bit depth of n , the image has an image definition of 2^n , meaning a pixel can have 2^n different values. For example, if n equals 8 bits, a pixel can have 256 different values ranging from 0 to 255. If n equals 16 bits, a pixel can have 65,536 different values ranging from 0 to 65,535 or from -32,768 to 32,767.

Vision can process images with 8-bit, 10-bit, 12-bit, 14-bit, 16-bit, floating point, or color encoding. The manner in which you encode your image depends on the nature of the image acquisition device, the type of image processing you need to use, and the type of analysis you need to perform. For example, 8-bit encoding is sufficient if you need to obtain the shape information of objects in an image. However, if you need to precisely measure the light intensity of an image or region in an image, you should use 16-bit or floating-point encoding.

Use color encoded images when your machine vision or image processing application depends on the color content of the objects you are inspecting or analyzing.

Vision does not directly support other types of image encoding, particularly images encoded as 1-bit, 2-bit, or 4-bit images. In these cases, Vision automatically transforms the image into an 8-bit image—the minimum bit depth for NI Vision—when opening the image file.

The number of planes in an image corresponds to the number of arrays of pixels that compose the image. A grayscale or pseudo-color image is composed of one plane. A true-color image is composed of three planes—one each for the red component, blue component, and green component.

Number of Planes

In true-color images, the color component intensities of a pixel are coded into three different values. A color image is the combination of three arrays of pixels corresponding to the red, green, and blue components in an RGB image. HSL images are defined by their hue, saturation, and luminance values.

Image Types

The Vision libraries can manipulate three types of images: grayscale, color, and complex images. Although Vision supports all three image types, certain operations on specific image types are not possible. For example, you cannot apply the logic operator AND to a complex image.

The following table shows how many bytes per pixel grayscale, color, and complex images use. For an identical spatial resolution, a color image occupies four times the memory space of an 8-bit grayscale image, and a complex image occupies eight times the memory of an 8-bit grayscale image.

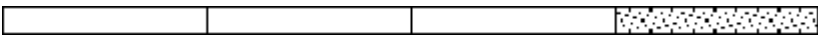
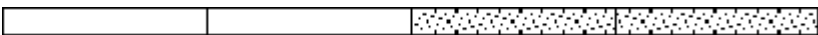
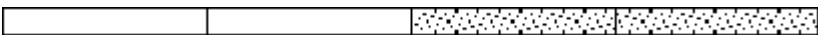




Image Type	Number of Bytes per Pixel Data
8-bit (Unsigned) Integer Grayscale (1 byte or 8-bit)	 8-bit for the grayscale intensity
16-bit (Unsigned) Integer Grayscale (2 bytes or 16-bit)	 16-bit for the grayscale intensity
16-bit (Signed) Integer Grayscale (2 bytes or 16-bit)	 16-bit for the grayscale intensity
32-bit Floating-Point Grayscale	

Image Type	Number of Bytes per Pixel Data
(4 bytes or 32-bit)	32-bit for the grayscale intensity
32-bit RGB Color (4 bytes or 32-bit)	 8-bit for the alpha value (not used), 8-bit for the red intensity, 8-bit for the green intensity, 8-bit for the blue intensity
64-bit (Unsigned) RGB Color (8 bytes or 64-bit)	 16-bit for the alpha value (not used), 16-bit for the green intensity, 16-bit for the red intensity, 16-bit for the blue intensity
64-bit Complex Color (8 bytes or 64-bit)	 32-bit floating for the real part, 32-bit for the imaginary part

Grayscale Images

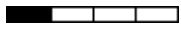


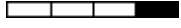
A grayscale image is composed of a single plane of pixels. Each pixel is encoded using one of the following single numbers:

- An 8-bit unsigned integer representing grayscale values between 0 and 255
- A 16-bit unsigned integer representing grayscale values between 0 and 65,535
- A 16-bit signed integer representing grayscale values between -32,768 and 32,767
- A single-precision floating point number, encoded using four bytes, representing grayscale values ranging from $-\infty$ to ∞

Color Images

A color image is encoded in memory as either a red, green, and blue (RGB) image or a hue, saturation, and luminance (HSL) image. Color image pixels are a composite of four values. RGB images store color information using 8 bits each for the red, green, and blue planes. HSL images store color information using 8 bits each for hue, saturation, and luminance. RGB U64 images store color information using 16 bits each for the red, green, and blue planes. In the RGB and HSL color models, an additional

8-bit value goes unused. This representation is known as 4 × 8-bit or 32-bit encoding. In the RGB U64 color model, an additional 16-bit value goes unused. This representation is known as 4 × 16-bit or 64-bit encoding.

Alpha plane (not used)	
Red or hue plane	
Green or saturation plane	
Blue or luminance plane	

Complex Images

A complex image contains the frequency information of a grayscale image. You can create a complex image by applying a Fast Fourier transform (FFT) to a grayscale image. After you transform a grayscale image into a complex image, you can perform frequency domain operations on the image.

Each pixel in a complex image is encoded as two single-precision floating-point values, which represent the real and imaginary components of the complex pixel. You can extract the following four components from a complex image: the real part, imaginary part, magnitude, and phase.

Image Files

An image file is composed of a header followed by pixel values. Depending on the file format, the header contains image information about the horizontal and vertical resolution, pixel definition, and the original palette. Image files may also store information about calibration, pattern matching templates, and overlays. The following are common image file formats:

- Bitmap (BMP)
- Tagged image file format (TIFF)
- Portable network graphics (PNG)—Offers the capability of storing image information about spatial calibration, pattern matching templates, custom data, and overlays
- Joint Photographic Experts Group format (JPEG)

- Joint Photographic Experts Group 2000 format (JPEG2000)
- Audio Video Interleave (AVI)—Offers the capability of storing multiple image frames in a single file
- National Instruments internal image file format (AIPD)—Used for saving floating-point, complex, and HSL images

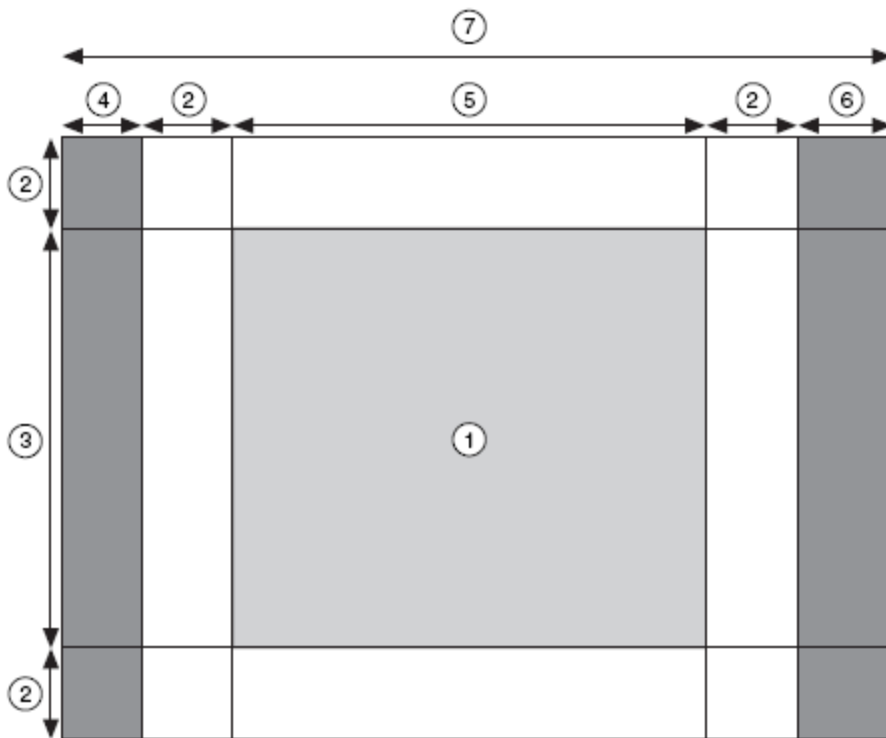
The following table lists the image file formats supported for each image type:

	BMP	TIFF	PNG	JPEG	JPEG2000	AVI	AIPD
8-bit Unsigned Grayscale	✓	✓	✓	✓	✓	✓	✓
16-bit Unsigned Grayscale		✓	✓		✓		✓
16-bit Signed Grayscale		✓	✓		✓		✓
32-bit Floating-Point Grayscale							✓
32-bit RGB Color	✓	✓	✓	✓	✓	✓	✓
64-bit RGB Color		✓	✓		✓		✓
32-bit HSL Color							✓
Complex							✓

Internal Representation of a Vision Image

The following figure illustrates how a Vision image is represented in system memory. In addition to the image pixels, the stored image includes additional rows and columns of pixels called the image border and the left and right alignments. Specific processing functions involving pixel neighborhood operations use image borders. The alignment regions ensure that the first pixel of the image is 64-byte aligned in memory. The size of the alignment blocks depend on the image width and border size. Aligning the image increases processing speed by as much as 30%.

The line width is the total number of pixels in a horizontal line of an image, which includes the sum of the horizontal resolution, the image borders, and the left and right alignments. The horizontal resolution and line width may be the same length if the horizontal resolution is a multiple of 32 bytes and the border size is 0.



1. Image
2. Image Border
3. Vertical Resolution
4. Left Alignment
5. Horizontal Resolution
6. Right Alignment
7. Line Width

Image Borders

Many image processing functions process a pixel by using the values of its neighbors. A neighbor is a pixel whose value affects the value of a nearby pixel when an image is processed. Pixels along the edge of an image do not have neighbors on all four sides. If you need to use a function that processes pixels based on the value of their neighboring pixels, specify an image border that surrounds the image to account for these outlying pixels. You define the image border by specifying a border size and the values of the border pixels.

The size of the border should accommodate the largest pixel neighborhood required by the function you are using. The size of the neighborhood is specified by the size of a

2D array. For example, if a function uses the eight adjoining neighbors of a pixel for processing, the size of the neighborhood is 3×3 , indicating an array with three columns and three rows. Set the border size to be greater than or equal to half the number of rows or columns of the 2D array rounded down to the nearest integer value. For example, if a function uses a 3×3 neighborhood, the image should have a border size of at least 1; if a function uses a 5×5 neighborhood, the image should have a border size of at least 2. In Vision, an image is created with a default border size of 3, which can support any function using up to a 7×7 neighborhood without any modification.

Vision provides three ways to specify the pixel values of the image border. The following figure illustrates these options. Figure A shows the pixel values of an image. By default, all image border pixels are uninitialized. You can set all of the border pixels to have a value of 0, as shown in figure B. You can copy the values of the pixels along the edge of the image into the border pixels, as shown in figure C, or you can mirror the pixel values along the edge of the image into the border pixels, as shown in figure D.

[illegible][illegible]

10	10	10	9	15	11	12	20	16	22	11	8	8	8
10	10	10	9	15	11	12	20	16	22	11	8	8	8
10	10	10	9	15	11	12	20	16	12	11	8	8	8
11	11	11	13	11	12	9	16	17	11	13	14	14	14
12	12	12	8	12	14	12	13	12	14	11	13	13	13
10	10	10	9	13	31	30	32	33	12	13	11	11	11
15	15	15	11	10	30	42	45	31	15	12	10	10	10
13	13	13	12	14	29	40	41	33	13	12	13	13	13
14	14	14	15	12	33	34	36	32	14	11	11	11	11
10	10	10	12	13	14	12	16	12	15	10	9	9	9
10	10	10	8	11	13	15	17	13	14	12	10	10	10
9	9	9	10	12	11	8	15	14	12	11	7	7	7
9	9	9	10	12	11	8	15	14	12	11	7	7	7
9	9	9	10	12	11	8	15	14	12	11	7	7	7

13	11	11	13	11	12	9	16	17	11	13	14	14	13
9	10	10	9	15	11	12	20	16	12	11	8	8	11
9	10	10	9	15	11	12	20	16	12	11	8	8	
13	11	11	13	11	12	9	16	17	11	13	14	14	13
8	12	12	8	12	14	12	13	12	14	11	13	13	11
9	10	10	9	13	31	30	32	33	12	13	11	11	13
11	15	15	11	10	30	42	45	31	15	12	10	10	12
12	13	13	12	14	29	40	41	33	13	12	13	13	12
15	14	14	15	12	33	34	36	32	12	14	11	11	14
12	10	10	12	13	14	12	16	12	15	10	9	9	10
8	10	10	8	11	13	15	17	13	14	12	10	10	12
10	9	9	10	12	11	8	15	14	12	11	7	7	11
10	9	9	10	12	11	8	15	14	12	11	7	7	11
8	10	10	8	11	13	15	17	13	14	12	10	10	12

The method you use to fill the border pixels depends on the processing function you require for your application. Review how the function works before choosing a border-filling method because your choice can drastically affect the processing results. For example, if you are using a function that detects edges in an image based on the difference between a pixel and its neighbors, do not set the border pixel values to zero. As shown in figure B, an image border containing zero values introduces significant differences between the pixel values in the border and the image pixels along the border, which causes the function to detect erroneous edges along the border of the image. If you are using an edge detection function, copy or mirror the pixel values along the border into the border region to obtain more accurate results.

In Vision, most image processing functions that use neighbors automatically set pixel values in the image border using neighborhoods. The grayscale filtering operations low pass, Nth order, and edge detection use the mirroring method to set pixels in the image border. The binary morphology, grayscale morphology, and segmentation

functions copy the pixel values along the border into the border region. The correlate, circles, reject border, remove particles, skeleton, and label functions set the pixel values in the border to zero.



Note The border of an image is taken into account only for processing. The border is never displayed or stored in a file.

Image Masks

An image mask isolates parts of an image for processing. If a function has an image mask parameter, the function process or analysis depends on both the source image and the image mask.

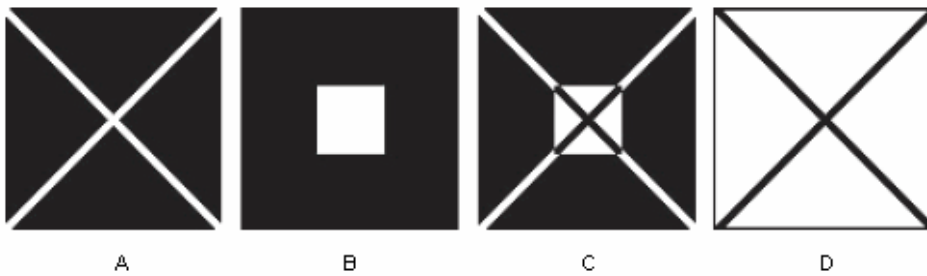
An image mask is an 8-bit binary image that is the same size as or smaller than the inspection image. Pixels in the image mask determine whether corresponding pixels in the inspection image are processed. If a pixel in the image mask has a nonzero value, the corresponding pixel in the inspection image is processed. If a pixel in the image mask has a value of 0, the corresponding pixel in the inspection image is not processed.

When to Use

Use image masks when you want to focus your processing or inspection on particular regions in the image.

Concepts

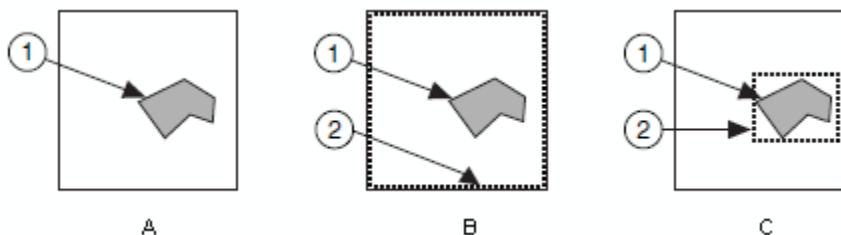
Pixels in the source image are processed if corresponding pixels in the image mask have values other than zero. The following figure shows how a mask affects the output of the function that inverts the pixel values in an image. Figure A shows the inspection image. Figure B shows the image mask. Pixels in the mask with zero values are represented in black, and pixels with nonzero values are represented in white. Figure C shows the inverse of the inspection image using the image mask. Figure D shows the inverse of the inspection image without the image mask.



The Effect of an Image Mask

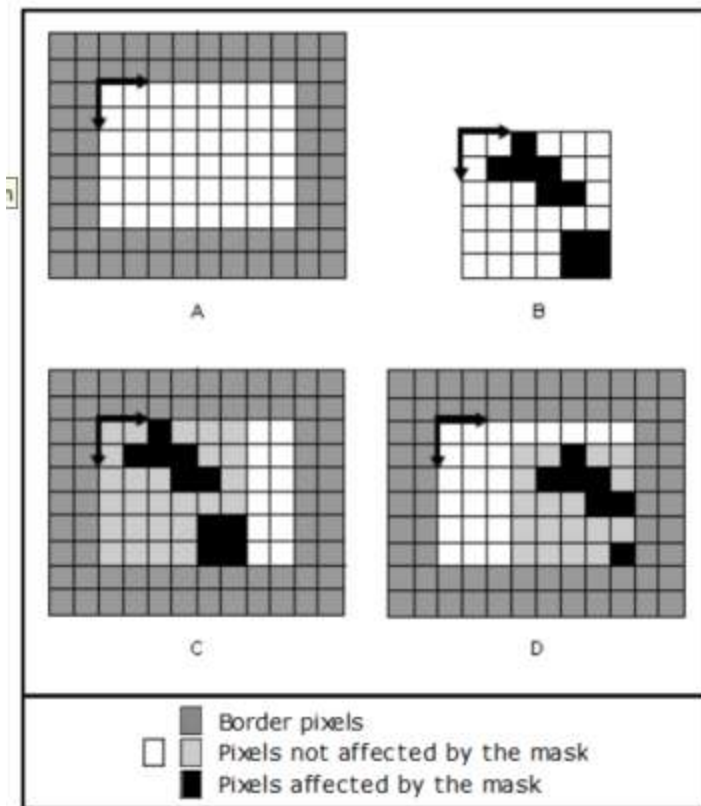
You can limit the area in which your function applies an image mask to the bounding rectangle of the region you want to process. This technique saves memory by limiting the image mask to only the part of the image containing significant information. To keep track of the location of this region of interest (ROI) in regard to the original image, Vision sets an offset. An offset defines the coordinate position in the original image where you want to place the origin of the image mask.

The following figure illustrates the different methods of applying image masks. Figure A shows the ROI in which you want to apply an image mask. Figure B shows an image mask with the same size as the inspection image. In this case, the offset is set to $[0, 0]$. A mask image also can be the size of the bounding rectangle of the ROI, as shown in figure C, where the offset specifies the location of the mask image in the reference image. You can define this offset to apply the mask image to different regions in the inspection image.



1. Region of Interest
2. Image Mask

the following figure illustrates the use of a mask with two different offsets. Figure A shows the inspection image, and figure B shows the image mask. Figure C and Figure D show the results of a function using the image mask given the offsets of $[0, 0]$ and $[3, 1]$, respectively.



Related concepts:

- [Regions of Interest](#)

Display

This section contains information about image display, palettes, regions of interest, and nondestructive overlays.

Image Display

Displaying images is an important component of a vision application because it gives you the ability to visualize your data. Image processing and image visualization are distinct and separate elements. Image processing refers to the creation, acquisition, and analysis of images. Image visualization refers to how image data is presented and how you can interact with the visualized images. A typical imaging application uses many images in memory that the application never displays.

When to Use

Use display functions to visualize your image data, retrieve generated events and the associated data from an image display environment, select ROIs from an image interactively, and annotate the image with additional information.

Concepts

Display functions display images, set attributes of the image display environment, assign color palettes to image display environments, close image display environments, and set up and use an image browser in image display environments. Some ROI functions—a subset of the display functions—interactively define ROIs in image display environments. These ROI functions configure and display different drawing tools, detect draw events, retrieve information about the region drawn on the image display environment, and move and rotate ROIs. Nondestructive overlays display important information on top of an image without changing the values of the image pixels.

In-Depth Discussion

The following section describes the display modes available in Vision and the 16-bit grayscale display mapping methods.

Display Modes

One of the key components of displaying images is the display mode that the video adapter operates. The display mode indicates how many bits specify the color of a pixel on the display screen. Generally, the display mode available from a video adapter ranges from 8 bits to 32 bits per pixel, depending the amount of video memory available on the video adapter and the screen resolution you choose.

If you have an 8-bit display mode, a pixel can be one of 256 different colors. If you have a 16-bit display mode, a pixel can be one of 65,536 colors. In 24-bit or 32-bit display mode, the color of a pixel on the screen is encoded using 3 or 4 bytes, respectively. In these modes, information is stored using 8 bits each for the red, green, and blue components of the pixel. These modes offer the possibility to display about 16.7 million colors.

Understanding your display mode is important to understanding how Vision displays the different image types on a screen. Image processing functions often use grayscale images. Because display screen pixels are made of red, green, and blue components, the pixels of a grayscale image cannot be rendered directly.

In 24-bit or 32-bit display mode, the display adapter uses 8 bits to encode a grayscale value, offering 256 gray shades. This color resolution is sufficient to display 8-bit grayscale images. However, higher bit depth images, such as 16-bit grayscale images, are not accurately represented in 24-bit or 32-bit display mode. To display a 16-bit grayscale image, either ignore the least significant bits or use a mapping function to convert 16 bits to 8 bits.

Mapping Methods for 16-Bit Image Display

The following techniques describe how Vision converts 16-bit images to 8-bit images and displays them using mapping functions. Mapping functions evenly distribute the dynamic range of the 16-bit image to an 8-bit image.

- **Full Dynamic**—The minimum intensity value of the 16-bit image is mapped to 0, and the maximum intensity value is mapped to 255. All other values in the image are mapped between 0 and 255 using the equation shown below. This mapping method is general purpose because it ensures the display of the complete dynamic range of the image. Because the minimum and maximum pixel values in an image are used to determine the full dynamic range of that image, the presence of noisy or defective pixels (for non-Class A sensors) with minimum or maximum values can affect the appearance of the displayed image. Vision uses the following technique by default:

$$z = \frac{x - y}{v - y} \times 255$$

where:

- z is the 8-bit pixel value,
 - x is the 16-bit value,
 - y is the minimum intensity value,
 - v is the maximum intensity value.
- **90% Dynamic**—The intensity corresponding to 5% of the cumulative histogram is mapped to 0, the intensity corresponding to 95% of the cumulated histogram is

mapped to 255. Values in the 0 to 5% range are mapped to 0, while values in the 95 to 100% range are mapped to 255. This mapping method is more robust than the full dynamic method and is not sensitive to small aberrations in the image. This method requires the computation of the cumulative histogram or an estimate of the histogram. Refer to ***image analysis***, for more information on histograms.

- **Given Percent Range**—This method is similar to the 90% Dynamic method, except that the minimum and maximum percentages of the cumulative histogram that the software maps to 8-bit are user defined.
- **Given Range**—This technique is similar to the Full Dynamic method, except that the minimum and maximum values to be mapped to 0 and 255 are user defined. You can use this method to enhance the contrast of some regions of the image by finding the minimum and maximum values of those regions and computing the histogram of those regions. A histogram of this region shows the minimum and maximum intensities of the pixels. Those values are used to stretch the dynamic range of the entire image.
- **Downshifts**—This technique is based on shifts of the pixel values. This method applies a given number of right shifts to the 16-bit pixel value and displays the least significant bit. This technique truncates some of the lowest bits, which are not displayed. This method is very fast, but it reduces the real dynamic of the sensor to 8-bit sensor capabilities. It requires knowledge of the bit-depth of the imaging sensor that has been used. For example, an image acquired with a 12-bit camera should be visualized using four right shifts in order to display the eight most significant bits acquired with the camera. If you are using a National Instruments image acquisition device, this technique is the default used by Measurement & Automation Explorer (MAX).

Related concepts:

- [Image Analysis](#)

Palettes

At the time a grayscale image is displayed on the screen, Vision converts the value of each pixel of the image into red, green, and blue intensities for the corresponding pixel displayed on the screen. This process uses a color table, called a palette, which associates a color to each possible grayscale value of an image. Vision provides the capability to customize the palette used to display an 8-bit grayscale image.

When to Use

With palettes, you can produce different visual representations of an image without altering the pixel data. Palettes can generate effects, such as photonegative displays or color-coded displays. In the latter case, palettes are useful for detailing particular image constituents in which the total number of colors is limited.

Displaying images in different palettes helps emphasize regions with particular intensities, identify smooth or abrupt gray-level variations, and convey details that might be difficult to perceive in a grayscale image. For example, the human eye is much more sensitive to small intensity variations in a bright area than in a dark area. Using a color palette may help you distinguish these slight changes.

Concepts

A palette is a pre-defined or user-defined array of RGB values. It defines for each possible gray-level value a corresponding color value to render the pixel. The gray-level value of a pixel acts as an address that is indexed into the table, returning three values corresponding to a red, green, and blue (RGB) intensity. This set of RGB values defines a palette in which varying amounts of red, green, and blue are mixed to produce a color representation of the value range.

In the case of 8-bit grayscale images, pixels can take 28, or 256, values ranging from 0 to 255. Color palettes are composed of 256 RGB elements. A specific color is the result of applying a value between 0 and 255 for each of the three color components: red, green, and blue. If the red, green, and blue components have an identical value, the result is a gray level pixel value.

A gray palette associates different shades of gray with each value so as to produce a linear and continuous gradation of gray, from black to white. You can set up the palette to assign the color black to the value 0 and white to 255, or vice versa. Other palettes can reflect linear or nonlinear gradations going from red to blue, light brown to dark brown, and so on.

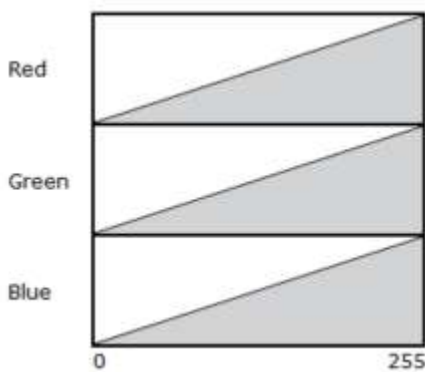
Vision has five predefined color palettes. Each palette emphasizes different shades of gray.

In-Depth Discussion

The following sections introduce the five predefined palettes available in NI Vision. The graphs in each section represent the color tables used by each palette. The horizontal axes of the graphs represent the input gray-level range $[0, 255]$, and the vertical axes represent the RGB intensities assigned to a given gray-level value.

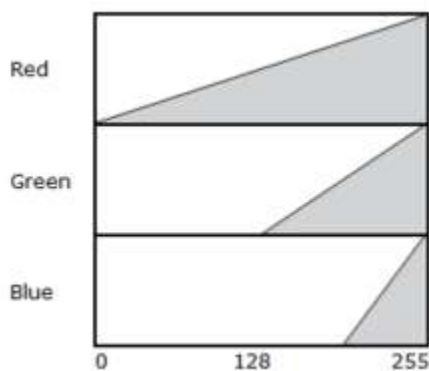
Gray Palette

This palette has a gradual gray-level variation from black to white. Each value is assigned to an equal amount of red, green, and blue in order to produce a gray-level.



Temperature Palette

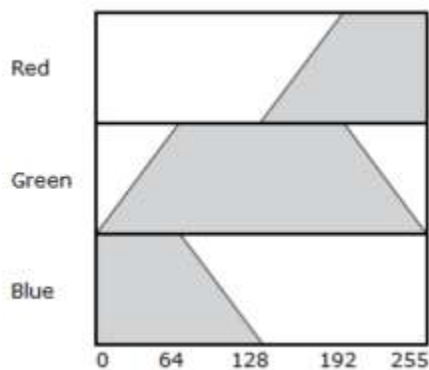
This palette has a gradation from light brown to dark brown. 0 is black and 255 is white.



Rainbow Palette

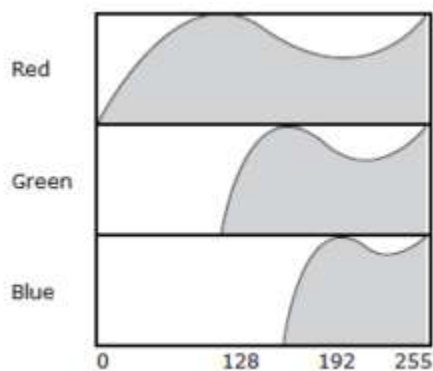
This palette has a gradation from blue to red with a prominent range of greens in the

middle value range. 0 is blue and 255 is red.



Gradient Palette

This palette has a gradation from red to white with a prominent range of light blue in the upper value range. 0 is black and 255 is white.



Binary Palette

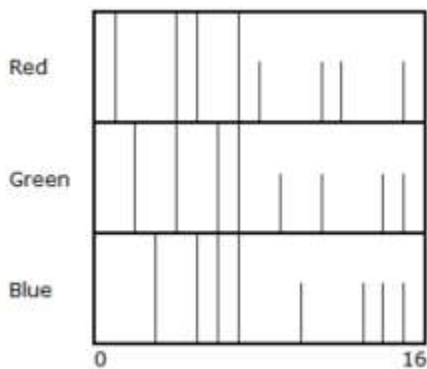
This palette has 17 cycles of 15 different colors. The following table illustrates these colors, where g is the gray-level value.

$g =$	R	G	B	Resulting Color
1	255	0	0	Red
2	0	255	0	Green
3	0	0	255	Blue
4	255	255	0	Yellow
5	255	0	255	Purple

g =	R	G	B	Resulting Color
6	0	255	255	Aqua
7	255	127	0	Orange
8	255	0	127	Magenta
9	127	255	0	Bright green
10	127	0	255	Violet
11	0	127	255	Sky blue
12	0	255	127	Sea green
13	255	127	127	Rose
14	127	255	127	Spring green
15	127	127	255	Periwinkle

The values 0 and 255 are special cases. A value of 0 results in black, and a value of 255 results in white.

This periodic palette is appropriate for the display of binary and labeled images.



Regions of Interest

A region of interest (ROI) is an area of an image in which you want to perform your image analysis. When to Use

Use ROIs to focus your processing and analysis on part of an image. You can define an ROI using standard contours, such as an oval or rectangle, or freehand contours. You also can perform any of the following options:

- Construct an ROI in an image display environment.
- Associate an ROI with an image display environment.
- Extract an ROI associated with an image display environment.
- Erase the current ROI from an image display environment.
- Transform an ROI into an image mask.
- Transform an image mask into an ROI.

Concepts

An ROI describes a region or multiple regions of an image in which you want to focus your processing and analysis. These regions are defined by specific contours. Vision supports the following contour types.

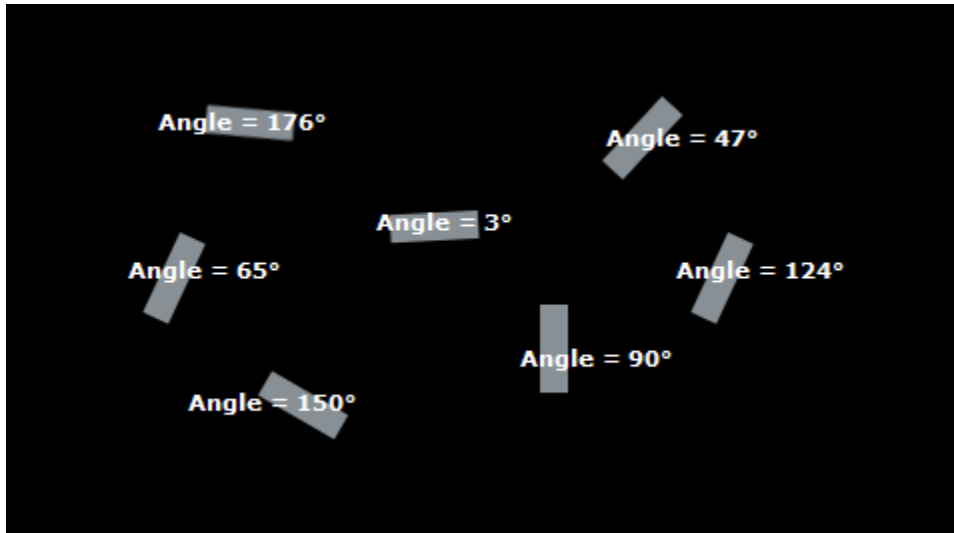
Icon	Contour Name
+	Point
/	Line
□	Rectangle
○	Oval
⬢	Polygon
⬢	Freehand Region
⊗	Annulus
⋈	Broken Line
↪	Freehand Line
◊	Rotated Rectangle

You can define an ROI interactively, programmatically, or with an image mask. Define an ROI interactively by using the tools from the tools palette to draw an ROI on a displayed image. For more information about defining ROIs programmatically or with an image mask, refer to your Vision user manual.

Nondestructive Overlay

A nondestructive overlay enables you to annotate the display of an image with useful

information without actually modifying the image. You can overlay text, lines, points, complex geometric shapes, and bitmaps on top of your image without changing the underlying pixel values in your image; only the display of the image is affected. You can also group several different overlays together to indicate a similarity between the overlays. Overlay groups act as a single overlay and allow you to apply common overlay functions to the entire group, such as clear, copy, and merge. The following figure shows how you can use the overlay to depict the orientation of each particle in the image.



When to Use

You can use nondestructive overlays for many purposes, such as the following:

- Highlighting the location in an image where objects have been detected.
- Adding quantitative or qualitative information to the displayed image, such as the match score from a pattern matching function.
- Displaying ruler grids or alignment marks.

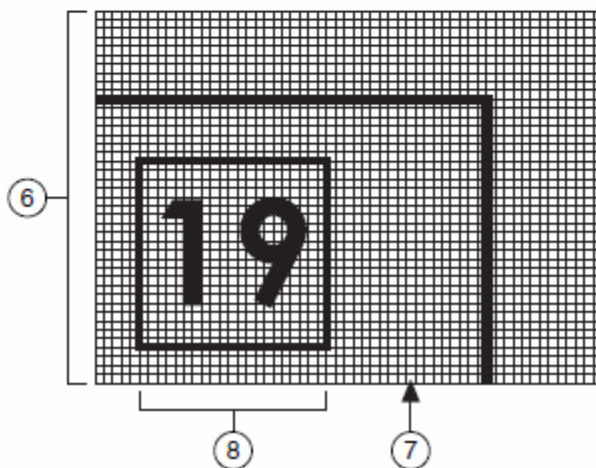
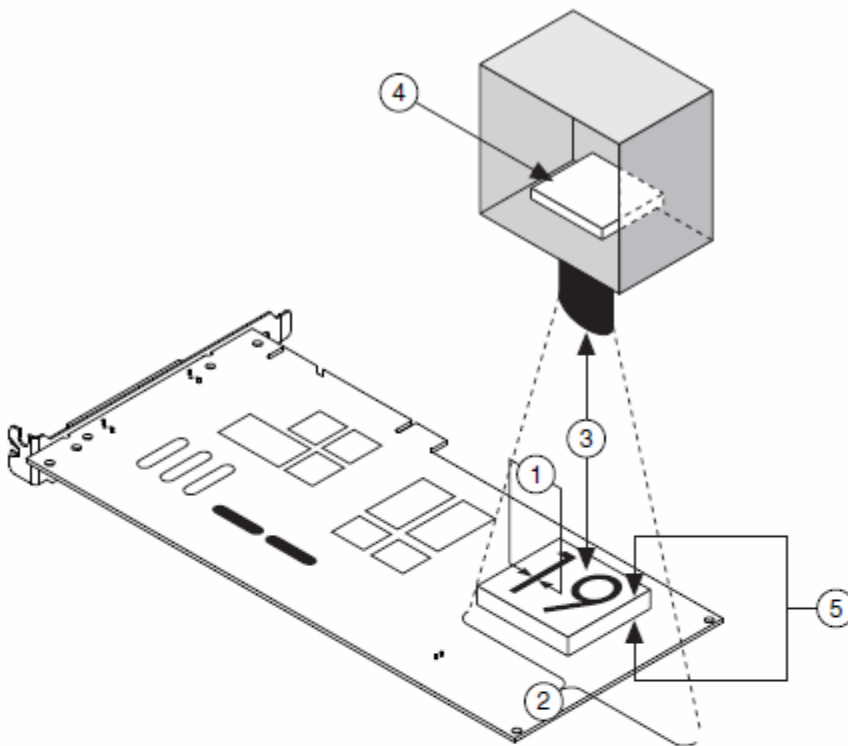
Concepts

Overlays do not affect the results of any analysis or processing functions—they affect only the display. The overlay is associated with an image, so there are no special overlay data types. You need only to add the overlay to your image. By default, Vision clears the overlay anytime you change the size or orientation of the image because the overlay ceases to have meaning. However, you can set the properties for an overlay group so that transformations applied to the image are also applied to the overlay

group. You can save overlays with images using the PNG file format.

Setting Up Your Imaging System

Before you acquire, analyze, and process images, you must set up your imaging system. Five factors comprise a imaging system: field of view, working distance, resolution, depth of field, and sensor size. The following figure illustrates these concepts.



1. Resolution—The smallest feature size on your object that the imaging system can distinguish
2. Field of view—The area of the object under inspection that the camera can acquire
3. Working distance—The distance from the front of the camera lens to the object under inspection
4. Sensor size—The size of a sensor's active area, typically defined by the sensor's horizontal dimension
5. Depth of field—The maximum object depth that remains in focus
6. Image—The image under inspection.
7. Pixel—The smallest division that makes up a digital image.
8. Pixel resolution—The minimum number of pixels needed to represent the object under inspection

For additional information about the fundamental parameters of an imaging system, refer to the **Application Notes** sections of the Edmund Industrial Optics **Optics and Optical Instruments Catalog**, or visit Edmund Industrial Optics at www.edmundoptics.com.

Acquiring Quality Images

The manner in which you set up your system depends on the type of analysis and processing you need to do. Your imaging system should produce images with high enough quality so that you can extract the information you need from the images. Five factors contribute to overall image quality: resolution, contrast, depth of field, perspective, and distortion.

Resolution

There are two kinds of resolution to consider when setting up your imaging system: pixel resolution and resolution. Pixel resolution refers to the minimum number of pixels you need to represent the object under inspection. You can determine the pixel resolution you need by the smallest feature you need to inspect. Try to have at least two pixels represent the smallest feature. You can use the following equation to determine the minimum pixel resolution required by your imaging system:

$$(\text{length of object's longest axis} / \text{size of object's smallest feature}) \times 2$$

If the object does not occupy the entire field of view, the image size will be greater than the pixel resolution.

Resolution indicates the amount of object detail that the imaging system can reproduce. Images with low resolution lack detail and often appear blurry. Three factors contribute to the resolution of your imaging system: field of view, the camera sensor size, and number of pixels in the sensor. When you know these three factors, you can determine the focal length of your camera lens.

Field of View

The field of view is the area of the object under inspection that the camera can acquire. The following figure describes the relationship between pixel resolution and the field of view.

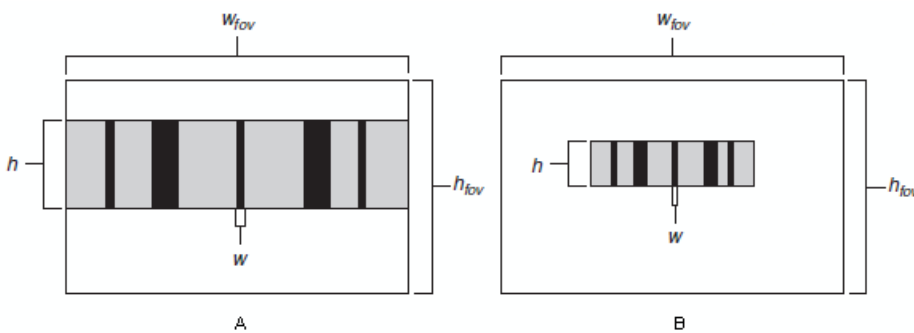


Figure A shows an object that occupies the field of view. Figure B shows an object that occupies less space than the field of view. If w is the size of the smallest feature in the x direction and h is the size of the smallest feature in the y direction, the minimum x pixel resolution is:

$$\frac{w_{fov}}{w} \times 2$$

The minimum y pixel resolution is:

$$\frac{h_{fov}}{h} \times 2$$

Choose the larger pixel resolution of the two for your imaging application.

Sensor Size and Number of Pixels in the Sensor

The camera sensor size is important in determining your field of view, which is a key element in determining your minimum resolution requirement. The sensor's diagonal length specifies the size of the sensor's active area. The number of pixels in your sensor should be greater than or equal to the pixel resolution. Choose a camera with a sensor that satisfies your minimum resolution requirement.

Lens Focal Length

When you determine the field of view and appropriate sensor size, you can decide which type of camera lens meets your imaging needs. A lens is defined primarily by its focal length. The relationship between the lens, field of view, and sensor size is as follows:

$$\text{focal length} = (\text{sensor size} \times \text{working distance}) / \text{field of view}.$$

If you cannot change the working distance, you are limited in choosing a focal length for your lens. If you have a fixed working distance and your focal length is short, your images may appear distorted. However, if you have the flexibility to change your working distance, modify the distance so that you can select a lens with the appropriate focal length and minimize distortion.

Contrast

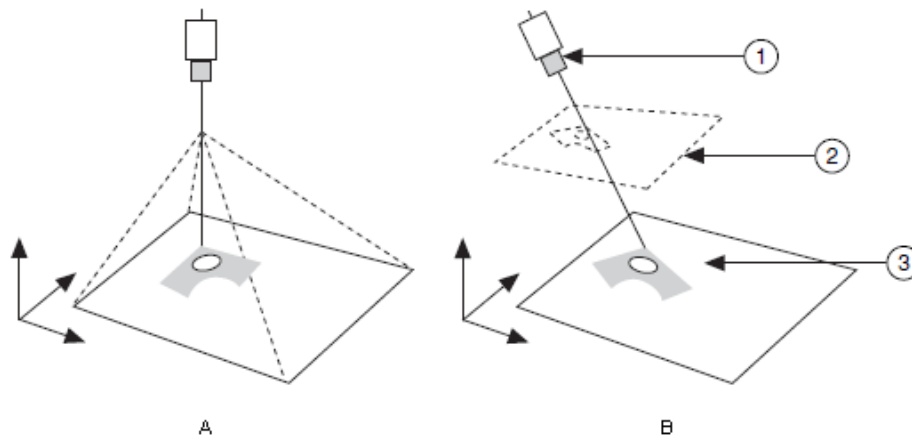
Resolution and contrast are closely related factors contributing to image quality. Contrast defines the differences in intensity values between the object under inspection and the background. Your imaging system should have enough contrast to distinguish objects from the background. Proper lighting techniques can enhance the contrast of your system.

Depth of Field

The depth of field of a lens is its ability to keep objects of varying heights in focus. If you need to inspect objects with various heights, choose a lens that can maintain the image quality you need as the objects move closer to and further from the lens.

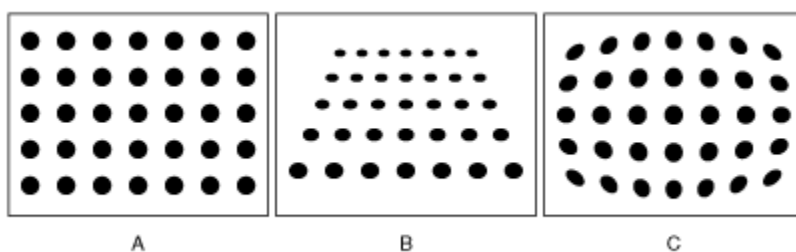
Perspective

Perspective errors often occur when the camera axis is not perpendicular to the object you are inspecting. Figure A shows an ideal camera position. Figure B shows a camera imaging an object from an angle.



1. Lens Distortion
2. Perspective Error
3. Known Orientation Offset

Perspective errors appear as changes in the object's magnification depending on the object's distance from the lens. Figure A shows a grid of dots. Figure B illustrates perspective errors caused by a camera imaging the grid from an angle.



Try to position your camera perpendicular to the object you are trying to inspect to reduce perspective errors. If you need to take precise measurements from your image, correct perspective error by applying calibration techniques to your image.

Distortion

Nonlinear distortion is a geometric aberration caused by optical errors in the camera lens. A typical camera lens introduces radial distortion. This causes points to appear further away from the optical center of the lens than they really are. Figure C illustrates the effect of distortion on a grid of dots. When distortion occurs, information in the image is misplaced relative to the center of the field of view, but the information is not necessarily lost. Therefore, you can undistort your image through spatial calibration.

Spatial Calibration

This section describes how to calibrate an imaging setup so that you can convert pixel coordinates to real-world coordinates. Converting pixel coordinates to real-world coordinates is useful when you need to make accurate measurements from inspection images using real-world units.

Introduction

Spatial calibration is the process of computing pixel to real-world unit transformations while accounting for many errors inherent to the imaging setup. Calibrating your imaging setup is important when you need to make accurate measurements in real-world units.

An image contains information in the form of pixels. Spatial calibration allows you to translate a measurement from pixel units into another unit, such as inches or centimeters. This conversion is easy if you know a conversion ratio between pixels and real-world units. For example, if one pixel equals one inch, a length measurement of 10 pixels equals 10 inches.

This conversion may not be straightforward because perspective projection and lens distortion affect the measurement in pixels. Calibration accounts for possible errors by constructing mappings that you can use to convert between pixel and real world units. You also can use the calibration information to correct perspective or nonlinear distortion errors for image display and shape measurements.

Vision calibration software supports area scan cameras using rectilinear or telecentric lenses. Vision calibration software may not accurately calibrate true fisheye or

curvilinear lenses.

When to Use

Use the Vision calibration tools to do the following:

- Calibrate your imaging setup automatically by imaging a standard pattern, such as a calibration template, or by providing reference points.
- Convert measurements such as lengths, areas, or widths between real-world units and pixel units.
- Apply a learned calibration mapping to correct an image acquired through a calibrated setup.
- Assign an arbitrary calibration axis to measure positions in real-world units relative to a point in an image.

Calibration Algorithms This section describes the calibration algorithms that are supported by Vision. The following table provides a brief description of when to use each algorithm. Refer to individual subsections for detailed information.

Name	When to Use
Simple Calibration	Use simple calibration when your camera is perpendicular to the plane of the object under inspection and distortion is negligible. For example, simple calibration can be used with an imaging setup that uses a telecentric lens.
Perspective Calibration	Use perspective calibration to correct perspective distortion introduced by a camera that is not perpendicular to plane of the object under inspection.
Distortion Modeling	<p>Use a distortion model to correct distortion introduced by lens imperfections. Vision supports the following distortion models:</p> <ul style="list-style-type: none"> • Division—Corrects radial distortion. • Polynomial—Corrects radial and tangential distortion. <p>If your camera is also not perpendicular to the object under inspection, you can combine distortion modelling with perspective calibration.</p>

Camera Modeling	<p>Use a camera model to model detailed camera characteristics, including the focal length, image center, and distortion model.</p> <p>Camera models are most commonly used in robotics applications to determine the relationship of the camera to the object under inspection.</p> <p>Because a camera model includes a distortion model, you do not need to compute a separate distortion model.</p>
Microplane Calibration	Use microplane calibration when the working plane is nonlinear.

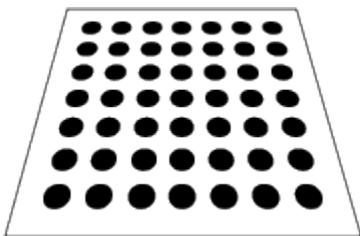
Simple Calibration

The simple calibration algorithm performs a direct conversion of pixel coordinates and real-world units.

Perspective Calibration

The perspective calibration algorithm computes a pixel to real-world mapping for the entire image, which allows you to easily convert between pixel coordinates and real-world units.

The following figure contains multiple dots of the same size in real world values which are distorted by perspective projection:



Distortion Modeling

A distortion model uses one or more calibration grids to model distortion introduced by lens imperfections and correct distortion for the entire image. Distortion modeling can model radial and tangential distortion.

The following figures illustrate typical radial lens distortion.

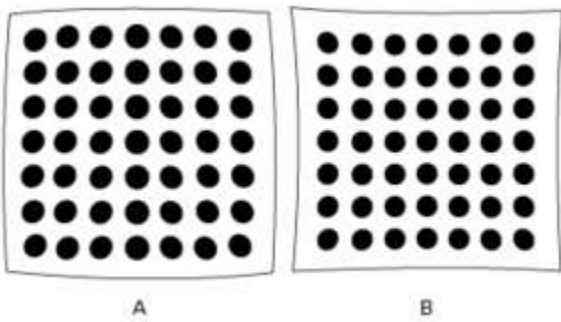


Figure A illustrates barrel distortion and figure B illustrates pincushion distortion. You can use both the division model and the polynomial model to correct for radial distortion.

Tangential distortion occurs when the camera sensor is not aligned with the optical axis. Use the polynomial model if your image exhibits tangential distortion.

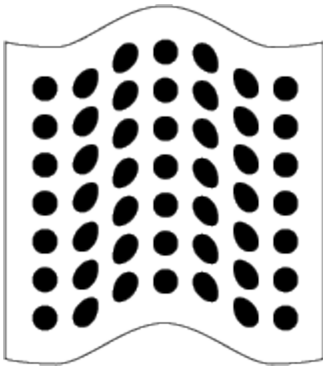
Camera Modeling

A camera model uses multiple calibration grids to model detailed camera characteristics, including the focal length, image center, and distortion model. Using a camera model, you can apply mathematical calculations to determine values such as the pose of an object.

Microplane Calibration

The microplane calibration algorithm computes pixel to real-world mappings in a rectangular region centered on each point in a calibration grid. Vision interpolates the mapping information around each point based on neighboring points.

Use microplane calibration to correct distortion introduced by a nonlinear working plane. The following figure illustrates nonlinear distortion:



Concepts

The calibration software uses a calibration algorithm and a list of known pixel to real-world mappings to compute calibration information for the entire image. The calibration software uses these known mappings to compute the pixel to real-world mapping for the entire image. Individual calibration algorithms may have specific requirements for creating the list of pixel to real-world mappings.

After you calibrate an image, you can define a calibration axis in order to express pixel measurements in real-world units, or spatially correct a portion of a distorted image.

You can also review statistical results to evaluate the quality or state of your calibration system.

Mapping Pixel Coordinates to Real-World Coordinates

You can specify a list of pixel to real-world mappings in two ways, depending on the calibration algorithm you select. You can manually map pixel coordinates to real-world coordinates, or you can use a calibration grid.

The resulting calibration information is valid only for the imaging setup that you used to create the mapping. Any change in the imaging setup that violates the mapping information compromises the accuracy of the calibration information.

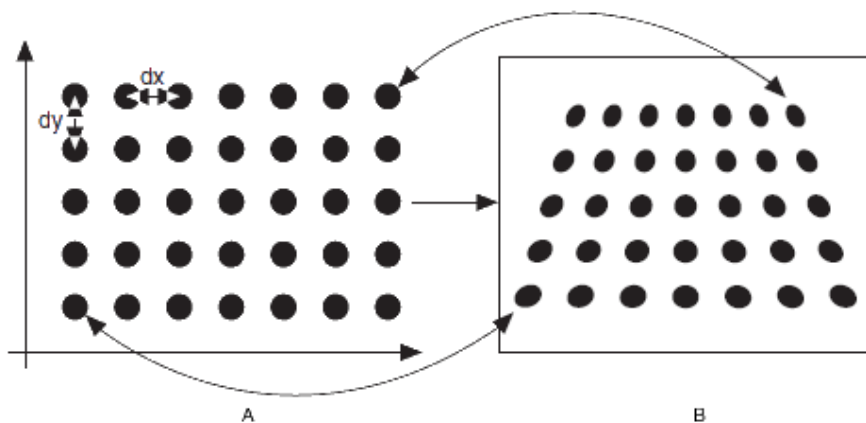
Defining Mappings Manually

To define mappings manually, input a list of real world points and the corresponding pixel coordinates to the calibration software. The following table describes the algorithms that allow you to define mappings manually:

Name	Description
Simple Calibration	Simple calibration transforms pixel coordinates to real-world coordinates by scaling the image horizontally and vertically. Provide the calibration software with the horizontal and vertical distance between pixels in real-world units.
Perspective Calibration	Provide a set of pixel to real-world mappings to perspective calibration to correct the perspective distortion. A minimum of 4 pixel to real-world mappings are required to correct the perspective distortion, but additional mappings may provide better results.

Using a Calibration Grid

A calibration grid consists of a grid of equidistant points similar to the grid of dots shown in figure A.



To use a calibration grid, provide the calibration with the horizontal (dx) and vertical (dy) spacing between the points in real-world units. The calibration software uses the image of the grid, shown in figure B, and the spacing between the dots in the grid to generate the list of pixel to real-world mappings required for the calibration process.

The following algorithms support using a calibration grid:

- Distortion Model,
 - Division
 - Polynomial

- Camera Model,
- Microplane Calibration.

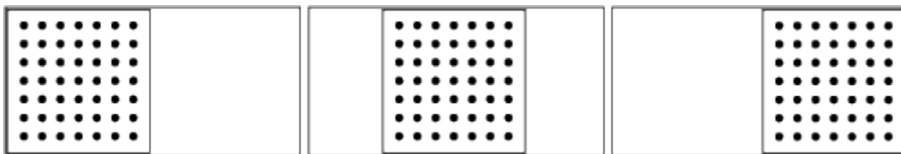
Refer to the following guidelines to achieve accurate results with a calibration grid:

- The calibration grid should cover most of the field of view or area under inspection.
- A minimum of 4 pixel to real-world mappings are required to correct distortion in the image, but additional mappings may provide better results.

Using Multiple Calibration Grid Images

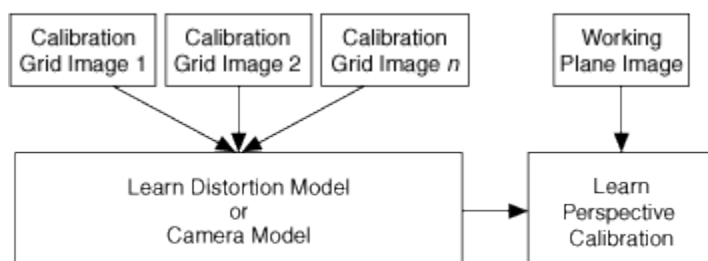
Some calibration algorithms may require multiple calibration grids. For example, a distortion model can only be learned for points that are present in a calibration grid image. If the points do not cover the entire field of view or area under inspection, the distortion model may be inaccurate.

Vision calibration software can use multiple calibration grid images. The following figure illustrates multiple calibration grid images obtained by repositioning the calibration grid within a single field of view.



When multiple images are supplied, the calibration software uses a least square method to optimize the distortion model. After learning the distortion model, you must perform perspective calibration to set the working plane on which you want to make measurements and enable pixel to real-world mappings.

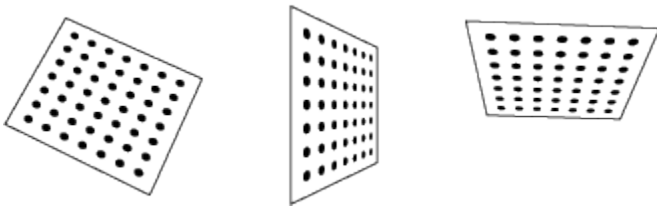
The following figure illustrates the steps involved in using multiple calibration grid images.



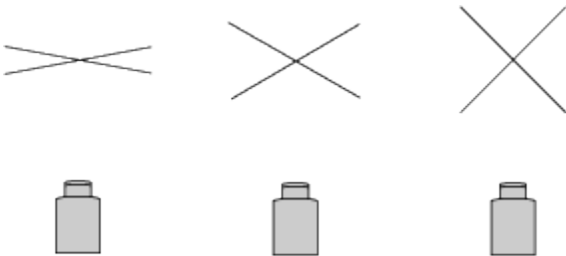
First, learn the distortion or camera model using points from multiple calibration grid images. Then learn the perspective calibration for the working plane on which you want to make the measurement. If the working plane changes, you must relearn the perspective calibration.

Using Multiple Calibration Grid Images in Multiple Planes

To compute a camera model, you must provide multiple grid images from at least three different projection planes. The following figure illustrates the same calibration grid in multiple projection planes.

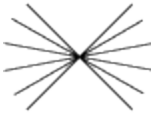


For accurate results, the calibration grids used to calculate the camera model should cover a minimum relative angle of 45 degrees. The following figures illustrate different relative angle coverages:



The leftmost figure illustrates two calibration grids with a relative angle range of 20 degrees. The rightmost figure illustrates two calibration grids with a relative angle of 90 degrees. A camera model calculated for the left figure would be less accurate than the camera model calculated for the right figure. For example, the point where the grids intersect in the right figure is much more clearly defined than in the other figures. If the relative angle between calibration grids is too small, the calibration software indicates that it does not have sufficient data to compute the camera model.

For the most accurate results, use multiple calibration grid images to compute a camera model, as illustrated in the following figure:



Calibration Axis

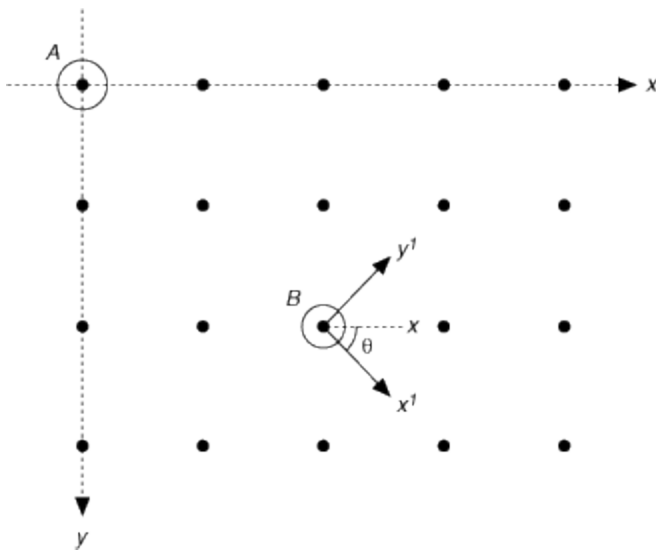
To express measurements in real-world units, you must define a calibration axis. To define a calibration axis, specify the following information:

- The origin of the calibration axis, expressed in pixel coordinates.
- The angle between the calibrated x-axis and the horizontal axis of the image, expressed in degrees.
- The direction of the calibrated vertical axis, either direct or indirect.

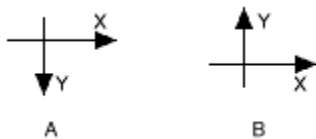
The following figure illustrates a default calibration axis and a user-defined calibration axis. The origins of the coordinate systems lie at the center of the circled dots. Point A indicates the origin for a default calibration axis starting at the top leftmost pixel of an image. Point B indicates the origin of a user-defined calibration axis.

The calibration axis angle, defined by θ , specifies the orientation of the calibrated x^1 axis with respect to the horizontal axis in the image.

The calibration axis originating at point A uses an indirect vertical axis, while the calibration axis originating at point B uses a direct vertical axis.



The following figures illustrate vertical axis directions. An indirect axis orientation, as shown in figure A, corresponds to a typical digital image where the top left pixel serves as the origin. A direct axis orientation, as shown in figure B, corresponds to the orientation of a real-world cartesian Y-axis.



Default Calibration Axis Definition

If the calibration uses multiple calibration grid images, the calibration axis is defined in the working plane image.

The calibration process defines a default calibration axis as follows:

1. The origin is set according to the following conditions:
 - If you use manually defined reference points, the origin is placed at point 0, 0 relative to the points you define.
 - If you use a calibration grid image, the origin is placed at the center of the left, top-most point in the calibration grid image.
2. The angle is set to zero. This aligns the x-axis with the topmost row of points in the calibration grid image.
3. The vertical axis direction is set to indirect. This aligns the y-axis to the leftmost column of points in the calibration grid image.

If you specify a list of points instead of using a calibration grid, the points define the default calibration axis origin, angle, and vertical axis direction.

Redefining a Calibration Axis

You can use Vision software to redefine a calibration axis. For example, you may want to define a calibration axis based on measurements taken from a part under inspection.

The following figure shows an inspection application whose objective is to determine the location of the hole in the board with respect to the corner of the board. The board is on a stage that can translate in the x and y directions and can rotate about its center. The corner of the board is located at the center of the stage.

In the initial setup, shown in figure A, the defined calibration axis aligns with the corner of the board using the following parameters:

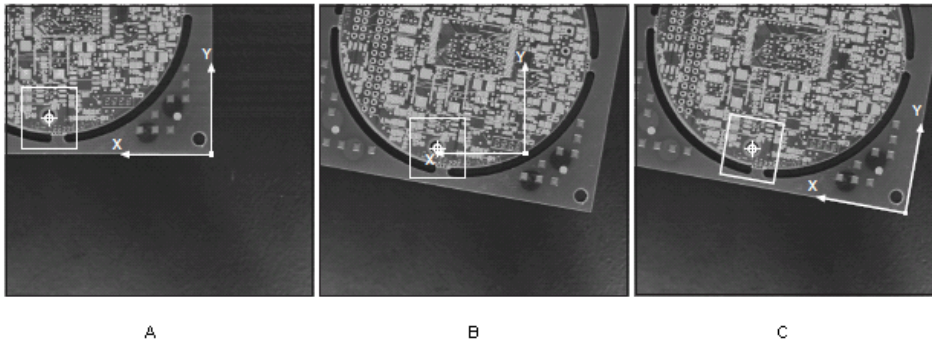
- The origin of the calibration axis is defined as the location in pixels of the corner of the board.
- The angle of the calibration axis is set to 180 degrees.
- The axis direction is set to indirect.

In this example, you can use pattern matching to find the location in pixels of the hole, as illustrated by the crosshair mark in figure A. Convert the location of the hole in pixels to a real world location. This conversion returns the real world location of the hole with respect to the defined calibration axis.

During the inspection process, the stage may translate and rotate by a known amount. This causes the board to occupy a new location in the camera's field of view, which makes the board appear translated and rotated in subsequent images, as shown in figure B. Because the board has moved, the original calibration axis no longer aligns with the corner of the board. Therefore, measurements made using this calibration axis are inaccurate.

Use the information about how much the stage has moved to determine the new location of the corner of the board in the image. Use the Set Calibration function to update the calibration axis to reflect the new position, as illustrated in figure C. The origin of the updated calibration axis becomes the new pixel location of the corner of

the board, and the angle of the calibration axis is the angle by which the stage has rotated.



Calibration Quality Information

Distortion is specified in relative terms. For example, a lens which exhibits 2 percent barrel radial distortion over a given field will image a point in the corner of the field 2 percent too far from the optical axis. In the resulting image, a corner that should be 400 pixels from the optical axis measures 408 pixels away from the optical axis.

Vision calibration software provides a percentage distortion statistic to indicate the quality of a calibrated system. Vision software calculates the error divided by the distance from the optical axis for each pixel. The average result is presented as the percentage distortion statistic.

Use the percentage distortion statistic to determine whether the selected calibration algorithm is adequate for your application. For example, you may receive a high percentage distortion statistic if you use perspective calibration or a sparse calibration grid to correct an image exhibiting nonlinear or lens distortion. You can also use the percentage distortion statistic to determine whether there is a problem with the calibrated system. For example, if your lens introduces negligible distortion, a high percentage distortion statistic may indicate a problem such as a physically distorted calibration template.

Error Map

Vision calibration software computes an error map, along with the following error statistics:

- Mean error,

- Maximum error,
- Standard deviation.

The error map is an estimate of the positional error that you can expect when you convert a pixel coordinate into a real-world coordinate. The error map is a 2D array that contains the expected positional error for each pixel in the image.

The error value of the pixel coordinate (i, j) indicates the largest possible location error for the estimated real-world coordinate (x, y) as compared to the true real-world location. The following equation shows how to calculate the error value.

$$e(i, j) = \sqrt{(x - x_{true})^2 + (y - y_{true})^2}$$

The error value indicates the radial distance from the true real world position in which the estimated real world coordinates can exist. The error value has a confidence interval of 95%, which implies that the positional error of the estimated real-world coordinate is equal to or smaller than the error value 95% of the time. A pixel coordinate with a small error value indicates that its estimated real-world coordinate is computed very accurately. A large error value indicates that the estimated real-world coordinate for a pixel may not be accurate.

Use the error map to determine whether your imaging setup and calibration information satisfy the accuracy requirements of your inspection application. If the error values are greater than the positional errors that your application can tolerate, you need to improve your imaging setup. An imaging system with high lens distortion usually results in an error map with high values. If you are using a lens with considerable distortion, you can use the error map to determine the position of the pixels that satisfy the accuracy requirements of your application. Because the effect of lens distortion increases toward the image borders, pixels close to the center of the image have lower error values than the pixels at the image borders.

Image Correction

Image correction involves transforming a distorted image acquired in a calibrated setup into an image where perspective errors and lens distortion are corrected. Vision corrects an image by applying the transformation from pixel to real-world coordinates for each pixel in the input image. Then Vision applies simple shift and scaling transformations to position the real-world coordinates into a new image. Vision uses

interpolation during the scaling process to generate the new image.

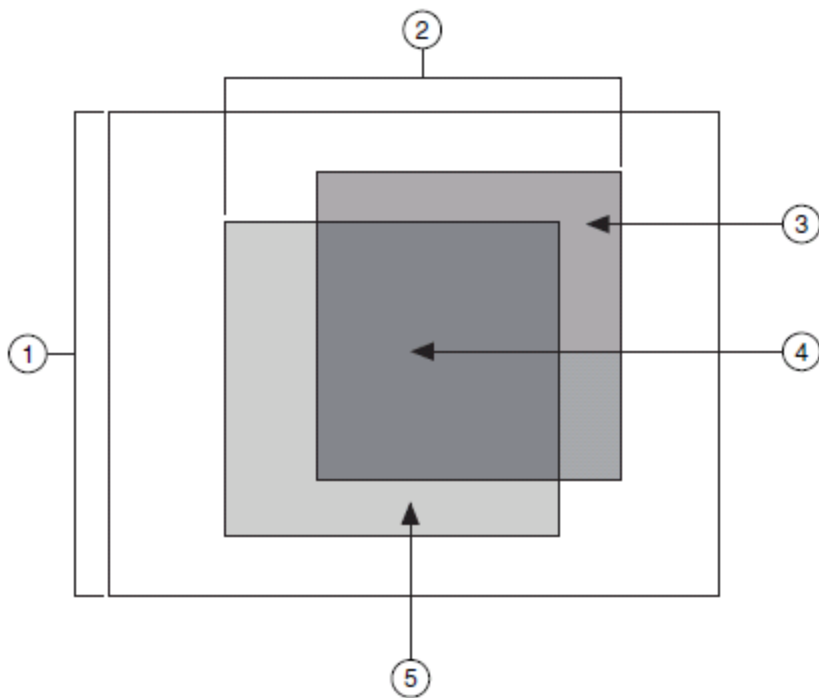
When you learn for correction, you have the option of constructing a correction table. The correction table is a lookup table, stored in memory, that contains the real-world location information of all the pixels in the image. The lookup table greatly increases the speed of image correction but requires more memory and increases your learning time. Use this option when you want to correct several images at a time in your vision application.



Tip Correcting images is a time-intensive operation. You may be able to get the measurements you need without image correction. For example, you can use Vision particle analysis functions to compute calibrated measurements directly from an image that contains calibration information but has not been corrected. Also, you can convert pixel coordinates returned by edge detection tools into real-world coordinates.

Correction Area

You can correct an entire image or regions in the image based on user-defined ROIs or the calibration ROI defined by the calibration software. The following figure illustrates the different image areas you can specify for correction. Vision learns calibration information for only the regions you specify.



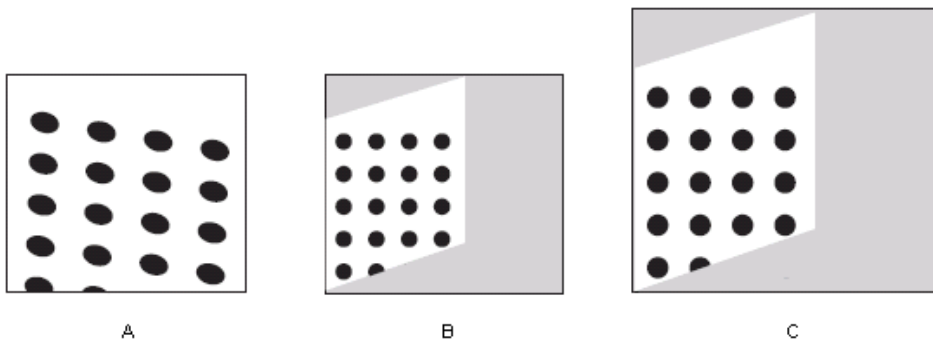
1. Full Image—Corrects the entire image regardless of the calibration ROI and the user-defined ROI.
2. User or Calibration ROI—Corrects pixels in both the user-defined ROI and the calibration ROI.
3. User ROI—Corrects only the pixels inside the user-defined ROI specified during the learn calibration phase.
4. User and Calibration ROI—Corrects only the pixels that lie in the intersection of the user-defined ROI and the calibration ROI.
5. Calibration ROI—Corrects only the pixels inside the calibration ROI. The calibration ROI is computed by the calibration algorithm.

The valid coordinate indicates whether the pixel coordinate you are trying to map to a real-world coordinate lies within the image region you corrected. For example, if you corrected only the pixels within the calibration ROI but you try to map a pixel outside the calibration ROI to real-world coordinates, the Corrected Image Learn ROI parameter indicates an error.

Scaling Mode

The scaling mode defines how to scale a corrected image. Two scaling mode options are available: scale to fit and scale to preserve area. The following illustrates the scaling modes. Figure A shows the original image. With the scale to fit option, the

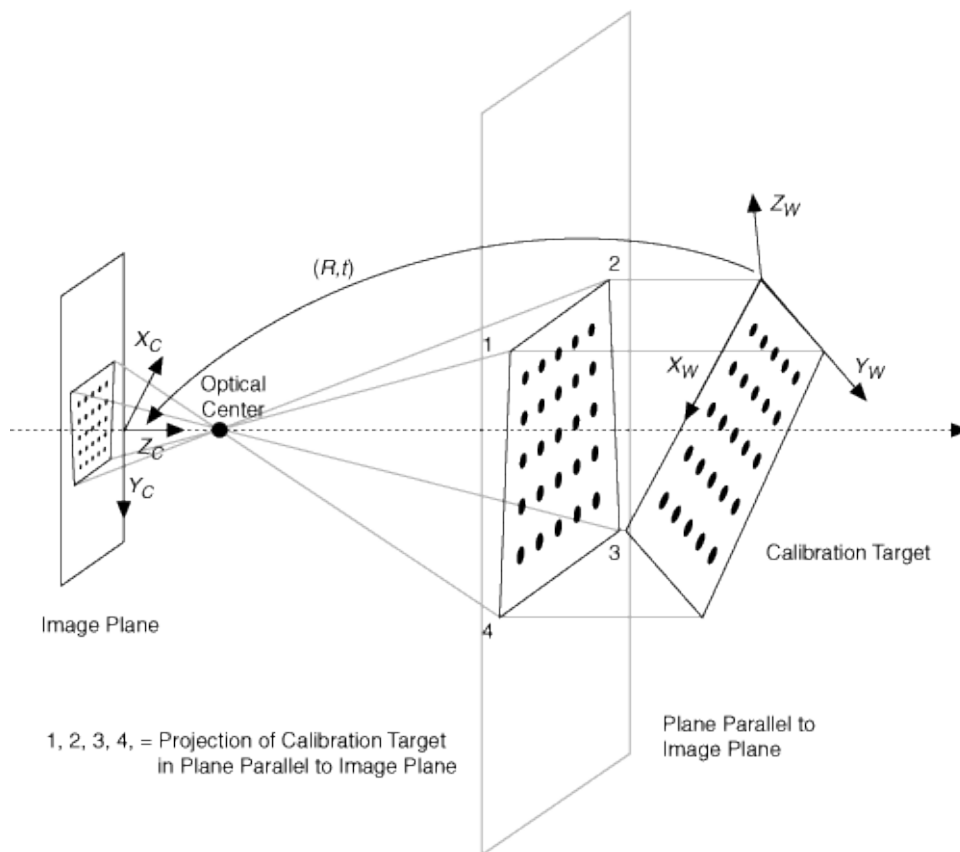
corrected image is scaled to fit in an image the same size as the original image, as shown in figure B. With the scale to preserve area option, the corrected image is scaled such that features in the image retain the same area as they did in the original image, as shown in figure C. Images that are scaled to preserve area are usually larger than the original image. Because scaling to preserve the area increases the size of the image, the processing time for the function may increase.



In-Depth Discussion

This section describes the factors responsible for producing an image and how those factors can be used to map real-world coordinates to image coordinates. This section also describes how Vision software corrects for distortion, and how you can calculate the pose of an object in an image.

The following illustration shows projective mapping:



where:

- the calibration target is represented in a real-world coordinate system (X_w, Y_w, Z_w) ,
- the camera coordinate system is represented as (x_c, y_c, z_c) , with the z axis aligned with the optical axis and the x and y axes aligned with the horizontal and vertical axes of the image plane,
- an intermediate plane, parallel to the image plane, illustrates the shape of the calibration target in an image.

A camera coordinate axis is displaced from a real-world coordinate axis by the following transformation: $P_c = R(P_w - T)$

where:

- P_w is a point in the real-world coordinate system,
- P_c is the homogenous point in the camera coordinate system,
- R is the rotation matrix between the real-world coordinate axis and the camera

coordinate axis,

- T is the origin of the real-world coordinate axis minus the camera coordinate axis.

Homography

A physical projection, or homography, defines a geometric mapping of points from one plane to another. In calibration, a homography can describe the conversion of 3D coordinates in the real world to pixel coordinates in the image.

A physical projection transformation maps the real-world point $P(X_w, Y_w, Z_w)$ to the image point $p(x_w, y_w)$.



Note Because p is defined in homogenous coordinates, you must divide through by z to recover the actual image coordinates.

A typical homography can be expressed as $p = sHP$,

where:

- p is the image plane projection expressed in camera coordinates $[X_c Y_c 1]^T$ that correspond to 2D image coordinates,
- P is a real-world point expressed in 3D real world coordinates $[X_w Y_w Z_w 1]^T$,
- s is a scaling factor,
- H is homography.

In Vision calibration software, homography (H) is a 3×3 matrix which is the product of two matrices: a camera matrix (M) and a homography matrix (W).

Camera Matrix

A camera matrix (M) describes the following internal camera parameters:

- focal length

- principle point position
- pixel size
- pixel skew angle

Internal camera parameters are represented with the following matrix:

$$M = \begin{bmatrix} f_x & \alpha f_x & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

where,

$$f_x = \frac{F}{s_x}$$

$$f_y = \frac{F}{s_y}$$

and

- F is the focal length, in millimeters,
- s_x is the horizontal size of a pixel in the camera sensor, in pixels per millimeter,
- s_y is the vertical size of a pixel in the camera sensor, in pixels per millimeter,
- c_x is the horizontal displacement of the imager from the optical axis, in millimeters,
- c_y is the vertical displacement of the imager from the optical axis, in millimeters,
- α is the pixel skew angle of y with respect to x. α is typically equal to 0.



Note Because α is typically equal to 0, M is defined as:

$$M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

The camera focal length (F) and pixel dimensions (s_x, s_y) cannot be directly calculated. Camera calibration can only calculate the derivative focal length and pixel dimension combinations (f_x, f_y).

Homography Matrix

A homography matrix (W) consists of the rotation matrix and translation vector that relate a point in a real-world plane and a point in an image plane. The physical projection transformation matrix can be expressed as:

$$W = [R \ t]$$

where:

- R equals the rotation matrix,
- t equals the translation vector.

The rotation matrix (R) can be expressed as 3 separate 3×1 matrices, so that:

$$R = [r_1 \ r_2 \ r_3]$$

Thus, the original homography:

$$p = sHP$$

can be expressed as:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = sM[r_1 \ r_2 \ r_3 \ t] \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix}$$

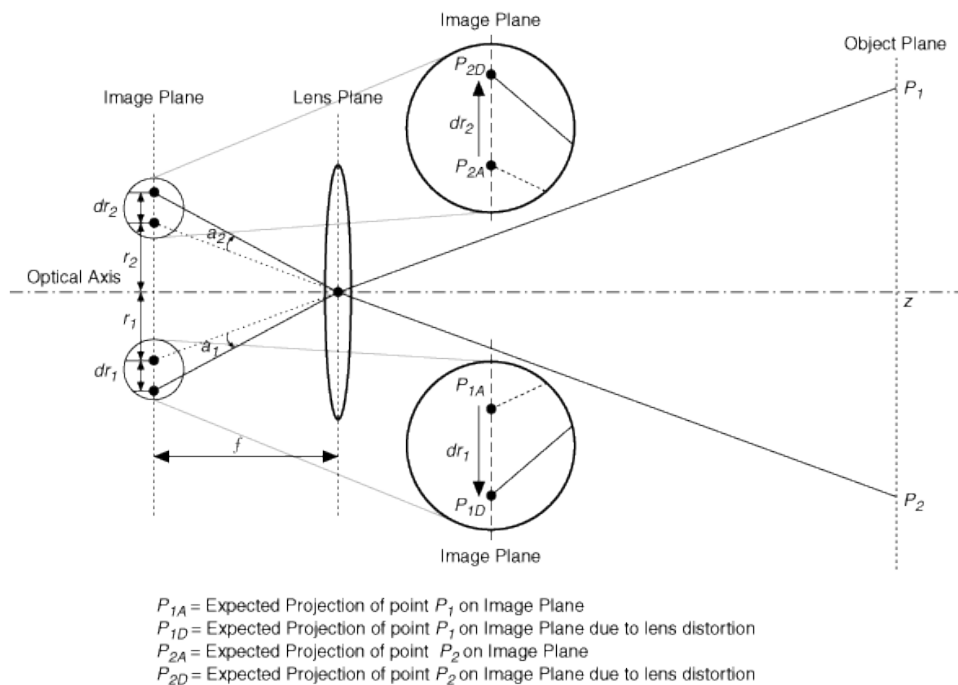
Because calibration is performed with a planar calibration target, we can generalize that $Z = 0$ to further simplify the homography as:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = sM \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

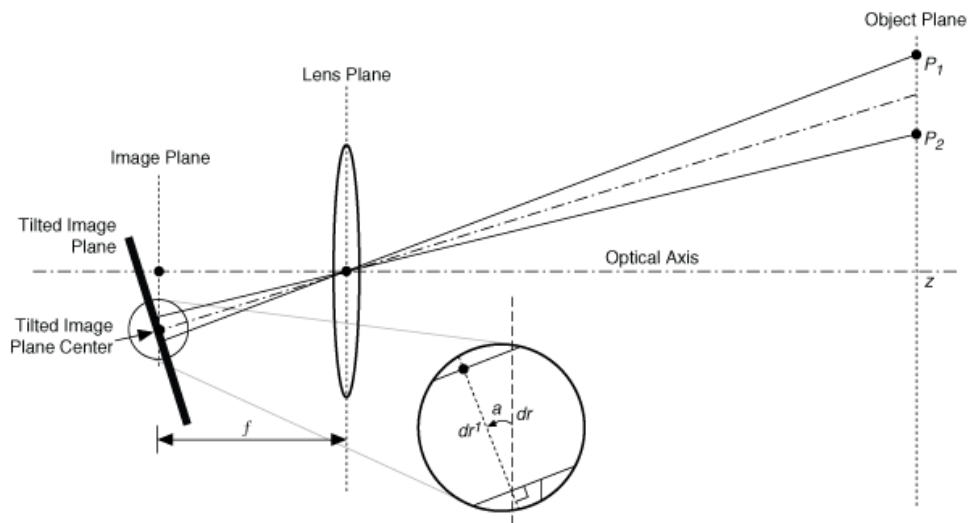
Lens and Camera Distortion

There are two common types of distortion. Lens characteristics may introduce radial distortion, while a misalignment of the lens and camera sensor may introduce tangential distortion.

The following illustration shows radial distortion:



The following illustration shows tangential distortion:



Vision provides division and polynomial distortion models which can correct distortion in an image.

Division Distortion Model

The division distortion model can correct radial distortion. The division distortion model uses a single coefficient parameter (K) to model distortion.

The division distortion model can be represented as:

$$x_{corrected} = \frac{2x}{1 + \sqrt{1 - 4K(x^2 + y^2)}}$$

$$y_{corrected} = \frac{2y}{1 + \sqrt{1 - 4K(x^2 + y^2)}}$$

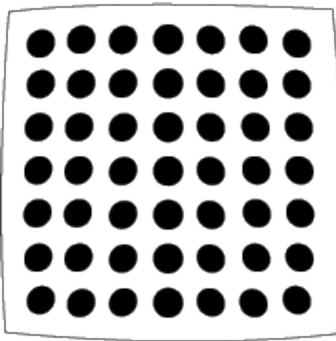
.

where,

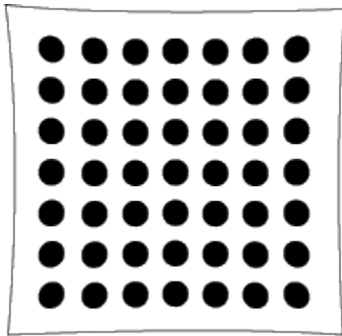
$K > 0$ corrects pincushion distortion,

$K < 0$ corrects barrel distortion.

The following illustration shows barrel distortion:



The following illustration shows pincushion distortion:



Use the inverse equations to distort corrected coordinates:

$$x = \frac{x}{1 + k(x^2 + y^2)}$$

$$y = \frac{y}{1 + k(x^2 + y^2)}$$

Polynomial Distortion Model

The polynomial distortion model can correct both radial and tangential distortion.

Using the Polynomial Distortion Model to Correct Radial Distortion

The polynomial distortion model uses one or more coefficient parameters (K) to model distortion. The distortion model for radial distortion can be represented as:

$$x_{corrected} = x \left(1 + K_1 r^2 + K_2 r^4 + K_3 r^6 + K_4 r^8 + K_5 r^{10} + K_n + r^{(n \times 2)} \right)$$

$$y_{corrected} = y \left(1 + K_1 r^2 + K_2 r^4 + K_3 r^6 + K_4 r^8 + K_5 r^{10} + K_n + r^{(n \times 2)} \right)$$

You can specify the number of coefficients required to model radial distortion for your specific lens.

Using the Polynomial Distortion Model to Correct Tangential Distortion

The polynomial distortion model uses two parameters, P_1 and P_2 , to characterize tangential distortion. The distortion model for tangential distortion can be represented as:

Determining the Pose of an Object

The Homography section explains that a typical homography can be expressed as $p = sHP$

where:

- p is the image plane projection expressed in camera coordinates $[X_c \ Y_c \ 1]^T$ that correspond to 2D image coordinates,
- P is a real-world point expressed in 3D real world coordinates $[X_w \ Y_w \ Z_w \ 1]^T$,
- s is a scaling factor,
- H is homography.

In Vision calibration software, homography (H) is a 3×3 matrix which is the product of two matrices: a camera matrix (M) and a homography matrix (W).

The Homography Matrix section explains that the original homography $p = sHP$ can be simplified as $p = sM[r_1 \ r_2 \ t]P$

where:

- p is the image plane projection expressed in camera coordinates $[X_c \ Y_c \ 1]^T$ that correspond to 2D image coordinates,
- P equals a real-world point expressed in 3D real world coordinates $[X_w \ Y_w \ Z_w \ 1]^T$,

- s equals a scaling factor,
- M equals the camera matrix:

$$\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

- r_1 and r_2 equal the simplified rotational vector R,
- t equals the translation vector.

Calculating the Pose of an Object

To calculate the pose of the working plane, you must complete the following steps:

1. Learn the camera model and generate the camera matrix (M).
2. Perform perspective correction and generate the homography matrix (W).
3. Using the homography (H) and camera matrix (M) information provided by NI Vision software, apply the following matrix transformation to calculate the rotational and translational coefficients.

$$RR^T = I$$

where I is the identity matrix:

$$\begin{aligned} \| r_1 \| &= \| r_2 \| \\ r^3 &= r_1 \times r_2 \end{aligned}$$

4. Use the calculated coefficients to derive the Euler angles for an object to determine the pose.

Image Processing and Analysis

This section describes conceptual information about image analysis and processing, operators, and frequency domain analysis.

Image Analysis

This section contains information about histograms, line profiles, and intensity measurements.

Image analysis combines techniques that compute statistics and measurements based on the gray-level intensities of the image pixels. You can use the image analysis functions to understand the content of the image and to decide which type of inspection tools to use to solve your application. Image analysis functions also provide measurements that you can use to perform basic inspection tasks such as presence or absence verification.

Histogram

A histogram counts and graphs the total number of pixels at each grayscale level. From the graph, you can tell whether the image contains distinct regions of a certain gray-level value.

A histogram provides a general description of the appearance of an image and helps identify various components such as the background, objects, and noise.

When to Use

The histogram is a fundamental image analysis tool that describes the distribution of the pixel intensities in an image. Use the histogram to determine if the overall intensity in the image is high enough for your inspection task. You can use the histogram to determine whether an image contains distinct regions of certain grayscale values. You also can use a histogram to adjust the image acquisition conditions.

You can detect two important criteria by looking at the histogram.

- **Saturation**—Too little light in the imaging environment leads to underexposure of the imaging sensor, while too much light causes overexposure, or saturation, of the imaging sensor. Images acquired under underexposed or saturated conditions will not contain all the information that you want to inspect from the scene being observed. It is important to detect these imaging conditions and correct for them during setup of your imaging system. You can detect whether a sensor is underexposed or saturated by looking at the histogram. An underexposed image contains a large number of pixels with low gray-level values. This appears as a peak at the lower end of the histogram. An overexposed or saturated image contains a large number of pixels with very high gray-level values.
- **Lack of contrast**—A widely-used type of imaging application involves inspecting and counting parts of interest in a scene. A strategy to separate the objects from the background relies on a difference in the intensities of both, for example, a bright part and a darker background. In this case, the analysis of the histogram of the image reveals two or more well-separated intensity populations. Tune your imaging setup until the histogram of your acquired images has the contrast required by your application.

Concepts

The histogram is the function H defined on the grayscale range $[0, \dots, k, \dots, 255]$ such that the number of pixels equal to the gray-level value k is:

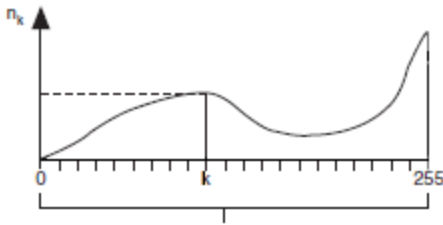
$$H(k) = n_k$$

where

- k is the gray-level value,
- n_k is the number of pixels in an image with a gray-level value equal to k ,
- $\sum n_k$ from $k = 0$ to 255 is the total number of pixels in an image.

The histogram plot in the following figure reveals which gray levels occur frequently and which occur rarely.

The following image shows grayscale range:



Two types of histograms can be calculated: the linear and cumulative histograms.

In both cases, the horizontal axis represents the gray-level value that ranges from 0 to 255. For a gray-level value k , the vertical axis of the linear histogram indicates the number of pixels n_k set to the value k , and the vertical axis of the cumulative histogram indicates the percentage of pixels set to a value less than or equal to k .

Linear Histogram

The density function is

$$H_{linear}(k) = n_k$$

where

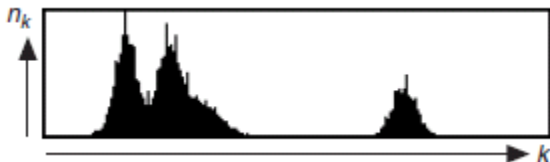
- $H_{Linear}(k)$ is the number of pixels equal to k .

The probability function is

$$P_{linear}(k) = \frac{n_k}{n}$$

where

- $P_{Linear}(k)$ is the probability that a pixel is equal to k .



Cumulative Histogram

The distribution function is

$$H_{Cumul}(k) = \sum_{i=0}^k n_i$$

where

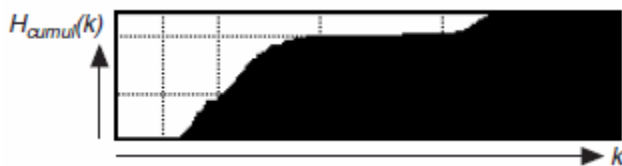
- $H_{Cumul}(k)$ is the number of pixels that are less than or equal to k .

The probability function is

$$P_{Cumul}(k) = \sum_{i=0}^k \frac{n_i}{n}$$

where

- $P_{Cumul}(k)$ is the probability that a pixel is less than or equal to k .



Interpretation

The gray-level intervals featuring a concentrated set of pixels reveal the presence of significant components in the image and their respective intensity ranges.

The **linear histogram** reveals that the image is composed of three major elements. The **cumulative histogram** of the same image shows that the two left-most peaks compose approximately 80% of the image, while the remaining 20% corresponds to the third peak.

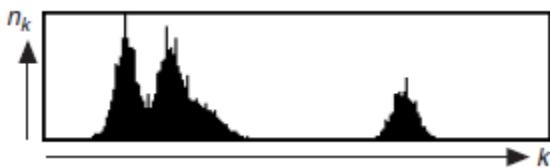
Histogram Scale

The vertical axis of a histogram plot can be shown in a linear or logarithmic scale. A logarithmic scale lets you visualize gray-level values used by small numbers of pixels.

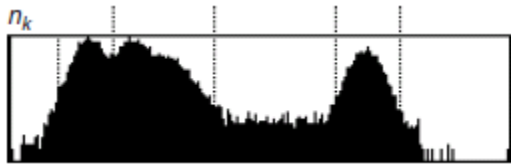
These values might appear unused when the histogram is displayed in a linear scale.

In a logarithmic scale, the vertical axis of the histogram gives the logarithm of the number of pixels per gray-level value. The use of minor gray-level values becomes more prominent at the expense of the dominant gray-level values. The logarithmic scale emphasizes small histogram values that are not typically noticeable in a linear scale. The following figure illustrates the difference between the display of the histogram of the same image in a linear and logarithmic scale. In this particular image, three pixels are equal to 0.

The following image shows a linear vertical scale:



The following image shows a logarithmic vertical scale:



Histogram of Color Images

The histogram of a color image is expressed as a series of three tables, each corresponding to the histograms of the three primary components in the color model in the following table.

Color Model	Components
RGB	Red, Green, Blue
HSL	Hue, Saturation, Luminance

Line Profile

A line profile plots the variations of intensity along a line. It returns the grayscale values of the pixels along a line and graphs it.

When to Use

The line profile utility is helpful for examining boundaries between components, quantifying the magnitude of intensity variations, and detecting the presence of repetitive patterns.

Concepts

The following figure illustrates a typical line profile.



The peaks and valleys represent increases and decreases of the light intensity along the line selected in the image. Their width and magnitude are proportional to the size and intensity of their related regions.

For example, a bright object with uniform intensity appears in the plot as a plateau. The higher the contrast between an object and its surrounding background, the steeper the slopes of the plateau. Noisy pixels, on the other hand, produce a series of narrow peaks.

Intensity Measurements

Intensity measurements measure the grayscale image statistics in an image or regions in an image.

When to Use

You can use intensity measurements to measure the average intensity value in a region

of the image to determine, for example, the presence or absence of a part or a defect in a part.

Concepts

Vision contains the following densitometry parameters:

- **Minimum Gray Value**—Minimum intensity value in gray-level units
- **Maximum Gray Value**—Maximum intensity value in gray-level units
- **Mean Gray Value**—Mean intensity value in the particle expressed in gray-level units
- **Standard Deviation**—Standard deviation of the intensity values

Structural Similarity Index

Structural Similarity (SSIM) Index is an image quality metric. SSIM index is computed for the image with respect to the reference image. The reference image is usually needs to be of perfect quality. This quantitative measure considers three parameters namely luminance, contrast and structural information between the two images to computed the SSIM value.

When to Use

- SSIM can be used in television industry to determine the quality of video streamed from the satellites.
- SSIM can be used as a benchmark to check the performance of other image progressing algorithms, like image compression.

Concepts

The human visual system is adapted to extract structural information. The SSIM algorithm separates out the similarity measurements into three different components:

- Luminance
- Contrast
- Structural

The luminance between the two signals is determined by the mean intensity of the signals. The contrast is determined by the standard deviation. And the structural is

determined by the correlation of the two signals.

$$L_{(x, y)} = \left(2\mu_x\mu_y + C_1 \right) / \left(2\mu_x^2 + \mu_y^2 + C_1 \right)$$

$$C_{(x, y)} = \left(2\sigma_x\sigma_y + C_2 \right) / \left(2\sigma_x^2 + \sigma_y^2 + C_2 \right)$$

$$S_{(x, y)} = \left(\sigma_{xy} + C_3 \right) / \left(\sigma_x\sigma_y + C_3 \right)$$

where

- μ_x is the mean over a window in Image X,
- μ_y is the mean over a window in Image Y,
- σ_x is standard deviation (square root of variance) over a window in Image X,
- σ_y is standard deviation (square root of variance) over a window in Image Y,
- σ_{xy} is co-variance over a window between Image X and Image Y,
- x and y refer to a local window in the Image X and Y respectively,
- C_1 , C_2 and C_3 are constants.

SSIM (x,y) is a multiplication of these three components.

If C_3 is set to $C_2/2$, then over a particular window:

$$SSIM(x, y) = \left((2\mu_x\mu_y + C_1) * (2\sigma_{xy} + C_2) \right) / \left((\mu_x^2 + \mu_y^2 + C_1) * (\sigma_x^2 + \sigma_y^2 + C_2) \right)$$

The Mean-SSIM is the average over all such local windows. The window is moved across the image one pixel at a time.

Normal SSIM

In normal SSIM, a circular symmetric Gaussian weighting function is used calculate the mean values. Choose Normal SSIM when the image has low contrast or does not

contain clear structural information; for example, when the image is a texture sample.

Fast SSIM

Fast SSIM uses a faster approach to calculate the variance and mean values, which are time-consuming in normal SSIM. The gradient images are calculated using the Roberts gradient templates to represent variance. Mean values are calculated by averaging pixels in the local window. Fast SSIM is best suited for images which have clear structural information, such as strong edges. Because it is based on gradient images, Fast SSIM may not be sufficient if the image has low contrast or poor structural information.

Feature Extraction

Use feature extraction to extract important or interesting features from an image.

When to Use

Use feature extraction on images along with classifiers for applications like:

- **Classification**—Identify an unknown sample by comparing a set of its significant features to a set of features that conceptually represent classes of known samples.
- **Image Segmentation**—Segment an image and find defects in parts of an image.
- **Texture Analysis**—Classify textures and identify defects in textures.

Concepts

Histogram of Oriented Gradients Concepts

Vision has 2 feature extraction algorithms: histogram of oriented gradients (HOG) and local binary patterns (LBP).

Histogram of oriented gradients is a technique to extract features from an image. The technique counts occurrences of gradient orientation in localized portions of an image.

The concept behind the HOG descriptors is that local object appearance and shape within an image can be described by the distribution of intensity gradients or edge directions. The implementation of these descriptors can be achieved by dividing the

image into a grid of small connected regions, called blocks, and for each block constructing a histogram of gradient directions for the pixels within the cell. The histograms are concatenated to represent the features. For improved accuracy, the local histograms are normalized by calculating a measure of the gradient across a block, and then using this value to normalize all pixels within the block.

Gradient Computation

The gradients in the x- and y-directions are computed for the whole image or the ROI. The kernels used for the computation of gradients in the x- and y-directions are:

-1	0	1	-1	-1	-1
-1	0	1	0	0	0
-1	0	1	1	1	1

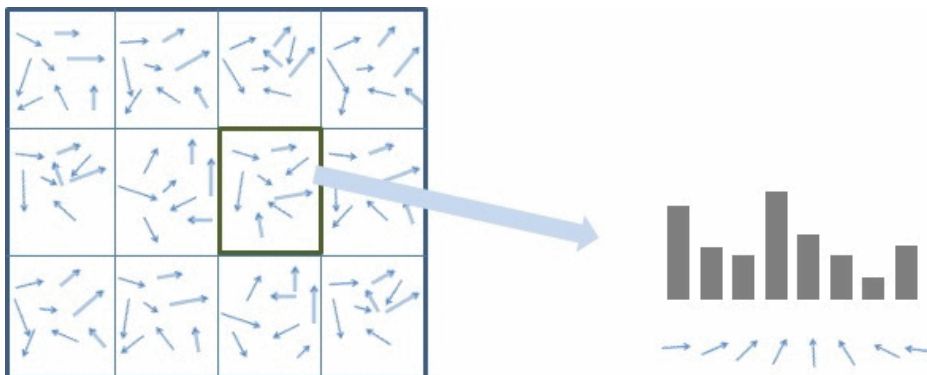
$$Magnitude = \sqrt{I_x^2 + I_y^2}$$

$$Angle = \arctan(I_y / I_x)$$

Histogram Computation

The histogram is computed for each block by binning the gradient magnitude for each angle range. Each histogram is normalized to values ranging between 0 and 1 using the gradient magnitude.

The following graphic shows Local Binary Patterns Concepts:



The following graphic shows Local Binary Patterns Concepts:



Binary Value = 10101010

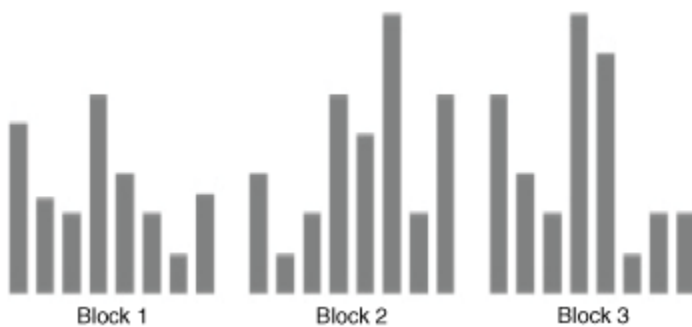
Decimal Value = 170

Histogram Computation

The histogram is computed for each block by counting the LBP values in each bin. Each histogram is normalized to values ranging between 0 and 1.

Grid Based computation for HOG and LBP

The histograms computed for each block give the features local to the block. All the histograms are concatenated to form the final histogram which represents the features for the whole image.



How to Use the Bin Size

The bin size plays an important role in deciding the accuracy in using the features. Use a lower bin size for small images, images with noise, and images which do not have finer details to be captured by the user. Use a higher bin sized for large images, in images with low noise, and in images where finer details play a significant role.

Reference

N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," Proc. IEEE Conf. Computer Vision and Pattern Recognition, 2005.

Image Processing

This section contains information about lookup tables, convolution kernels, spatial filters, and grayscale morphology.

Lookup Tables

The lookup table (LUT) transformations are basic image-processing functions that highlight details in areas containing significant information, at the expense of other areas. These functions include histogram equalization, gamma corrections, logarithmic corrections, and exponential corrections.

When to Use

Use LUT transformations to improve the contrast and brightness of an image by modifying the dynamic intensity of regions with poor contrast.

Concepts

A LUT transformation converts input gray-level values from the source image into other gray-level values in the transformed image.

A LUT transformation applies the transform $T(x)$ over a specified input range $[\text{rangeMin}, \text{rangeMax}]$ in the following manner:

- $T(x) = \text{dynamicMin}$ if $x \leq \text{rangeMin}$
- $f(x)$ if $\text{rangeMin} < x \leq \text{rangeMax}$
- dynamicMax if $x > \text{rangeMax}$

where

- x represents the input gray-level value,
- $\text{dynamicMin} = 0$ (8-bit images) or the smallest initial pixel value (16-bit and floating

- point images),
- $\text{dynamicMax} = 255$ (8-bit images) or the largest initial pixel value (16-bit and floating point images),
- $\text{dynamicRange} = \text{dynamicMax} - \text{dynamicMin}$,
- $f(x)$ represents the new value.

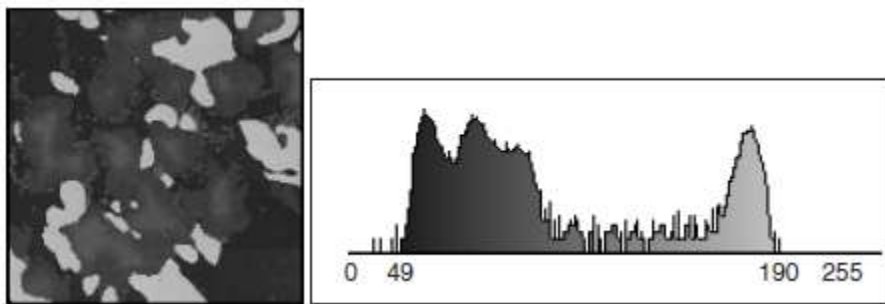
The function scales $f(x)$ so that $f(\text{rangeMin}) = \text{dynamicMin}$ and $f(\text{rangeMax}) = \text{dynamicMax}$. $f(x)$ behaves on $[\text{rangeMin}, \text{rangeMax}]$ according to the method you select.

In the case of an 8-bit resolution, a LUT is a table of 256 elements. The index element of the array represents an input gray-level value. The value of each element indicates the output value.

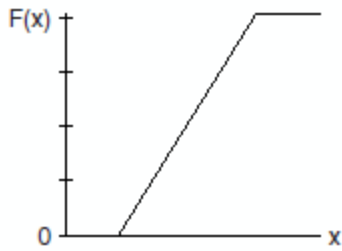
The transfer function associated with a LUT has an intended effect on the brightness and contrast of the image.

Example

The following example uses the following source image. In the linear histogram of the source image, the gray-level intervals $[0, 49]$ and $[191, 254]$ do not contain significant information.



Using the following LUT transformation, any pixel with a value less than 49 is set to 0, and any pixel with a value greater than 191 is set to 255. The interval $[50, 190]$ expands to $[1, 254]$, increasing the intensity dynamic of the regions with a concentration of pixels in the gray-level range $[50, 190]$.

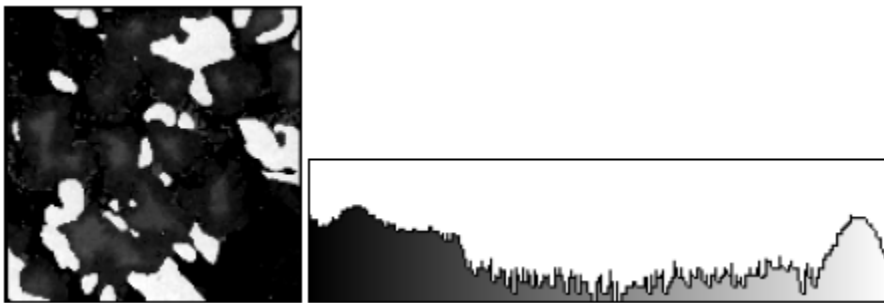


If $x \in (0, 49)$, $F(x) = 0$

If $x \in (191, 254)$, $f(x) = 225$

else $F(x) = 1.81 \times x - 89.5$


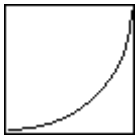
The LUT transformation produces the following image. The linear histogram of the new image contains only the two peaks of the interval [50, 190].



Predefined Lookup Tables

Seven predefined LUTs are available in NI Vision: Linear, Logarithmic, Power 1/Y, Square Root, Exponential, Power Y, and Square. The following table shows the transfer function for each LUT and describes its effect on an image displayed in a palette that associates dark colors to low-intensity values and bright colors to high-intensity values, such as the Gray palette.

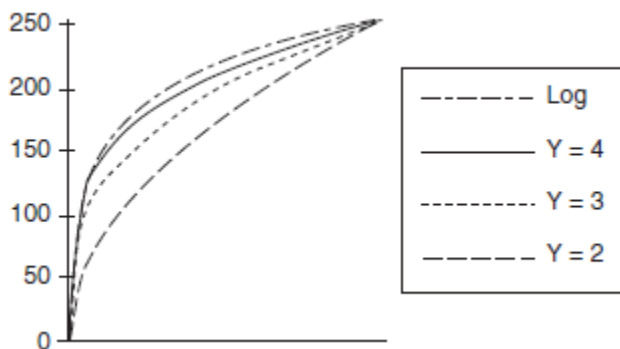
LUT	Transfer Function	Shading Correction
Linear		Increases the intensity dynamic by evenly distributing a given gray-level interval [min, max] over the full gray scale [0, 255]. Min and max default values are 0 and 255 for an 8-bit image.

LUT	Transfer Function	Shading Correction
Logarithmic Power $1/Y$ Square Root		Increases the brightness and contrast in dark regions. Decreases the contrast in bright regions.
Exponential Power Y Square		Decreases the brightness and contrast in dark regions. Increases the contrast in bright regions.

Logarithmic and Inverse Gamma Correction

The **logarithmic** and **inverse gamma corrections** expand low gray-level ranges while compressing high gray-level ranges. When using the Gray palette, these transformations increase the overall brightness of an image and increase the contrast in dark areas at the expense of the contrast in bright areas.

The following graphs show how the transformations behave. The horizontal axis represents the input gray-level range, and the vertical axis represents the output gray-level range. Each input gray-level value is plotted vertically, and its point of intersection with the look-up curve is plotted horizontally to give an output value.



The Logarithmic, Square Root, and Power $1/Y$ functions expand intervals containing low gray-level values while compressing intervals containing high gray-level values.

The higher the gamma coefficient Y , the stronger the intensity correction. The Logarithmic correction has a stronger effect than the Power $1/Y$ function.

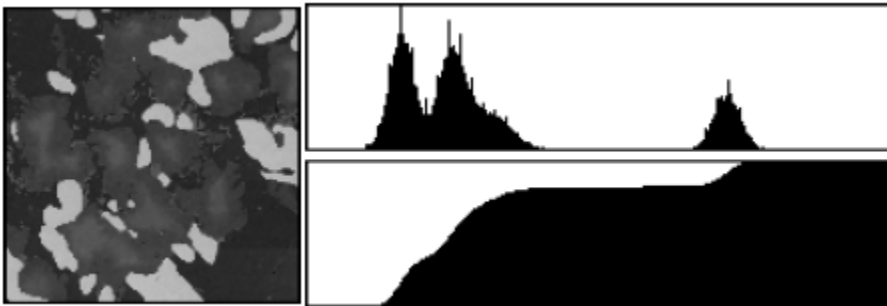
Logarithmic and Inverse Gamma Correction Examples

The following series of illustrations presents the linear and cumulative histograms of an image after various LUT transformations. The more the histogram is compressed on the right, the brighter the image.

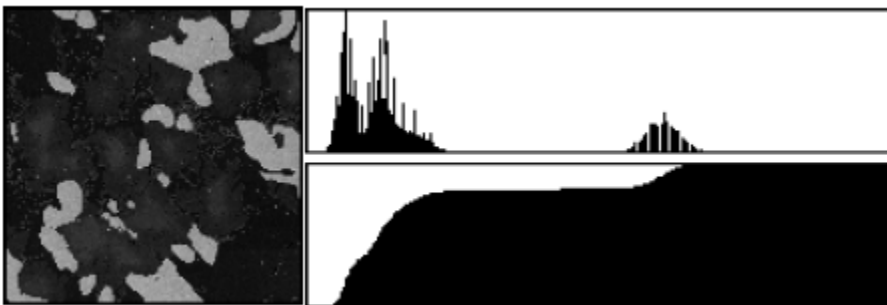


Note Graphics on the left represent the original image, graphics on the top right represent the linear histogram, and graphics on the bottom right represent the cumulative histogram.

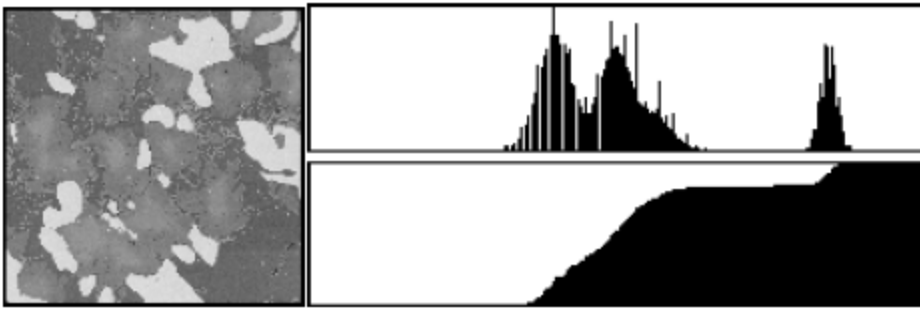
The following illustrations show the original image and histograms.



A Power $1/Y$ transformation (where $Y = 1.5$) produces the following image and histograms.



A Square Root or Power $1/Y$ transformation (where $Y = 2$) produces the following image and histograms.



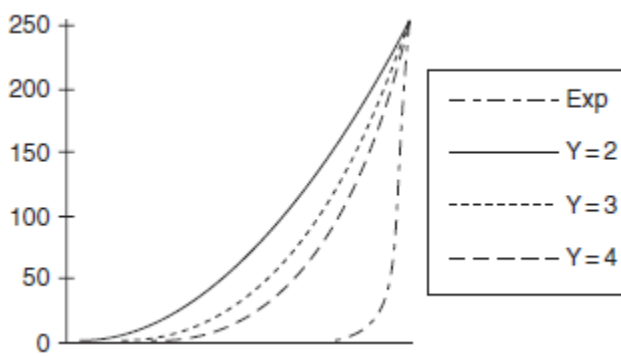
A Logarithm transformation produces the following image and histograms.



Exponential and Gamma Correction

The exponential and gamma corrections expand high gray-level ranges while compressing low gray-level ranges. When using the Gray palette, these transformations decrease the overall brightness of an image and increase the contrast in bright areas at the expense of the contrast in dark areas.

The following graphs show how the transformations behave. The horizontal axis represents the input gray-level range, and the vertical axis represents the output gray-level range. Each input gray-level value is plotted vertically, and its point of intersection with the look-up curve is plotted horizontally to give an output value.



The Exponential, Square, and Power γ functions expand intervals containing high gray-level values while compressing intervals containing low gray-level values.

The higher the gamma coefficient γ , the stronger the intensity correction. The Exponential correction has a stronger effect than the Power γ function.

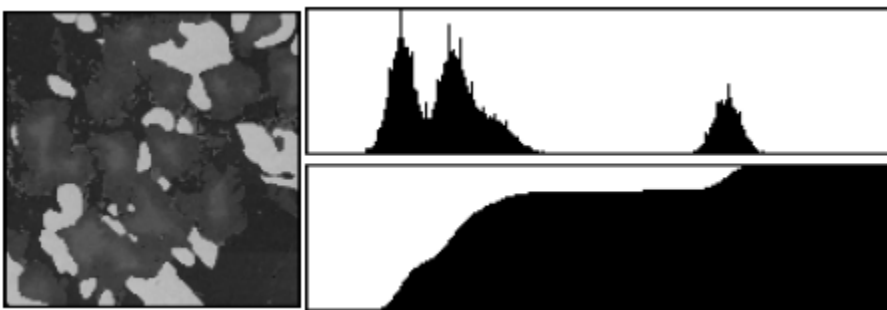
Exponential and Gamma Correction Examples

The following series of illustrations presents the linear and cumulative histograms of an image after various LUT transformations. The more the histogram is compressed, the darker the image.

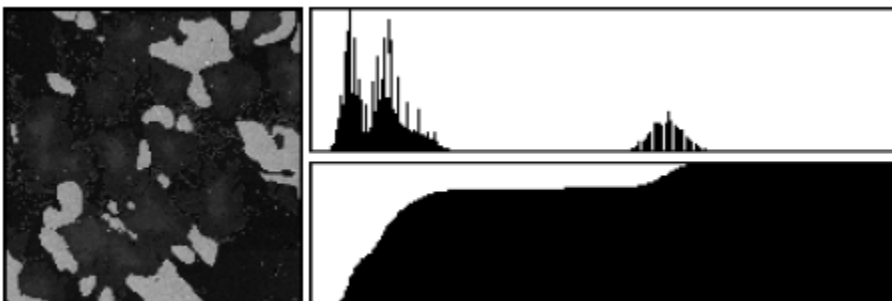


Note Graphics on the left represent the original image, graphics on the top right represent the linear histogram, and graphics on the bottom right represent the cumulative histogram.

The following illustrations show the original image and histograms.

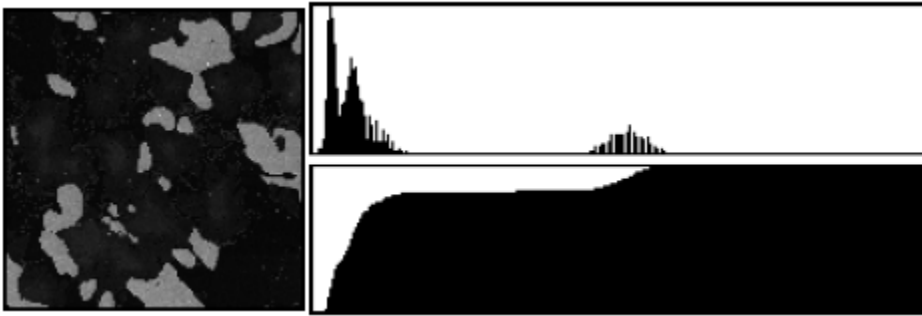


A Power γ transformation (where $\gamma = 1.5$) produces the following image and histograms.



A Square or Power γ transformation (where $\gamma = 2$) produces the following image and

histograms.



An Exponential transformation produces the following image and histograms.



Equalize

The Equalize function is a lookup table operation that does not work on a predefined LUT. Instead, the LUT is computed based on the content of the image where the function is applied.

The Equalize function alters the gray-level values of pixels so that they become evenly distributed in the defined grayscale range, which is 0 to 255 for an 8-bit image. The function associates an equal amount of pixels per constant gray-level interval and takes full advantage of the available shades of gray. Use this transformation to increase the contrast in images that do not use all gray levels.

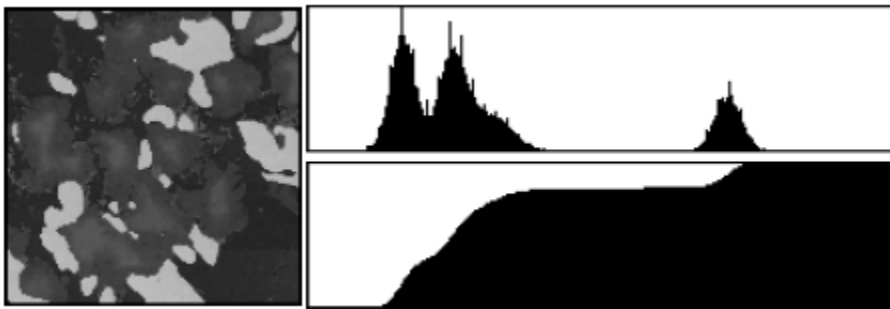
The equalization can be limited to a gray-level interval, also called the equalization range. In this case, the function evenly distributes the pixels belonging to the equalization range over the full interval, which is 0 to 255 for an 8-bit image. The other pixels are set to 0. The image produced reveals details in the regions that have an intensity in the equalization range; other areas are cleared.

Equalization Example

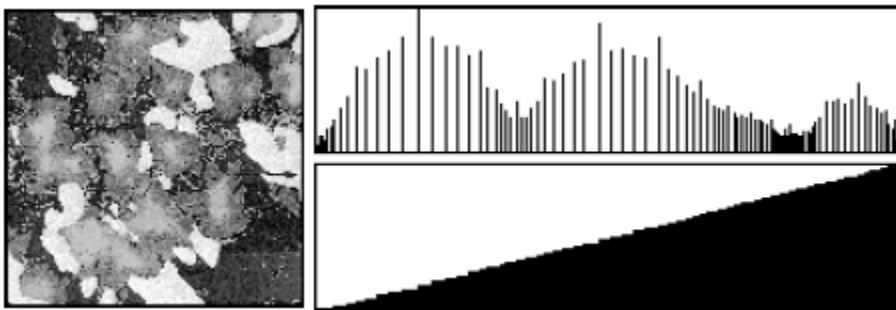
This example shows how an equalization of the interval $[0, 255]$ can spread the information contained in the three original peaks over larger intervals. The transformed image reveals more details about each component in the original image. The following graphics show the original image and histograms.



Note In Examples 1 and 2, graphics on the left represent the original image, graphics on the top right represent the linear histogram, and graphics on the bottom right represent the cumulative histogram.



An equalization from $[0, 255]$ to $[0, 255]$ produces the following image and histograms.

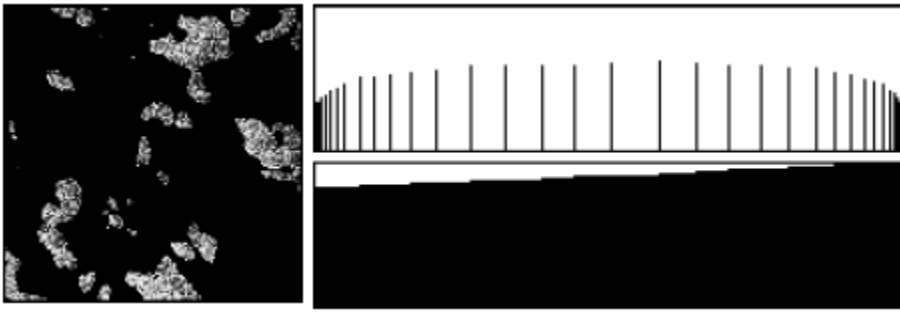


Note The cumulative histogram of an image after a histogram equalization always has a linear profile, as seen in the preceding example.

Equalization Example 2

This example shows how an equalization of the interval $[166, 200]$ can spread the information contained in the original third peak (ranging from 166 to 200) to the interval $[0, 255]$. The transformed image reveals details about the component with the original intensity range $[166, 200]$ while all other components are set to black. An

equalization from [166, 200] to [0, 255] produces the following image and histograms.



Convolution Kernels

A convolution kernel defines a 2D filter that you can apply to a grayscale image. A convolution kernel is a 2D structure whose coefficients define the characteristics of the convolution filter that it represents. In a typical filtering operation, the coefficients of the convolution kernel determine the filtered value of each pixel in the image. Vision provides a set of convolution kernels that you can use to perform different types of filtering operations on an image. You also can define your own convolution kernels, thus creating custom filters.

When to Use

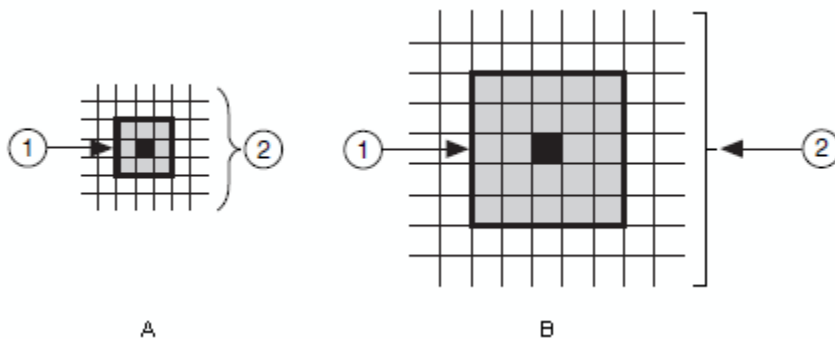
Use a convolution kernel whenever you want to filter a grayscale image. Filtering a grayscale image enhances the quality of the image to meet the requirements of your application. Use filters to smooth an image, remove noise from an image, enhance the edge information in an image, and other related edits.

Concepts

A convolution kernel defines how a filter alters the pixel values in a grayscale image. The convolution kernel is a 2D structure whose coefficients define how the filtered value at each pixel is computed. The filtered value of a pixel is a weighted combination of its original value and the values of its neighboring pixels. The convolution kernel coefficients define the contribution of each neighboring pixel to the pixel being updated. The convolution kernel size determines the number of neighboring pixels whose values are considered during the filtering process.

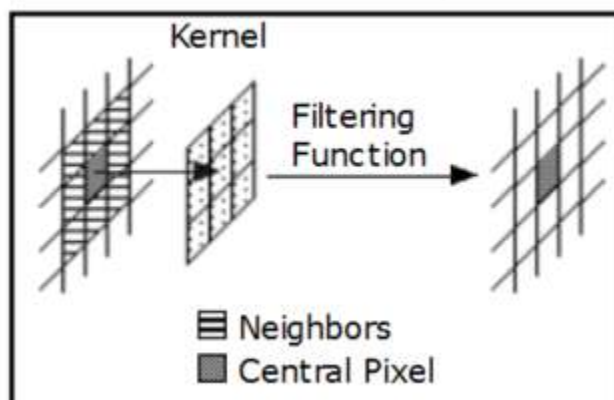
In the case of a 3×3 kernel, illustrated in figure A, the value of the central pixel (shown

in black) is derived from the values of its eight surrounding neighbors (shown in gray). A 5×5 kernel, shown in figure B, specifies 24 neighbors, a 7×7 kernel specifies 48 neighbors, and so forth.

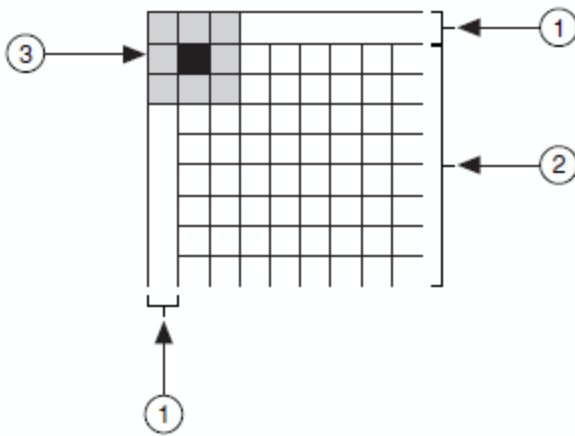


1. Kernel
2. Image

A filtering operation on an image involves moving the kernel from the leftmost and topmost pixel in the image to the rightmost and bottommost point in the image. At each pixel in the image, the new value is computed using the values that lie under the kernel, as shown in the following illustration.



When computing the filtered values of the pixels that lie along the border of the image (the first row, last row, first column, or last column of pixels), part of the kernel falls outside the image. For example, the following figure shows that one row and one column of a 3×3 kernel fall outside the image when computing the value of the topmost leftmost pixel.



1. Border
2. Image
3. Kernel

Vision automatically allocates a border region when you create an image. The default border region is three pixels deep and contains pixel values of 0. You also can define a custom border region and specify the pixel values within the region. The size of the border region should be greater than or equal to half the number of rows or columns in your kernel. The filtering results from along the border of an image are unreliable because the neighbors necessary to compute these values are missing, therefore decreasing the efficiency of the filter, which works on a much smaller number of pixels than specified for the rest of the image. For more information about border regions, refer to the ***digital images*** section.

Related concepts:

- [Digital Images](#)

Spatial Filtering

Filters are divided into two types: linear (also called convolution) and nonlinear.

A convolution is an algorithm that consists of recalculating the value of a pixel based on its own pixel value and the pixel values of its neighbors weighted by the coefficients of a convolution kernel. The sum of this calculation is divided by the sum of the elements in the kernel to obtain a new pixel value. The size of the convolution kernel does not have a theoretical limit and can be either square or rectangular (3×3 , 5×5 ,

5×7 , 9×3 , 127×127 , and so on).

Convolutions are divided into four families:

- Gradient
- Laplacian
- Smoothing
- Gaussian

This grouping is determined by the convolution kernel contents or the weight assigned to each pixel, which depends on the geographical position of that pixel in relation to the central kernel pixel.

Vision features a set of standard convolution kernels for each family and for the usual sizes (3×3 , 5×5 , and 7×7). You also can create your own kernels and choose what to put into them. The size of the user-defined kernel is virtually unlimited. With this capability, you can create filters with specific characteristics.

When to Use

Spatial filters serve a variety of purposes, such as detecting edges along a specific direction, contouring patterns, reducing noise, and detail outlining or smoothing. Filters smooth, sharpen, transform, and remove noise from an image so that you can extract the information you need.

Nonlinear filters either extract the contours (edge detection) or remove the isolated pixels. NI Vision has six different methods you can use for contour extraction (Differentiation, Gradient, Prewitt, Roberts, Sigma, or Sobel). The Canny Edge Detection filter is a specialized edge detection method that locates edges accurately, even under low signal-to-noise conditions in an image.

To harmonize pixel values, choose between two filters, each of which uses a different method: NthOrder and LowPass. These functions require that either a kernel size and order number or percentage is specified on input.

Spatial filters alter pixel values with respect to variations in light intensity in their neighborhood. The neighborhood of a pixel is defined by the size of a matrix, or mask, centered on the pixel itself. These filters can be sensitive to the presence or absence of

light-intensity variations.

Spatial filters fall into two categories:

- **Highpass filters** emphasize significant variations of the light intensity usually found at the boundary of objects. Highpass frequency filters help isolate abruptly varying patterns that correspond to sharp edges, details, and noise.
- **Lowpass filters** attenuate variations of the light intensity. Lowpass frequency filters help emphasize gradually varying patterns such as objects and the background. They have the tendency to smooth images by eliminating details and blurring edges.

Concepts

The following table describes the different types of spatial filters.

Filter Type	Filters
Linear Highpass	Gradient, Laplacian
Linear Lowpass	Smoothing, Gaussian
Nonlinear Highpass	Gradient, Roberts, Sobel, Prewitt, Differentiation, Sigma
Nonlinear Lowpass	Median, Nth Order, Lowpass

Linear Filters

A linear filter replaces each pixel by a weighted sum of its neighbors. The matrix defining the neighborhood of the pixel also specifies the weight assigned to each neighbor. This matrix is called the convolution kernel.

If the filter kernel contains both negative and positive coefficients, the transfer function is equivalent to a weighted differentiation and produces a sharpening or

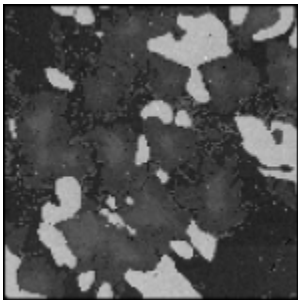
highpass filter. Typical highpass filters include gradient and Laplacian filters.

If all coefficients in the kernel are positive, the transfer function is equivalent to a weighted summation and produces a smoothing or lowpass filter. Typical lowpass filters include smoothing and Gaussian filters.

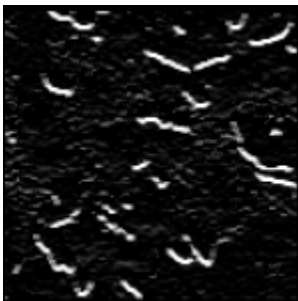
Gradient Filter

A gradient filter highlights the variations of light intensity along a specific direction, which has the effect of outlining edges and revealing texture.

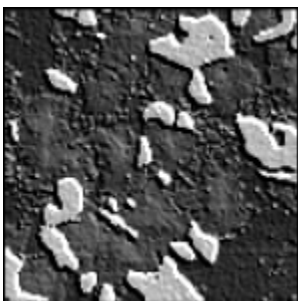
Given the following source image:



A gradient filter extracts horizontal edges to produce the following image.



A gradient filter highlights diagonal edges to produce the following image.



Kernel Definition

A gradient convolution filter is a first-order derivative. Its kernel uses the following model:

$$\begin{bmatrix} a & -b & c \\ b & x & -d \\ c & d & -a \end{bmatrix}$$

where a , b , c , and d are integers and $x = 0$ or 1 .

Filter Axis and Direction

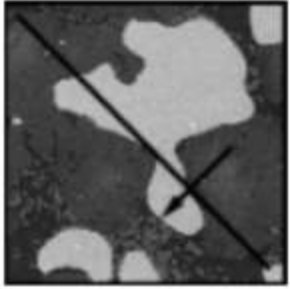
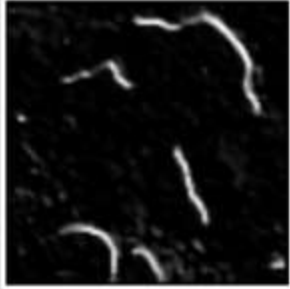
This kernel has an axis of symmetry that runs between the positive and negative coefficients of the kernel and through the central element. This axis of symmetry gives the orientation of the edges to outline. For example, if $a = 0$, $b = -1$, $c = -1$, $d = -1$, and $x = 0$, the kernel is the following:

$$\begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}$$

The axis of symmetry is located at 135° .

For a given direction, you can design a gradient filter to highlight or darken the edges along that direction. The filter actually is sensitive to the variations of intensity perpendicular to the axis of symmetry of its kernel. Given the direction D going from the negative coefficients of the kernel towards the positive coefficients, the filter highlights the pixels where the light intensity increases along the direction D , and darkens the pixels where the light intensity decreases.

The following two kernels emphasize edges oriented at 135° .

Source Image	Prewitt #10	Filtered Image
	$\begin{matrix} 0 & -1 & -1 \\ 1 & 0 & -1 \\ 1 & 1 & 0 \end{matrix}$	

Prewitt #10 highlights pixels where the light intensity increases along the direction going from northeast to southwest. It darkens pixels where the light intensity decreases along that same direction. This processing outlines the northeast front edges of bright regions such as the ones in the illustration.

Source Image	Prewitt #2	Filtered Image
	$\begin{matrix} 0 & -1 & -1 \\ 1 & 0 & -1 \\ 1 & 1 & 0 \end{matrix}$	

Prewitt #2 highlights pixels where the light intensity increases along the direction going from southwest to northeast. It darkens pixels where the light intensity decreases along that same direction. This processing outlines the southwest front edges of bright regions such as the ones in the illustration.

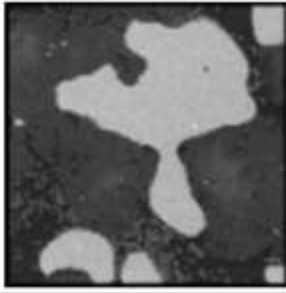
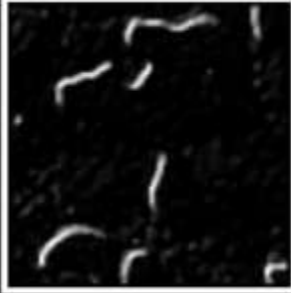


Note Applying Prewitt #10 to an image returns the same results as applying Prewitt #2 to its photometric negative because reversing the lookup table of an image converts bright regions into dark regions and vice versa.

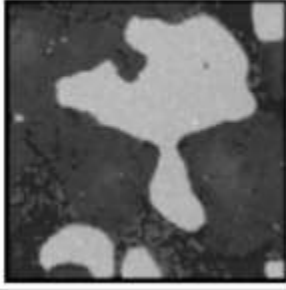
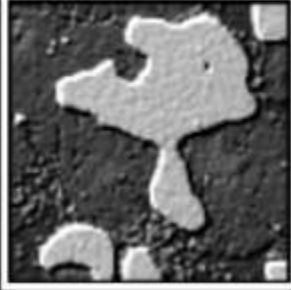
Edge Extraction and Edge Highlighting

The gradient filter has two effects, depending on whether the central coefficient x is equal to 1 or 0.

- If the central coefficient is null ($x = 0$), the gradient filter highlights the pixels where variations of light intensity occur along a direction specified by the configuration of the coefficients a , b , c , and d . The transformed image contains black-white borders at the original edges, and the shades of the overall patterns are darkened.

Source Image	Prewitt #14	Filtered Image
	$\begin{bmatrix} -1 & -1 & 0 \\ -1 & \mathbf{0} & 1 \\ 0 & 1 & 1 \end{bmatrix}$	

If the central coefficient is equal to 1 ($x = 1$), the gradient filter detects the same variations as mentioned above, but superimposes them over the source image. The transformed image looks like the source image with edges highlighted. Use this type of kernel for grain extraction and perception of texture.

Source Image	Prewitt #15	Filtered Image
	$\begin{bmatrix} -1 & -1 & 0 \\ -1 & \mathbf{1} & 1 \\ 0 & 1 & 1 \end{bmatrix}$	

Notice that Prewitt #15 can be decomposed as follows:

$$\begin{bmatrix} -1 & -1 & 0 \\ -1 & \mathbf{0} & 1 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & -1 & 0 \\ -1 & \mathbf{1} & 1 \\ 0 & 1 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 \end{bmatrix}$$



Note The convolution filter using the second kernel on the right side of the

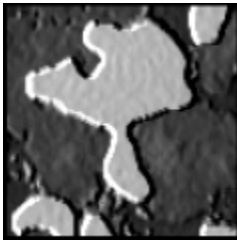
equation reproduces the source image. All neighboring pixels are multiplied by 0 and the central pixel remains equal to itself: $(P_{(i,j)} = 1 \times P_{(i,j)})$.

This equation indicates that Prewitt #15 adds the edges extracted by the Kernel C to the source image.

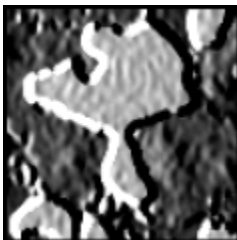
Prewitt #15 = Prewitt #14 + Source Image

Edge Thickness

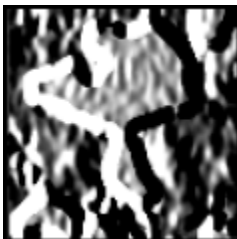
The larger the kernel, the thicker the edges. The following image illustrates gradient west-east 3×3 .



The following image illustrates gradient west-east 5×5 .



Finally, the following image illustrates gradient west-east 7×7 .

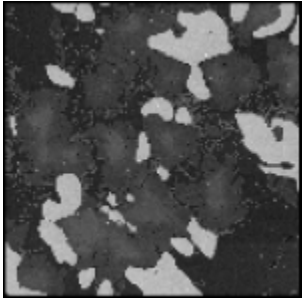


Laplacian Filters

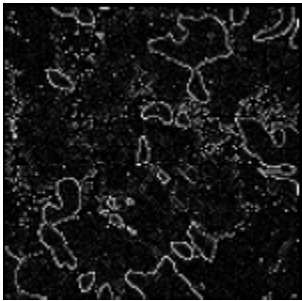
A Laplacian filter highlights the variation of the light intensity surrounding a pixel. The

filter extracts the contour of objects and outlines details. Unlike the gradient filter, it is omnidirectional.

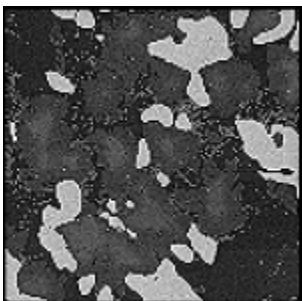
Given the following source image:



A Laplacian filter extracts contours to produce the following image.



A Laplacian filter highlights contours to produce the following image.



Kernel Definition

The Laplacian convolution filter is a second-order derivative, and its kernel uses the following model:

$$\begin{array}{ccc} a & d & c \\ b & x & b \\ c & d & a \end{array}$$

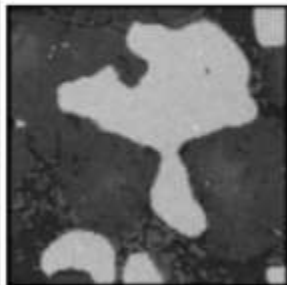

where a , b , c , and d are integers.

The Laplacian filter has two different effects, depending on whether the central coefficient x is equal to or greater than the sum of the absolute values of the outer coefficients.

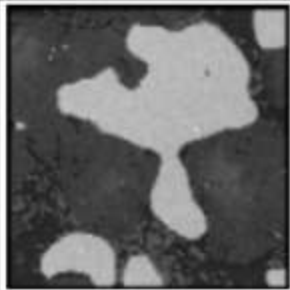

Contour Extraction and Highlighting

If the central coefficient is equal to this sum $x = 2(|a| + |b| + |c| + |d|)$, the Laplacian filter extracts the pixels where significant variations of light intensity are found. The presence of sharp edges, boundaries between objects, modification in the texture of a background, noise, or other effects can cause these variations. The transformed image contains white contours on a black background.

Notice the following source image, Laplacian kernel, and filtered image.

Source Image	Laplacian #3	Filtered Image
	$\begin{array}{ccc} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{array}$	

If the central coefficient is greater than the sum of the outer coefficients ($x > 2(a + b + c + d)$), the Laplacian filter detects the same variations as mentioned above, but superimposes them over the source image. The transformed image looks like the source image, with all significant variations of the light intensity highlighted.

Source Image	Laplacian #4	Filtered Image
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & \mathbf{9} & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

Notice that the Laplacian #4 kernel can be decomposed as follows:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & \mathbf{9} & -1 \\ -1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & \mathbf{8} & -1 \\ -1 & -1 & -1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 \end{bmatrix}$$



Note The convolution filter, using the second kernel on the right side of the equation, reproduces the source image. All neighboring pixels are multiplied by 0, and the central pixel remains equal to itself: $(P_{(i,j)} = 1 \times P_{(i,j)})$.

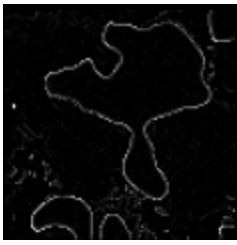
This equation indicates that the Laplacian #2 kernel adds the contours extracted by the Laplacian #1 kernel to the source image.

Laplacian #4 = Laplacian #3 + Source Image.

For example, if the central coefficient of Laplacian #4 kernel is 10, the Laplacian filter adds the contours extracted by Laplacian #3 kernel to the source image times 2, and so forth. A greater central coefficient corresponds to less-prominent contours and details highlighted by the filter.

Contour Thickness

Larger kernels correspond to thicker contours. The following image is a Laplacian 3×3 .



The following image is a Laplacian 5×5 .



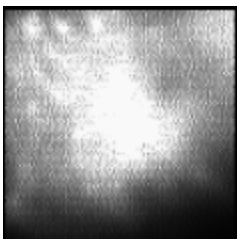
The following image is a Laplacian 7×7 .



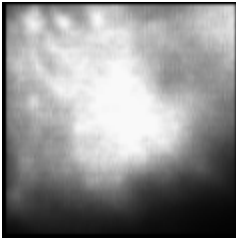
Smoothing Filter

A smoothing filter attenuates the variations of light intensity in the neighborhood of a pixel. It smooths the overall shape of objects, blurs edges, and removes details.

Given the following source image,



a smoothing filter produces the following image.



Kernel Definition

A smoothing convolution filter is an averaging filter whose kernel uses the following model:

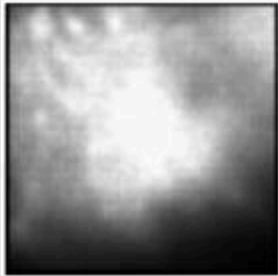
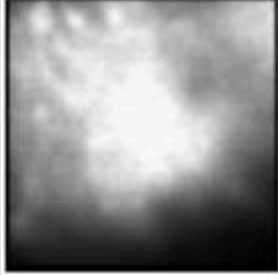
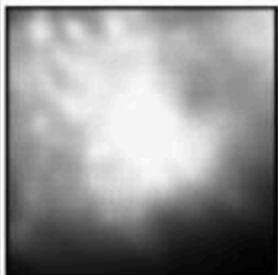
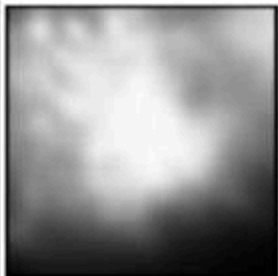
$$\begin{array}{ccc} a & d & c \\ b & x & b \\ c & d & a \end{array}$$

where a , b , c , and d are positive integers, and $x = 0$ or 1 .

Because all the coefficients in a smoothing kernel are positive, each central pixel becomes a weighted average of its neighbors. The stronger the weight of a neighboring pixel, the more influence it has on the new value of the central pixel.

For a given set of coefficients (a , b , c , d), a smoothing kernel with a central coefficient equal to 0 ($x = 0$) has a stronger blurring effect than a smoothing kernel with a central coefficient equal to 1 ($x = 1$).

Notice the following smoothing kernels and filtered images. A larger kernel size corresponds to a stronger smoothing effect.

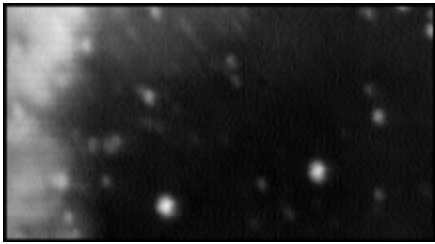
Kernel	Filtered Image
Kernel A 0 1 0 1 0 1 0 1 0	
Kernel B 2 2 2 2 1 2 2 2 2	
Kernel C 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	
Kernel D 1 1 1	

Gaussian Filters

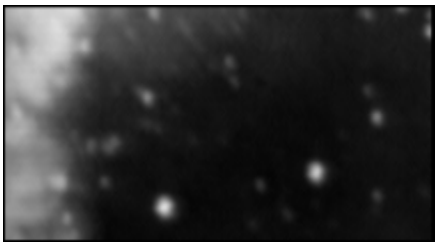
A Gaussian filter attenuates the variations of light intensity in the neighborhood of a pixel. It smooths the overall shape of objects and attenuates details. It is similar to a

smoothing filter, but its blurring effect is more subdued.

Given the following source image,



a Gaussian filter produces the following image.



Kernel Definition

A Gaussian convolution filter is an averaging filter, and its kernel uses the model

$$\begin{array}{ccc} a & d & c \\ b & x & b \\ c & d & a \end{array}$$

where, a , b , c , and d are positive integers, and $x > 1$.

3×3			5×5				
1	2	1	1	2	4	2	1
2	4	2	2	4	8	4	2
1	2	1	4	8	16	8	4
			2	4	8	4	2
			1	2	4	2	1

Because all the coefficients in a Gaussian kernel are positive, each pixel becomes a weighted average of its neighbors. The stronger the weight of a neighboring pixel, the more influence it has on the new value of the central pixel.

Unlike a smoothing kernel, the central coefficient of a Gaussian filter is greater than 1. Therefore the original value of a pixel is multiplied by a weight greater than the weight of any of its neighbors. As a result, a greater central coefficient corresponds to a more subtle smoothing effect. A larger kernel size corresponds to a stronger smoothing effect.

Nonlinear Filters

A nonlinear filter replaces each pixel value with a nonlinear function of its surrounding pixels. Like the linear filters, the nonlinear filters operate on a neighborhood.

Nonlinear Prewitt Filter

The nonlinear Prewitt filter is a highpass filter that extracts the outer contours of objects. It highlights significant variations of the light intensity along the vertical and horizontal axes.

Each pixel is assigned the maximum value of its horizontal and vertical gradient obtained with the following Prewitt convolution kernels:

Prewitt #0	Prewitt #12
-1 0 1	-1 -1 -1
-1 0 1	0 0 0
-1 0 1	-1 1 1

Nonlinear Sobel Filter

The nonlinear Sobel filter is a highpass filter that extracts the outer contours of objects. It highlights significant variations of the light intensity along the vertical and horizontal axes.

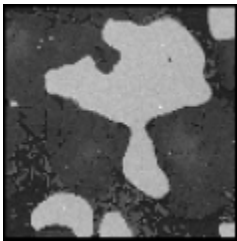
Each pixel is assigned the maximum value of its horizontal and vertical gradient obtained with the following Sobel convolution kernels:

Sobel #16			Sobel #28		
-1	0	1	-1	-2	-1
-2	0	2	0	0	0
-1	0	1	-1	2	1

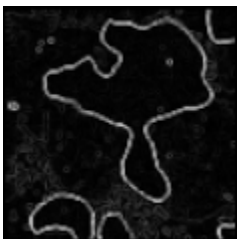
As opposed to the Prewitt filter, the Sobel filter assigns a higher weight to the horizontal and vertical neighbors of the central pixel.

Nonlinear Prewitt and Nonlinear Sobel Example

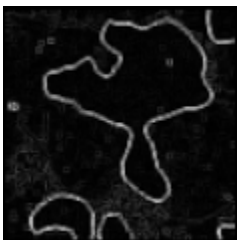
This example uses the following source image.



A nonlinear Prewitt filter produces the following image.



A nonlinear Sobel filter produces the following image.



Both filters outline the contours of the objects. Because of the different convolution kernels they combine, the nonlinear Prewitt has the tendency to outline curved contours while the nonlinear Sobel extracts square contours. This difference is

noticeable when observing the outlines of isolated pixels.

Nonlinear Gradient Filter

The nonlinear gradient filter outlines contours where an intensity variation occurs along the vertical axis.

Roberts Filter

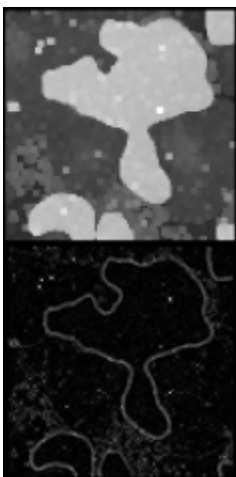
The Roberts filter outlines the contours that highlight pixels where an intensity variation occurs along the diagonal axes.

Differentiation Filter

The differentiation filter produces continuous contours by highlighting each pixel where an intensity variation occurs between itself and its three upper-left neighbors.

Sigma Filter

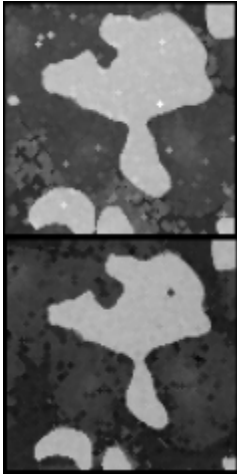
The Sigma filter is a highpass filter. It outlines contours and details by setting pixels to the mean value found in their neighborhood, if their deviation from this value is not significant. The example on the left shows an image before filtering. The example on the right shows the image after filtering.



Lowpass Filter

The lowpass filter reduces details and blurs edges by setting pixels to the mean value found in their neighborhood, if their deviation from this value is large. The example on

the left shows an image before filtering. The example on the right shows the image after filtering.



Median Filter

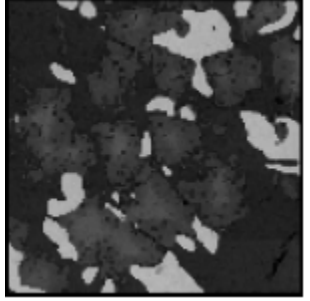
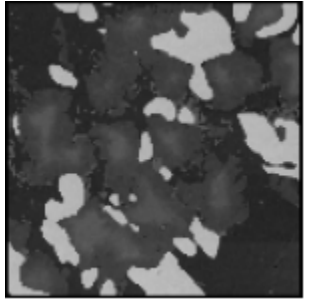
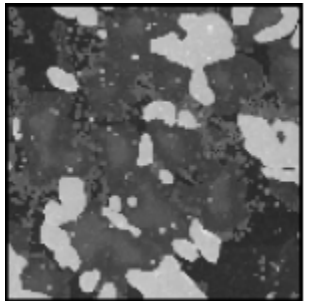
The median filter is a lowpass filter. It assigns to each pixel the median value of its neighborhood, effectively removing isolated pixels and reducing detail. However, the median filter does not blur the contour of objects.

You can implement the median filter by performing an Nth order filter and setting the order to $(f^2 - 1)/2$ for a given filter size of $f \times f$.

Nth Order Filter

The Nth order filter is an extension of the median filter. It assigns to each pixel the Nth value of its neighborhood when they are sorted in increasing order. The value N specifies the order of the filter, which you can use to moderate the effect of the filter on the overall light intensity of the image. A lower order corresponds to a darker transformed image; a higher order corresponds to a brighter transformed image.

To see the effect of the Nth order filter, notice the example of an image with bright objects and a dark background. When viewing this image with the Gray palette, the objects have higher gray-level values than the background.

For a Given Filter Size $f \times f$	Example of a Filter Size 3×3	
<ul style="list-style-type: none"> If $N < (f^2 - 1)/2$, the Nth order filter has the tendency to erode bright regions (or dilate dark regions). If $N = 0$, each pixel is replaced by its local minimum. 	Order 0 (smooths image, erodes bright objects)	
<ul style="list-style-type: none"> If $N = (f^2 - 1)/2$, each pixel is replaced by its local median value. Dark pixels isolated in objects are removed, as well as bright pixels isolated in the background. The overall area of the background and object regions does not change. 	Order 4 (equivalent to a median filter)	
<ul style="list-style-type: none"> If $N > (f^2 - 1)/2$, the Nth order filter has the tendency to dilate bright regions and erode dark regions. If $N = f^2 - 1$, each pixel is replaced by its local maximum. 	Order 8 (smooths image, dilates bright objects)	

In-Depth Discussion

If $P_{(i,j)}$ represents the intensity of the pixel P with the coordinates (i, j) , the pixels surrounding $P_{(i,j)}$ can be indexed as follows (in the case of a 3×3 matrix):

$P_{(i-1, j-1)}$	$P_{(i, j-1)}$	$P_{(i+1, j-1)}$
$P_{(i-1, j)}$	$P_{(i, j)}$	$P_{(i+1, j)}$

$P(i-1, j+1)$	$P(i, j+1)$	$P(i+1, j+1)$
---------------	-------------	---------------

A linear filter assigns to $P(i, j)$ a value that is a linear combination of its surrounding values.

For example:

$$P(i, j) = P(i, j-1) + P(i-1, j) + 2P(i, j) + P(i+1, j) + P(i, j+1)$$

A nonlinear filter assigns to $P(i, j)$ a value that is not a linear combination of the surrounding values.

For example:

$$P(i, j) = \max(P(i-1, j-1), P(i+1, j-1), P(i-1, j+1), P(i+1, j+1))$$

In the case of a 5×5 neighborhood, the i and j indexes vary from -2 to 2 . The series of pixels that includes $P(i, j)$ and its surrounding pixels is annotated as $P(n, m)$.

Linear Filters

For each pixel $P(i, j)$ in an image where i and j represent the coordinates of the pixel, the convolution kernel is centered on $P(i, j)$. Each pixel masked by the kernel is multiplied by the coefficient placed on top of it. $P(i, j)$ becomes either the sum of these products divided by the sum of the coefficient or 1, depending on which is greater.

In the case of a 3×3 neighborhood, the pixels surrounding $P(i, j)$ and the coefficients of the kernel, K , can be indexed as follows:

$P(i-1, j-1)$	$P(i, j-1)$	$P(i+1, j-1)$
$P(i-1, j)$	$P(i, j)$	$P(i+1, j)$
$P(i-1, j+1)$	$P(i, j+1)$	$P(i+1, j+1)$

$K(i-1, j-1)$	$K(i, j-1)$	$K(i+1, j-1)$
$K(i-1, j)$	$K(i, j)$	$K(i+1, j)$
$K(i-1, j+1)$	$K(i, j+1)$	$K(i+1, j+1)$

The pixel $P(i, j)$ is given the value $(1 / N) \sum K(a, b) P(a, b)$, with a ranging from $(i - 1)$ to $(i + 1)$, and b ranging from $(j - 1)$ to $(j + 1)$. N is the normalization factor, equal to $\sum K(a, b)$ or 1, whichever is greater.

If the new value $P(i, j)$ is negative, it is set to 0. If the new value $P(i, j)$ is greater than 255, it is set to 255 (in the case of 8-bit resolution).

The greater the absolute value of a coefficient $K(a, b)$, the more the pixel $P(a, b)$ contributes to the new value of $P(i, j)$. If a coefficient v is 0, the neighbor $P(a, b)$ does not contribute to the new value of $P(i, j)$ (notice that $P(a, b)$ might be $P(i, j)$ itself).

If the convolution kernel is:

$$\begin{array}{ccc} 0 & 0 & 0 \\ -2 & \mathbf{1} & 2 \\ 0 & 0 & 0 \end{array}$$

$$\text{then } P(i, j) = (-2P(i-1, j) + P(i, j) + 2P(i+1, j))$$

If the convolution kernel is:

$$\begin{array}{ccc} 0 & 1 & 0 \\ 1 & \mathbf{0} & 1 \\ 0 & 1 & 0 \end{array}$$

$$\text{then } P(i, j) = (P(i, j-1) + P(i-1, j) + P(i+1, j) + P(i, j+1))$$

Nonlinear Prewitt Filter

$$P_{(i,j)} = \max \left(\left(P_{(i+1,j-1)} - P_{(i-1,j-1)} + P_{(i+1,j)} - P_{(i-1,j)} + P_{(i+1,j+1)} - P_{(i-1,j+1)} \right), \left(P_{(i-1,j+1)} - P_{(i-1,j-1)} + P_{(i,j+1)} - P_{(i,j-1)} + P_{(i+1,j+1)} \right) - P_{(i+1,j-1)} \right)$$

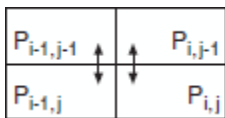
Nonlinear Sobel Filter

$$P_{(i,j)} = \max \left(\left(P_{(i+1,j-1)} - P_{(i-1,j-1)} + 2P_{(i+1,j)} - 2P_{(i-1,j)} + P_{(i+1,j+1)} - P_{(i-1,j+1)} \right), \left(P_{(i-1,j+1)} - P_{(i-1,j-1)} + P_{(i,j+1)} - P_{(i,j-1)} + P_{(i+1,j+1)} \right) - P_{(i+1,j-1)} \right)$$

Nonlinear Gradient Filter

The new value of a pixel becomes the maximum absolute value between its deviation from the upper neighbor and the deviation of its two left neighbors.

$$P(i,j) = \max[|P(i,j-1) - P(i,j)|, |P(i-1,j-1) - P(i-1,j)|]$$



Roberts Filter

The new value of a pixel becomes the maximum absolute value between the deviation of its upper-left neighbor and the deviation of its two other neighbors.

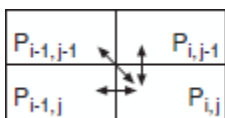
$$P(i,j) = \max[|P(i-1,j-1) - P(i,j)|, |P(i,j-1) - P(i-1,j)|]$$



Differentiation Filter

The new value of a pixel becomes the absolute value of its maximum deviation from its upper-left neighbors.

$$P(i,j) = \max[|P(i-1,j) - P(i,j)|, |P(i-1,j-1) - P(i,j)|, |P(i,j-1) - P(i-1,j)|]$$



Sigma Filter

If $P_{(i,j)} - M > S$
 Then $P_{(i,j)} = P_{(i,j)} - M$
 Else $P_{(i,j)} = M$

Given M , the mean value of $P_{(i,j)}$ and its neighbors, and S , their standard deviation, each pixel $P_{(i,j)}$ is set to the mean value M if it falls inside the range $[M - S, M + S]$.

Lowpass Filter

If $P_{(i,j)} - M < S$
 Then $P_{(i,j)} = P_{(i,j)}$
 Else $P_{(i,j)} = M$

Given M , the mean value of $P_{(i,j)}$ and its neighbors, and S , their standard deviation, each pixel $P_{(i,j)}$ is set to the mean value M if it falls outside the range $[M - S, M + S]$.

Median Filter

$P_{(i,j)} = \text{median value of the series } [P_{(n,m)}]$

Nth Order Filter

$P_{(i,j)} = \text{Nth value in the series } [P_{(n,m)}]$

where the $P_{(n,m)}$ are sorted in increasing order.

The following example uses a 3×3 neighborhood.

13	10	9
12	4	8
5	5	6

The following table shows the new output value of the central pixel for each Nth order value.

Nth Order	0	1	2	3	4	5	6	7	8
New Pixel Value	4	5	5	6	8	9	10	12	13

Notice that for a given filter size f , the Nth order can rank from 0 to $f^2 - 1$. For example, in the case of a filter size 3, the Nth order ranges from 0 to 8 ($3^2 - 1$).

Grayscale Morphology

Morphological transformations extract and alter the structure of particles in an image. They fall into two categories:

- **Binary morphology** functions, which apply to binary images.
- **Grayscale morphology** functions, which apply to gray-level images.

In grayscale morphology, a pixel is compared to those pixels surrounding it in order to keep the pixels whose values are the smallest (in the case of an erosion) or the largest (in the case of a dilation).

When to Use

Use grayscale morphology functions to filter or smooth the pixel intensities of an image. Applications include noise filtering, uneven background correction, and gray-level feature extraction.

Concepts

The **gray-level morphology** functions apply to gray-level images. You can use these functions to alter the shape of regions by expanding bright areas at the expense of dark areas and vice versa. These functions smooth gradually varying patterns and increase the contrast in boundary areas. This section describes the following gray-level morphology functions:

- erosion,
- dilation,
- opening,
- closing,

- proper-opening,
- proper-closing,
- auto-median.

These functions are derived from the combination of gray-level erosions and dilations that use a structuring element.

Erosion Function

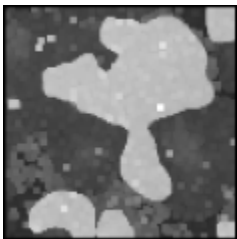
A gray-level erosion reduces the brightness of pixels that are surrounded by neighbors with a lower intensity. The neighborhood is defined by a structuring element.

Dilation Function

A gray-level dilation increases the brightness of each pixel that is surrounded by neighbors with a higher intensity. The neighborhood is defined by a structuring element. The gray-level dilation has the opposite effect of the gray-level erosion because dilating bright regions also erodes dark regions.

Erosion and Dilation Examples

This example uses the following source image.



The following table provides example structuring elements and the corresponding eroded and dilated images

Structuring Element	Erosion	Dilation
<pre> 1 1 1 1 1 1 1 1 1 </pre>		
<pre> 0 1 0 1 1 1 0 1 0 </pre>		

Opening Function

The gray-level opening function consists of a gray-level erosion followed by a gray-level dilation. It removes bright spots isolated in dark regions and smooths boundaries. The effects of the function are moderated by the configuration of the structuring element.

$$\text{opening}(I) = \text{dilation}(\text{erosion}(I))$$

This operation does not significantly alter the area and shape of particles because erosion and dilation are morphological opposites. Bright borders reduced by the erosion are restored by the dilation. However, small bright particles that vanish during the erosion do not reappear after the dilation.

Closing Function

The gray-level closing function consists of a gray-level dilation followed by a gray-level erosion. It removes dark spots isolated in bright regions and smooths boundaries. The effects of the function are moderated by the configuration of the structuring element.

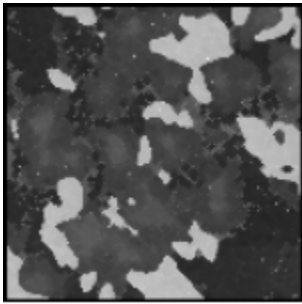
$$\text{closing}(I) = \text{erosion}(\text{dilation}(I))$$

This operation does not significantly alter the area and shape of particles because

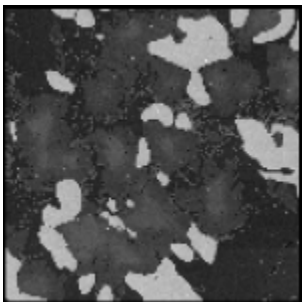
dilation and erosion are morphological opposites. Bright borders expanded by the dilation are reduced by the erosion. However, small dark particles that vanish during the dilation do not reappear after the erosion.

Opening and Closing Examples

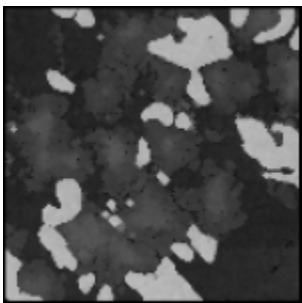
This example uses the following source image.



The opening function produces the following image.



A closing function produces the following image.



Note Consecutive applications of an opening or closing function always give the same results.

Proper-Opening Function

The gray-level proper-opening function is a finite and dual combination of openings and closings. It removes bright pixels isolated in dark regions and smooths the boundaries of bright regions. The effects of the function are moderated by the configuration of the structuring element.

Proper-Closing Function

The proper-closing function is a finite and dual combination of closings and openings. It removes dark pixels isolated in bright regions and smooths the boundaries of dark regions. The effects of the function are moderated by the configuration of the structuring element.

Auto-Median Function

The auto-median function uses dual combinations of openings and closings. It generates simpler particles that have fewer details.

Erosion Concept and Mathematics

Each pixel in an image becomes equal to the minimum value of its neighbors.

For a given pixel P_0 , the structuring element is centered on P_0 .

The pixels masked by a coefficient of the structuring element equal to 1 are then referred as P_i .

$$P_0 = \min(P_i)$$

A gray-level erosion using a structuring element $f \times f$ with all its coefficients set to 1 is equivalent to an Nth order filter with a filter size $f \times f$ and the value N equal to 0. Refer to the ***nonlinear filters*** section for more information.

Dilation Concept and Mathematics

Each pixel in an image becomes equal to the maximum value of its neighbors.

For a given pixel P_0 , the structuring element is centered on P_0 .

The pixels masked by a coefficient of the structuring element equal to 1 are then referred as P_i .

$$P_0 = \max(P_i)$$



Note A gray-level dilation using a structuring element $f \times f$ with all its coefficients set to 1 is equivalent to an Nth order filter with a filter size $f \times f$ and the value N equal to $f^2 - 1$. Refer to the **nonlinear filters** section for more information.

Proper-Opening Concept and Mathematics

If I is the source image, the proper-opening function extracts the minimum value of each pixel between the source image I and its transformed image obtained after an opening, followed by a closing, and followed by another opening.

$$\text{proper-opening}(I) = \min(I, \text{OCO}(I))$$

or

$$\text{proper-opening}(I) = \min(\text{DEEDDE}(I))$$

where:

- I is the source image,
- E is an erosion,
- D is a dilation,
- O is an opening,
- C is a closing,
- $F(I)$ is the image obtained after applying the function F to the image I ,
- $GF(I)$ is the image obtained after applying the function F to the image I followed by the function G to the image I .

Proper-Closing Concept and Mathematics

If I is the source image, the proper-closing function extracts the maximum value of each pixel between the source image I and its transformed image obtained after a closing, followed by an opening, and followed by another closing.

$$\text{proper-closing}(I) = \max(I, \text{OCO}(I))$$

or

$$\text{proper-closing}(I) = \max(EDDEED(I))$$

where:

- I is the source image,
- E is an erosion,
- D is a dilation,
- O is an opening,
- C is a closing,
- $F(I)$ is the image obtained after applying the function F to the image I ,
- $GF(I)$ is the image obtained after applying the function F to the image I followed by the function G to the image I .

Auto-Median Concept and Mathematics

If I is the source image, the auto-median function extracts the minimum value of each pixel between the two images obtained by applying a proper-opening and a proper-closing of the source image I .

$$\text{auto-median}(I) = \min(\text{OCO}(I), \text{COC}(I))$$

or

$$\text{auto-median}(I) = \min(DEEDDE(I), EDDEED(I))$$

where:

- I is the source image,
- E is an erosion,
- D is a dilation,
- O is an opening,
- C is a closing,
- $F(I)$ is the image obtained after applying the function F to the image I ,
- $GF(I)$ is the image obtained after applying the function F to the image I followed by the function G to the image I .

Related concepts:

- [Spatial Filtering](#)

Operators

This section contains information about arithmetic and logic operators, which mask, combine, and compare images.

Introduction

Operators perform basic arithmetic and logical operations on images. Use operators to add, subtract, multiply, and divide an image with other images or constants. You also can perform logical operations, such as AND/NAND, OR/NOR, and XOR/XNOR, and make pixel comparisons between an image and other images or a constant.

When to Use

Common applications of these operators include time-delayed comparisons, identification of the union or intersection between images, correction of image backgrounds to eliminate light drifts, and comparisons between several images and a model. You also can use operators to threshold or mask images and to alter contrast and brightness.

Concepts

An arithmetic or logical operation between images is a pixel-by-pixel transformation. It

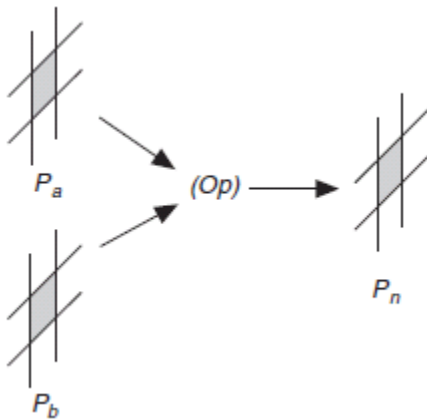
produces an image in which each pixel derives its value from the values of pixels with the same coordinates in other images.

If A is an image with a resolution XY, B is an image with a resolution XY, and Op is the operator, then the image N resulting from the combination of A and B through the operator Op is such that each pixel p of the resulting image N is assigned the value:

$$p_n = (p_a)(Op)(p_b)$$

where

- p_a is the value of pixel p in image a,
- p_b is the value of pixel p in image b.



Arithmetic Operators

The equations in the following table describe the usage of arithmetic operators with two images a and b.

Operator	Equation
Multiply	$p_n = p_a \times p_b$
Divide	$p_n = p_a / p_b$ The divide operator section contains additional information about the divide operation, including division by zero.
Add	$p_n = p_a + p_b$

Operator	Equation
Subtract	$p_n = p_a - p_b$
Modulo	$p_n = p_a \bmod p_b$
Absolute Difference	$p_n = p_a - p_b $

If the resulting pixel value p_n is lower than the minimum possible value for the given image type, the pixel is set to the lowest possible value. If the resulting pixel value is greater than the maximum possible value for the given image type, the pixel is set to the maximum possible value. The following table lists the range of possible values for each supported image type.

Image Type	Range
8-bit Unsigned Grayscale	$0 \leq p_n \leq 255$
16-bit Signed Grayscale	$-32,768 \leq p_n \leq 32,767$
32-bit Floating-Point Grayscale	$-\infty \leq p_n \leq \infty$
32-bit RGB Color	$0 \leq p_n \leq 255$, for each channel (red, green, blue) in the image

Divide Operator

Use of the divide operator can produce results that do not directly translate to appropriate pixel values for an image. In such cases, Vision uses the methods discussed in the following sections to resolve the result of a divide operation to a valid pixel value for the image.

Rounding Results

Dividing two pixel values sometimes produces a non-integer result. Since most common image types accept only integers for the value of a pixel, Vision applies the round-to-even rounding method to the result to produce an integer result to the operation. For the most part, the round-to-even method works like other traditional rounding methods.

If the digit after the last digit you want to keep is greater than five, or a five followed by

one or more non-zero numbers, the last digit is rounded up to the next number. If the following digit is less than five, the last digit is rounded down.

The difference between the round-to-even method and other rounding methods occurs when the digit after the last digit you want to keep is exactly equal to five. In NI Vision, when the following digit is equal to five, the result is rounded to the nearest even number. If the last digit is an odd number, the result is rounded up to the next even number. If the last digit to keep is an even number, the result is truncated at the last digit to keep.

Example

The following examples illustrate using the round-to-even method to round to the nearest integer value:

- 2.7 is rounded to 3 because the next digit is 6 or greater,
- 2.4 is rounded to 2 because the next digit is 4 or less,
- 3.5 is rounded to 4 because the next digit is 5, and the last digit to keep, 3, is odd,
- 2.5 is rounded to 2 because the next digit is 5, and the last digit to keep, 2, is even,
- 2.501 is rounded to 3 because the next digit is 5, but the last digit to keep is followed by one or more non-zero digits.

Division by Zero

The following table describes the effect of division by zero on pixels in an image.

Image Type	Divide by Zero Case	Result
8-bitUnsignedGrayscale	0 / 0	0
8-bitUnsignedGrayscale	$p_a / 0, p_a > 0$	255
16-bitSignedGrayscale	0 / 0	0
16-bitSignedGrayscale	$p_a / 0, p_a > 0$	32,767
16-bitSignedGrayscale	$p_a / 0, p_a < 0$	-32,768
32-bitFloating-PointGrayscale	0 / 0	NaN
32-bitFloating-	$p_a / 0, p_a > 0$	∞

Image Type	Divide by Zero Case	Result
PointGrayscale		
32-bitFloating-PointGrayscale	$p_a / 0, p_a < 0$	$-\infty$
32-bitRGBColor	$p_a(r): 0 / 0 \ p_a(g): 0 / 0 \ p_a(b): 0 / 0$	0
32-bitRGBColor	$p_a(r): p_a(r) / 0, p_a(r) > 0 \ p_a(g): p_a(g) / 0, p_a(g) > 0 \ p_a(b): p_a(b) / 0, p_a(b) > 0$	255

Logic and Comparison Operators

Logic operators are bitwise operators. They manipulate gray-level values coded on one byte at the bit level. The equations in the following table describe the usage of logical operators. The truth tables for logic operators are presented in the ***truth tables*** section.

Operator	Equation
Logical Operators	Logical Operators
AND	$p_n = p_a \text{ AND } p_b$
NAND	$p_n = p_a \text{ NAND } p_b$
OR	$p_n = p_a \text{ OR } p_b$
NOR	$p_n = p_a \text{ NOR } p_b$
XOR	$p_n = p_a \text{ XOR } p_b$
Logic Difference	$p_n = p_a \text{ AND } (\text{NOT} p_b)$
Comparison Operators	Comparison Operators
Mask	if $p_b = 0$, then $p_n = 0$, else $p_n = p_a$
Mean	$p_n = \text{mean}(p_a, p_b)$
Max	$p_n = \text{max}(p_a, p_b)$
Min	$p_n = \text{min}(p_a, p_b)$

In the case of images with 8-bit resolution, logic operators are mainly designed to do

the following:

- Combine gray-level images with binary mask images, which are composed of pixels equal to 0 or 255.
- Combine or compare images with binary or labeled contents.

The following table illustrates how logic operators can be used to extract or remove information in an image.

For a given p_a	If $p_b = 255$, then	If $p_b = 0$, then
AND	$p_a \text{ AND } 255 = p_a$	$p_a \text{ AND } 0 = 0$
NAND	$p_a \text{ NAND } 255 = \text{NOT } p_a$	$p_a \text{ NAND } 0 = 255$
OR	$p_a \text{ OR } 255 = 255$	$p_a \text{ OR } 0 = p_a$
NOR	$p_a \text{ NOR } 255 = 0$	$p_a \text{ NOR } 0 = \text{NOT } p_a$
XOR	$p_a \text{ XOR } 255 = \text{NOT } p_a$	$p_a \text{ XOR } 0 = p_a$
Logic Difference	$p_a - \text{NOT } 255 = p_a$	$p_a - \text{NOT } 0 = 0$

Truth Tables

The following truth tables describe the rules used by the logic operators. The top row and left column give the values of input bits. The cells in the table give the output value for a given set of two input bits.

AND		
	<i>b</i>	<i>b</i>
<i>a</i> = 0	0	0
<i>a</i> = 1	0	1

NAND		
	<i>b</i>	<i>b</i>
<i>a</i> = 0	1	1
<i>a</i> = 1	1	0

OR		
	<i>b</i>	<i>b</i>
<i>a</i> = 0	0	1
<i>a</i> = 1	1	1

NOR		
	<i>b</i>	<i>b</i>
<i>a</i> = 0	1	0
<i>a</i> = 1	0	0

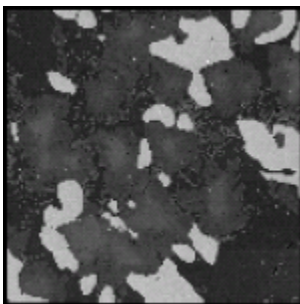
XOR		
	<i>b</i>	<i>b</i>
<i>a</i> = 0	0	1
<i>a</i> = 1	1	0

XNOR		
	<i>b</i> = 0	
<i>a</i> = 0	1	0
<i>a</i> = 1	0	1

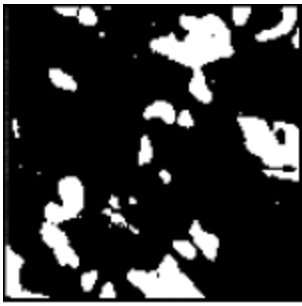
NOT	
	NOT <i>a</i>
<i>a</i> = 0	1
<i>a</i> = 1	0

Example 1

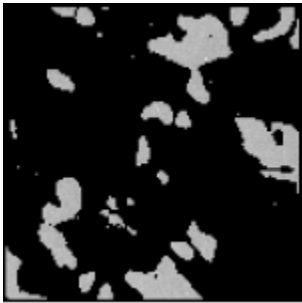
The following figure shows the source grayscale image used in this example.



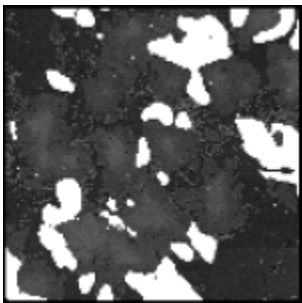
Regions of interest have been isolated in a binary format, retouched with morphological manipulations, and finally multiplied by 255 to obtain the following image mask.



The source image AND mask image operation restores the original intensity of the object regions in the mask.



The source image OR mask image operation restores the original intensity of the background region in the mask.



Example 2

This example demonstrates the use of the OR operation to produce an image containing the union of two binary images. The following image represents the first image, with a background value of 0 and objects with a gray-level value of 128.



The following figure shows the second image, featuring a background value of 0 and objects with gray-level values of 255.

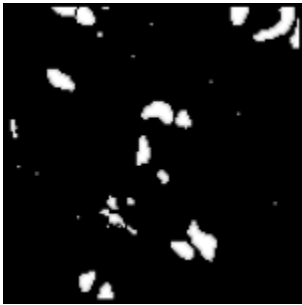
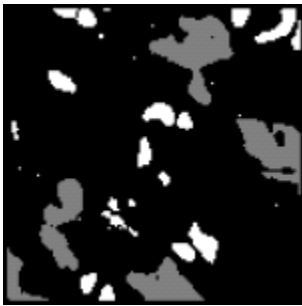


Image #1 OR Image #2 produces a union, as shown in the following image.



Frequency Domain Analysis

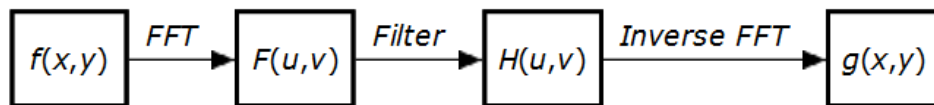
This section contains information about converting images into the frequency domain using the Fast Fourier transform, and information about analyzing and processing images in the frequency domain.

Introduction

Frequency filters alter pixel values with respect to the periodicity and spatial distribution of the variations in light intensity in the image. Unlike spatial filters,

frequency filters do not apply directly to a spatial image, but to its frequency representation. The frequency representation of an image is obtained through the Fast Fourier Transform (FFT) function, which reveals information about the periodicity and dispersion of the patterns found in the source image.

You can filter the spatial frequencies seen in an FFT image. The inverse FFT function then restores a spatial representation of the filtered FFT image.



Frequency processing is another technique for extracting information from an image. Instead of using the location and direction of light-intensity variations, you can use frequency processing to manipulate the frequency of the occurrence of these variations in the spatial domain. This new component is called the spatial frequency, which is the frequency with which the light intensity in an image varies as a function of spatial coordinates.

Spatial frequencies of an image are computed with the FFT. The FFT is calculated in two steps—a 1D Fast Fourier transform of the rows, followed by a 1D Fast Fourier transform of the columns of the previous results. The complex numbers that compose the FFT plane are encoded in a 64-bit floating-point image called a complex image. The complex image is formed by a 32-bit floating point number representing the real part and a 32-bit floating point number representing the imaginary part.

In an image, details and sharp edges are associated with moderate to high spatial frequencies because they introduce significant gray-level variations over short distances. Gradually varying patterns are associated with low spatial frequencies. By filtering spatial frequencies, you can remove, attenuate, or highlight the spatial components to which they relate.

Use a lowpass frequency filter to attenuate or remove, or truncate, high frequencies present in the image. This filter suppresses information related to rapid variations of light intensities in the spatial image. An inverse FFT, used after a lowpass frequency filter, produces an image in which noise, details, texture, and sharp edges are smoothed.

A highpass frequency filter attenuates or removes, or truncates, low frequencies present in the complex image. This filter suppresses information related to slow variations of light intensities in the spatial image. In this case, an inverse FFT used after a highpass frequency filter produces an image in which overall patterns are sharpened and details are emphasized.

A mask frequency filter removes frequencies contained in a mask specified by the user. Using a mask to alter the Fourier transform of an image offers more possibilities than applying a lowpass or highpass filter. The image mask is composed by the user and can describe very specific frequencies and directions in the image. You can apply this technique, for example, to filter dominant frequencies as well as their harmonics in the frequency domain.

When to Use

Because details and sharp edges introduce significant gray-level variations over short distances, they are associated with moderate to high spatial frequencies in an image. Gradually varying patterns are associated with low spatial frequencies.

An image can have extraneous noise introduced during the digitization process, such as periodic stripes. In the frequency domain, the periodic pattern is reduced to a limited set of high spatial frequencies. Truncating these particular frequencies and converting the filtered FFT image back to the spatial domain produces a new image in which the grid pattern has disappeared, while the overall features remain.

Concepts

The FFT of an image is a 2D array of complex numbers, also represented as a complex image. It represents the frequencies of occurrence of light-intensity variations in the spatial domain. The low frequencies correspond to smooth and gradual intensity variations found in the overall patterns of the source image. The high frequencies correspond to abrupt and short intensity variations found at the edges of objects, around noisy pixels, and around details.

FFT Representation

There are two possible representations of the Fast Fourier transform of an image: the

standard representation and the optical representation.

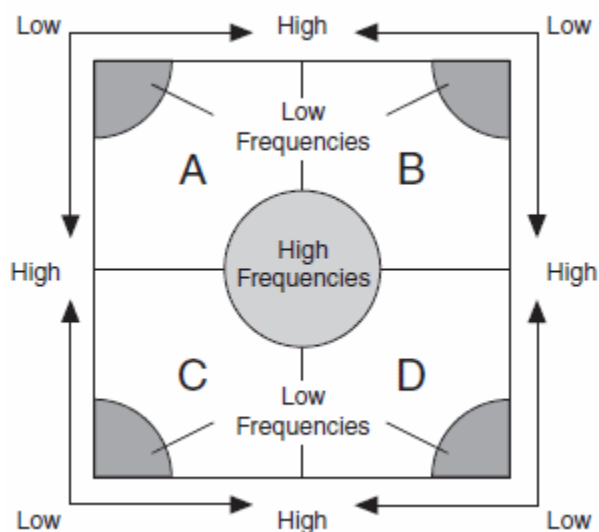
Standard Representation

In the standard representation, high frequencies are grouped at the center of the image while low frequencies are located at the edges. The constant term, or null frequency, is in the upper-left corner of the image.

The frequency range is as follows:

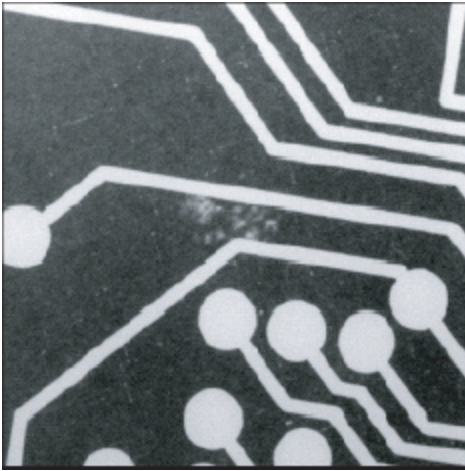
$$\left(-\frac{N}{2}, \frac{N}{2}\right) \times \left(-\frac{M}{2}, \frac{M}{2}\right)$$

where M is the horizontal resolution of the image, and N is the vertical resolution of the image.



Note Vision uses the standard representation to represent complex images in memory. Use this representation when building an image mask.

The illustration shows an original image:



The illustration shows the FFT of the same image using standard representation:

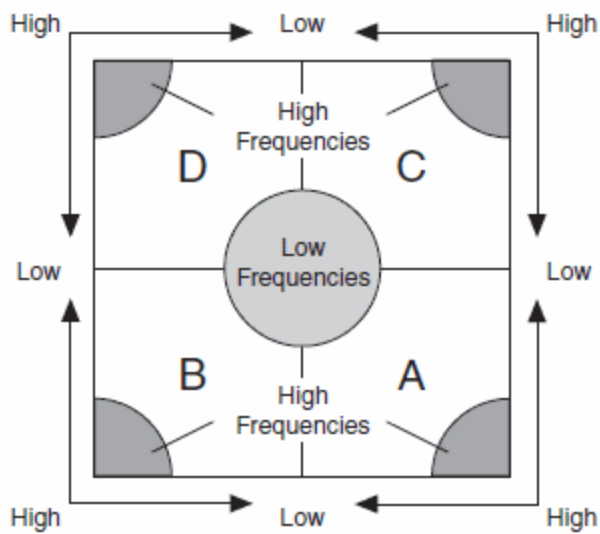


Optical Representation

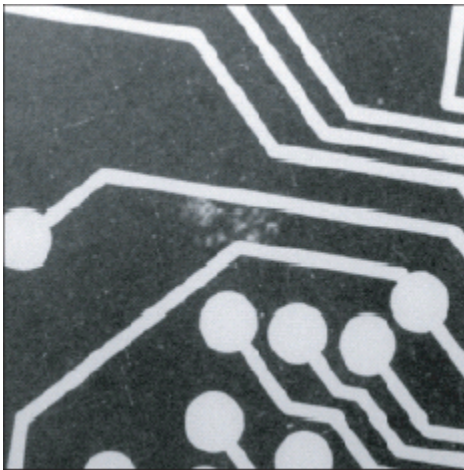
In the optical representation, low frequencies are grouped at the center of the image while high frequencies are located at the edges. The constant term, or null frequency, is at the center of the image.

The frequency range is as follows:

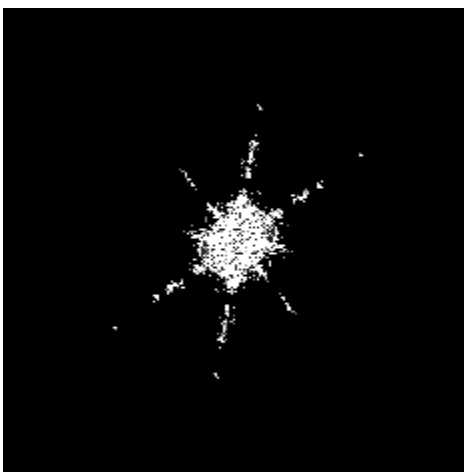
$$\left(-\frac{N}{2}, \frac{N}{2}\right) \times \left(-\frac{M}{2}, \frac{M}{2}\right)$$



The illustration shows an original image:



The illustration shows the FFT of the same image using optical representation:





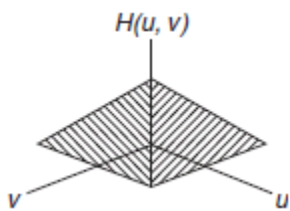
Note Vision uses optical representation when displaying a complex image.

You can switch from standard representation to optical representation by permuting the A, B, C, and D quarters.

Intensities in the FFT image are proportional to the amplitude of the displayed component.

Lowpass FFT Filters

A lowpass frequency filter attenuates, or removes, high frequencies present in the FFT plane. This filter suppresses information related to rapid variations of light intensities in the spatial image. In this case, an inverse FFT produces an image in which noise, details, texture, and sharp edges are smoothed.



A lowpass frequency filter attenuates, or removes, spatial frequencies located outside a frequency range centered on the fundamental (or null) frequency.

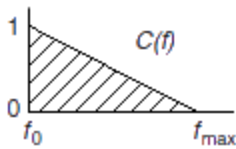
Lowpass Attenuation

Lowpass attenuation applies a linear attenuation to the full frequency range, increasing from the null frequency f_0 to the maximum frequency f_{\max} . This is done by multiplying each frequency by a coefficient C , which is a function of its deviation from the fundamental and maximum frequencies.

$$C_f = \frac{f_{\max} - f}{f_{\max} - f_0}$$

where,

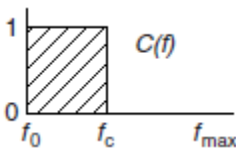
- $C(f_0) = 1$
- $C(f_{\max}) = 0$



Lowpass Truncation

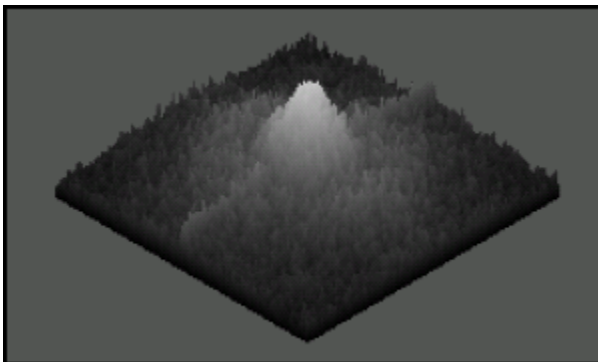
Lowpass truncation removes a frequency f if it is higher than the cutoff or truncation frequency, f_c . This is done by multiplying each frequency f by a coefficient C equal to 0 or 1, depending on whether the frequency f is greater than the truncation frequency f_c .

If $f > f_c$ then $C(f) = 0$ else $C(f) = 1$

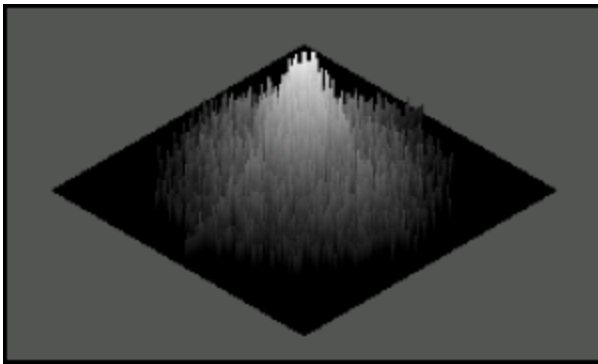


The following series of graphics illustrates the behavior of both types of lowpass filters. They represent the 3D-view profile of the magnitude of the FFT.

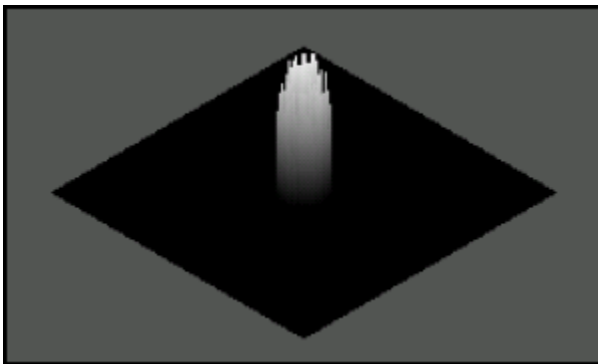
This example uses the following original FFT.



After lowpass attenuation, the magnitude of the central peak is the same, and variations at the edges almost have disappeared.

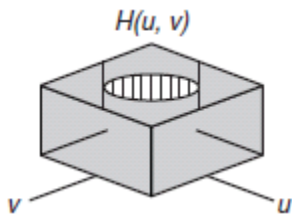


After lowpass truncation with $f_c = f_0 + 20\%(f_{\max} - f_0)$, spatial frequencies outside the truncation range $[f_0, f_c]$ are removed. The part of the central peak that remains is identical to the one in the original FFT plane.



Highpass FFT Filters

A highpass FFT filter attenuates, or removes, low frequencies present in the FFT plane. It has the effect of suppressing information related to slow variations of light intensities in the spatial image. In this case, the Inverse FFT command produces an image in which overall patterns are attenuated and details are emphasized.



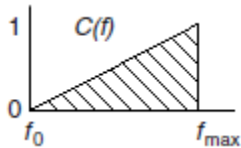
Highpass Attenuation

Highpass attenuation applies a linear attenuation to the full frequency range, increasing from the maximum frequency f_{\max} to the null frequency f_0 . This is done by

multiplying each frequency by a coefficient C , which is a function of its deviation from the fundamental and maximum frequencies.

where,

- $C(f_0) = 0$
- $C(f_{max}) = 1$



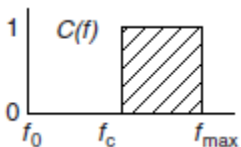
Highpass Truncation

Highpass truncation removes a frequency f if it is lower than the cutoff or truncation frequency, f_c . This is done by multiplying each frequency f by a coefficient C equal to 1 or 0, depending on whether the frequency f is greater than the truncation frequency f_c .

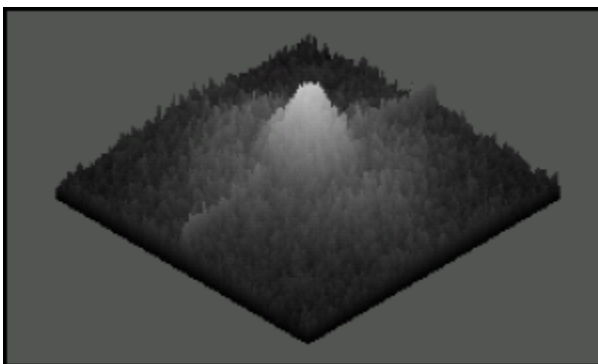
If $f < f_c$

Then $C_f = 0$

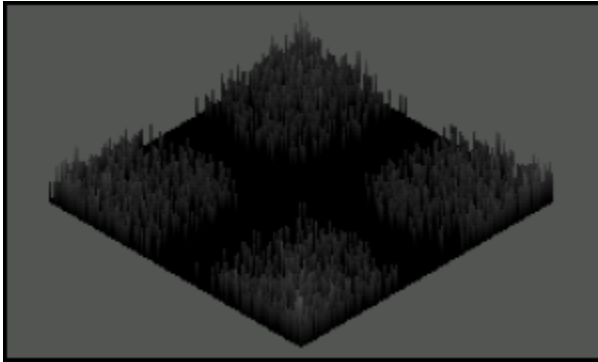
Else $C_f = 1$



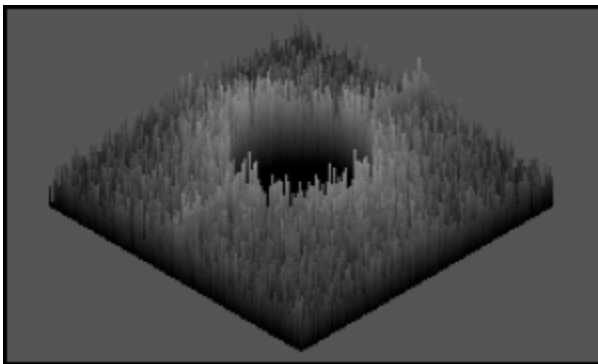
The following series of graphics illustrates the behavior of both types of highpass filters. They represent the 3D-view profile of the magnitude of the FFT. This example uses the following original FFT image.



After highpass attenuation, the central peak has been removed, and variations present at the edges remain.

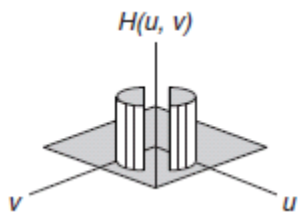


After highpass truncation with $f_c = f_0 + 20\%(f_{\max} - f_0)$, spatial frequencies inside the truncation range $[f_0, f_c]$ are set to 0. The remaining frequencies are identical to the ones in the original FFT plane.



Mask FFT Filters

A mask FFT filter removes frequencies contained in a mask specified by the user. Depending on the mask definition, this filter can act as a lowpass, bandpass, highpass, or any type of selective filter.



In-Depth Discussion

Fourier Transform

The spatial frequencies of an image are calculated by a function called the **Fourier Transform**.

It is defined in the continuous domain as:

$$F(u, v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) e^{-j2\pi(xu - yv)} dx dy$$

where,

- $f(x, y)$ is the light intensity of the point,
- x, y , and $(u + v)$ are the horizontal and vertical spatial frequencies.

The Fourier Transform assigns a complex number to each set (u, v) .

Inversely, a Fast Fourier Transform $F(u, v)$ can be transformed into a spatial image $f(x, y)$ of resolution NM using the following formula:

$$F(x, y) = \frac{1}{NM} \sum_{u=0}^{N-1} \sum_{v=0}^{M-1} F(u, v) e^{j2\pi\left(\frac{ux}{N} + \frac{vy}{M}\right)}$$

where,

- $N \times M$ is the resolution of the spatial image $f(x, y)$.

In the discrete domain, the Fourier Transform is calculated with an efficient algorithm called the Fast Fourier Transform (FFT).

$$F(u, v) = \frac{1}{NM} \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} F(x, y) e^{j2\pi\left(\frac{ux}{N} + \frac{vy}{M}\right)}$$

Because $e^{-j2\pi ux} = \cos 2\pi ux - j\sin 2\pi ux$, $F(u, v)$ is composed of an infinite sum of sine

and cosine terms. Each pair (u, v) determines the frequency of its corresponding sine and cosine pair. For a given set (u, v) , notice that all values $f(x, y)$ contribute to $F(u, v)$. Because of this complexity, the FFT calculation is time consuming.

Given an image with a resolution $N \times M$ and given Δx and Δy the spatial step increments, the FFT of the source image has the same resolution NM and its frequency step increments Δu and Δv , which are defined in the following equations:

$$\Delta u = \frac{1}{N \times \Delta x}$$

$$\Delta v = \frac{1}{M \times \Delta y}$$

FFT Display

An FFT image can be visualized using any of its four complex components: real part, imaginary part, magnitude, and phase. The relation between these components is expressed by:

$$F(u, v) = R(u, v) + jI(u, v)$$

where,

- $R(u)$ is the real part,
- $I(u)$ is the imaginary part, and

$$F(u, v) = |F(u, v)| \times e^{j\phi(u, v)}$$

where,

- $|F(u, v)|$ is the magnitude,
- $\phi(u, v)$ is the phase.

The magnitude of $F(u, v)$ is also called the Fourier spectrum and is equal to:

$$|F(u, v)| = \sqrt{R(u, v)^2 + I(u, v)^2}$$

The Fourier spectrum to the power of two is known as the power spectrum or spectral density.

The phase $\phi(u, v)$ is also called the phase angle and is equal to:

$$\phi(u, v) = \text{atan}\left(\frac{I(u, v)}{R(u, v)}\right)$$

By default, when you display a complex image, the magnitude plane of the complex image is displayed using the optical representation. To visualize the magnitude values properly, the magnitude values are scaled by the factor m before they are displayed. The factor m is calculated as

$$\frac{128}{w \times h}$$

where,

- w is the width of the image,
- h is the height of the image.

Texture Defect Detection

This section contains information about texture defect detection and analysis.

Introduction

Texture defect detection detects defects in a texture based on a texture classifier trained with texture samples that do not contain defects. During inspection, the texture defect detection algorithm identifies as defective any regions that do not match the trained texture samples. The identified defects appear in the output image as blobs. You can use the **particle analysis** tools in the Vision library to analyze the properties of the detected defects. Texture defect detection is not designed for continuous web or surface inspection applications.

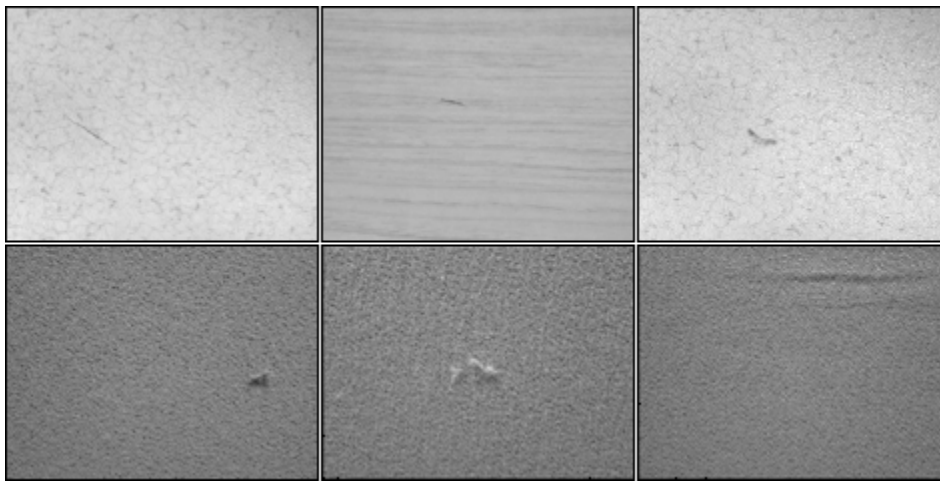
Related concepts:

- [Particle Analysis](#)

When to Use

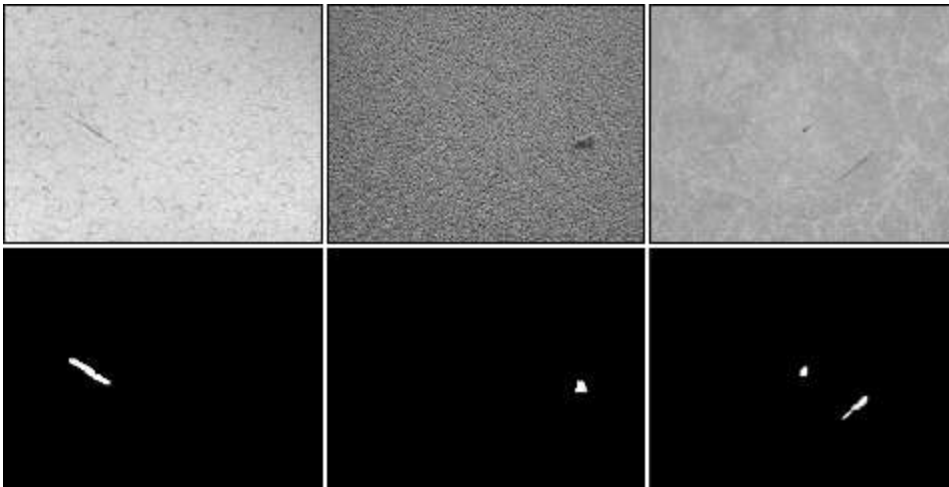
Texture defect detection recognizes scratches, cracks, stains, and other defects that may vary in size and shape on textured surfaces. Use texture defect detection when traditional machine vision techniques such as dynamic thresholding and edge detection are not adequate to find the defects on the parts being inspected. Applications include the automated inspection of materials such as ceramic tiles, textiles, lumber, paper, plastic surfaces and glass, which are often characterized by irregular texture patterns.

The following figure shows examples of texture defects that traditional machine vision techniques may not adequately detect.



What to Expect from Texture Defect Detection

Texture defect detection detects defects in a texture based on a texture classifier trained with texture samples that do not contain defects. The texture classifier is trained to recognize texture samples that are acceptable in the current inspection. The texture defect detection algorithm accepts an image of a texture surface as an input, identifies texture defects, and returns a binary image of the texture defects. The following figure illustrates typical input and output images.

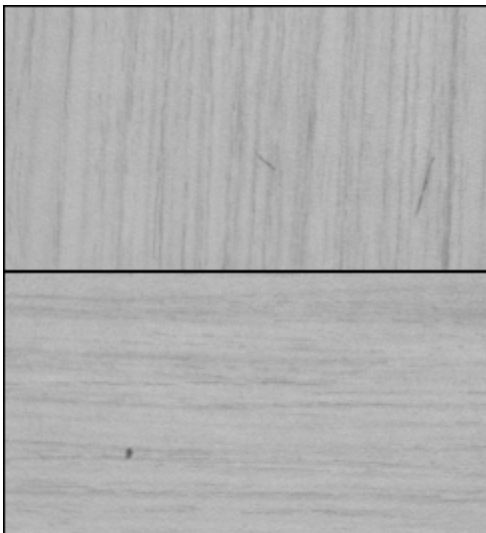


Shift Variation

Texture defect detection is invariant to shift. For example, if the texture in the inspection image shifts vertically or horizontally from the trained texture samples, the texture defect detection algorithm continues to correctly identify any texture defects.

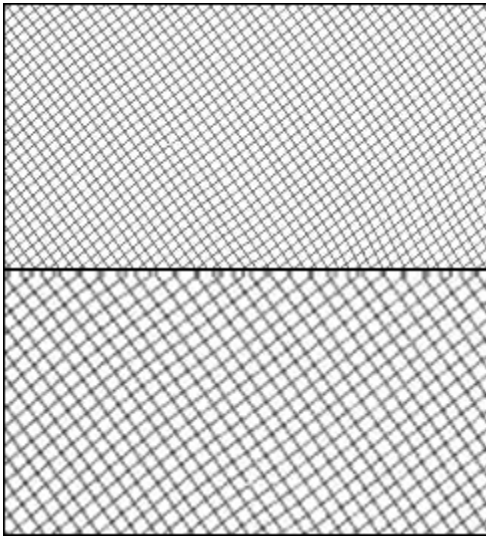
Rotation Variation

Texture defect detection is invariant to rotation of approximately ± 5 degrees. If the texture under inspection can shift more than 5 degrees, you must train the classifier with texture samples at every expected orientation. The following figures illustrate the same texture at distinct orientations that require trained samples for each variation.



Scale Variation

Texture defect detection is invariant to rotation of approximately ± 10 degrees. If the texture under inspection can vary more than 10 degrees in scale, you must train the classifier with texture samples at every expected scale variation. The following figures illustrate a difference in scale that require trained samples for each variation.



In-Depth Discussion

This section provides additional information you may need for building successful texture defect detection applications.

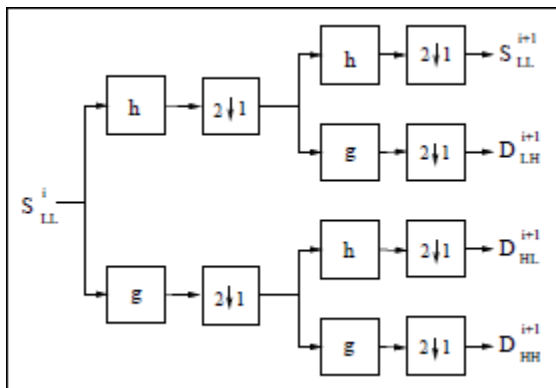
- The texture defect detection algorithm uses discrete wavelet frame decomposition and a statistical approach to characterize visual textures.
- The algorithm decomposes a texture inspection image into several subbands using over-complete and shift-invariant wavelet frames.
- The algorithm partitions each subband image into non-overlapping windows, and uses a gray-level co-occurrence matrix (GLCM) to analyze the coefficient distribution of each window. Second order statistics, or Haralick features, are calculated from the GLCM representations.
- The algorithm concatenates Haralick features extracted from all subbands in a high dimensional feature space.
- A one-class **support vector machine** (SVM) is trained with a general description of the texture under inspection in the same high dimensional feature space. During inspection, Haralick features extracted from the inspection image are classified

using the previously trained texture defect classifier as texture or defective pixels.

Wavelet Frame Decomposition

The first step in texture defect detection is a multi-resolution analysis of the texture in the inspection image, inspired by psycho-visual findings that humans perceive images in a multi-scale manner. The texture defect detection algorithm uses discrete wavelet frame transforms (DWFT) proposed by Unser¹ to obtain translation-invariant characteristics from textures with minimum dependencies between the transform coefficients. Wavelet frame transforms are wavelet transform variations in which the output of the filter banks are not subsampled. As a result, each subband image has the same size as the input image and performs better in texture classification and segmentation.

For texture analysis, discrete wavelet transforms (DWT) for hierarchical signal analysis and decomposition are implemented through an iterative filtering and downsampling operation with lowpass and highpass filters h and g . The following figure illustrates one stage of a 2-dimensional DWT where $2\downarrow 1$ denotes downsampling by a factor of 2.



The filters h and g and their corresponding reconstruction counterparts satisfy the general perfect reconstruction constraint $h(z) \sim h(z - 1) + g(z) \sim g(z - 1) = 1$ in the z -transform domain. At each iteration, the coefficients of the coarse approximation, s_{i+1} , and the detail coefficients, d_{i+1} , are calculated from current coefficients, s_i , by:

1. For more information about discrete wavelet frame transforms see Unser, M. "Texture Classification and Segmentation Using Wavelet Frames," Image Processing, IEEE Transactions on 4, no. 11 (1995) 1549–1560.

$$\begin{cases} s_{i+1}(k) = (h \times s_i(k))_{\downarrow 2} \\ d_{i+1}(k) = (g \times d_i(k))_{\downarrow 2} \quad (i = 0, \dots, l) \end{cases}$$

where,

- $s_0(k) = f(k)$ is the input signal to the filter bank.

This concept is extended to 2-dimensional discrete signals (images) where 2-dimensional filters are obtained by the tensor product of 1-dimensional lowpass and highpass filters h and g along the rows and columns. After one stage of decomposition the image at resolution i is decomposed into four subband images—one coarse approximation s^{i+1}_{LL} , and three detail images d^{i+1}_{LH} , d^{i+1}_{HL} , and sd^{i+1}_{HH} . The three detail subband images are referred to as the horizontal (H), vertical (V), and diagonal (D) details, respectively.

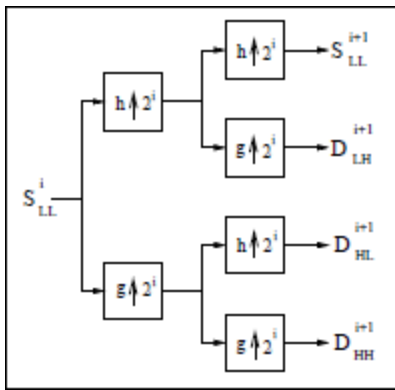
Unser proposes an over-complete DWFT decomposition, showing that it constitutes a tight frame of l^2 . Unser implements the following fast iterative decomposition algorithm.

$$\begin{cases} s_{i+1}(k) = (h)_{\downarrow 2} \times s_i(k) \\ d_{i+1}(k) = (g)_{\downarrow 2} \times d_i(k) \quad i = (0, \dots, l) \end{cases}$$

where,

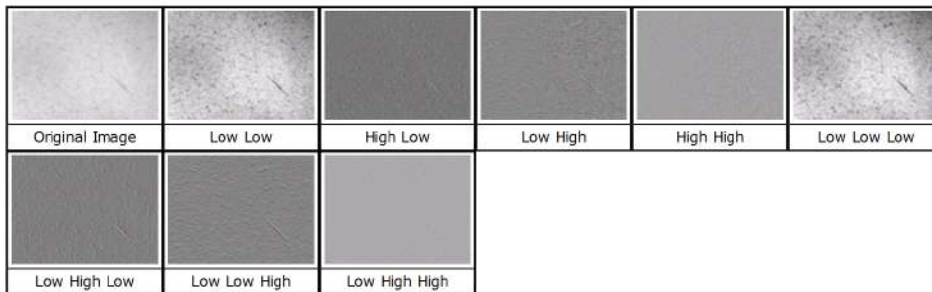
- $s_0(k) = f(k)$ is the input signal to the filter bank.

The following figure illustrates one stage of a 2-dimensional DWFT where the 1-dimensional filters $[h]_{\uparrow 2i}$ and $[g]_{\uparrow 2i}$ are used to perform successive convolution along the rows and columns of the image.



The 1-dimensional filters $[h]_{\uparrow 2^i}$ and $[g]_{\uparrow 2^i}$ are the filters h and g expanded by inserting an appropriate number of zeroes ($2i - 1$) between taps of filter h . Because there is no dyadic subsampling in this DWFT, the decomposed subband images are the same size as the original image.

The texture defect detection algorithm performs two levels of decomposition for each inspection image. The first decomposition step produces a coarse approximation A and horizontal, vertical, and diagonal details H , V , and D . The coarse approximation A is decomposed again to produce subbands AA , AV , AH , and AD . The following figure illustrates the subband images derived from two-level decomposition.



Wavelet Types

By default, the texture defect detection algorithm uses biorthogonal wavelets for subband decomposition, which means that the analysis filters h and g are different from synthesis filters \tilde{h} and \tilde{g} . When compared to orthogonal wavelets, biorthogonal wavelets have higher regularity, have finite impulse response, and preserve linear phase better.

Statistical Feature Extraction

Psychological findings by Julesz² indicate that the human eye cannot make a

preattentive discrimination of textures that have identical first and second order statistics. Therefore, an automatic inspection system that competes with human inspectors should consider second order statistics. Haralick et al.³ define second order statistical features based on the gray-level co-occurrence matrix (GLCM). Haralick features are commonly used for texture identification.

Gray-Level Co-Occurrence Matrix (GLCM)

To extract Haralick features, the texture defect detection algorithm constructs a GLCM from a texture image $I(x, y)$. The GLCM estimates the joint probability that a pixel value occurs at a displacement vector \vec{d} from another pixel value. Given that a texture image $I(x, y)$ is an $N \times M$ matrix consisting of G different grey shades, the GLCM for its GLCM for displacement vector $\vec{d} = (d_x, d_y)$ is a $G \times G$ matrix:

$$P_{\vec{d}}(i, j) = \sum_{x=1}^N \sum_{y=1}^M \delta(I(x, y) = i \wedge I(x + d_x, y + d_y) = j)$$

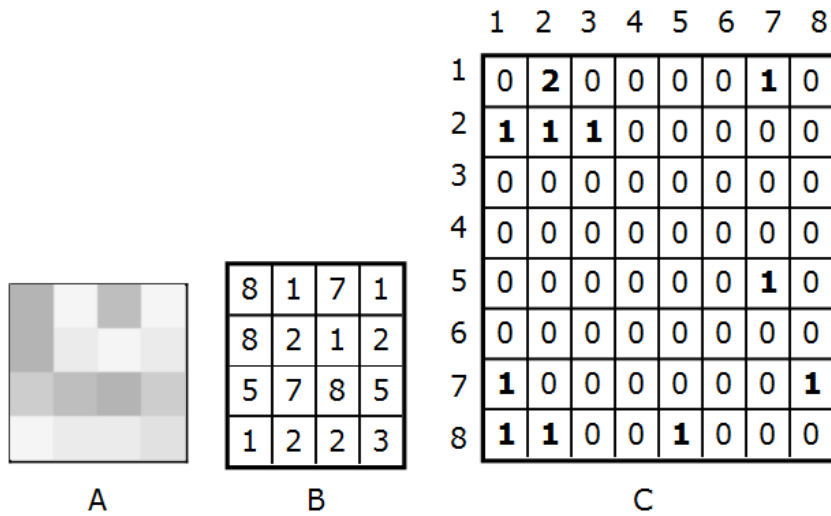
where,

- $\delta\{\text{true}\} = 1$ and $\delta\{\text{false}\} = 0$,
- the number in element (i, j) of the GLCM matrix $P_d(i, j)$ indicates the number of times pixel level i occurs at displacement vector \vec{d} from pixel level j .

The following figures illustrate a GLCM for a 4×4 pixel texture sample $I(x, y)$ that consists of 8 pixel values. Figure A illustrates the texture sample. Figure B illustrates the pixel values for the texture sample. Figure C illustrates the corresponding GLCM for displacement vector $\vec{d} = (1, 0)$. The GLCM is an 8×8 matrix $P(i, j)$ that represents the number of times a pixel value j occurs to the right of a pixel value i . For example, the pixel value 2 is twice located to the right of pixel value 1 in the texture sample. Thus, $P(0, 1)(1, 2) = 2$. Similarly, $P(0, 1)(8, 1) = 1$ because only once does pixel value 1 occur to

2. For more information about the role of second-order statistics in human perception see Julesz, B., Gilbert, E., Shepp, L., and Frisch, H. "Inability of Humans to Discriminate between Visual Textures that Agree in Second-Order Statistics Revisited," *Perception* 2 (1973) 391–405.
3. For more information about Haralick features see Haralick, R., Shanmugam, K., and Dinstein, I. "Textural Features for Image Classification," *Systems, Man and Cybernetics, IEEE Transactions on* 3 no. 6 (1973) 610–621.

the right of pixel value 8.



In applications like texture classification, the co-occurrence matrix can be extracted from the entire texture. In texture defect detection, however, it is better to extract the co-occurrence matrix from local features. The co-occurrence matrix can be extracted locally either by partitioning the texture into adjacent windows and calculating the GLCM for each window, or by moving a single window over the texture and calculating a GLCM that is associated with the center pixel in each instance of the window. By default, the texture defect detection algorithm calculates co-occurrence matrices locally from wavelet decomposed subband images using adjacent 15×15 windows; however, you can change the size of the window and the overlap between windows.

Haralick Feature Extraction

The texture defect detection algorithm extracts five Haralick features—entropy, dissimilarity, contrast, homogeneity, and correlation—from the GLCM calculated at each partition of the subband texture.

$$\text{Entropy} = \sum_{i=1}^G \sum_{j=1}^G P(i, j) (-\ln P(i, j))$$

$$\text{Dissimilarity} = \sum_{i=1}^G \sum_{j=1}^G P(i, j) (i - j)^2$$

$$\text{Contrast} = \sum_{i=1}^G \sum_{j=1}^G P(i, j) (i - j)^2$$

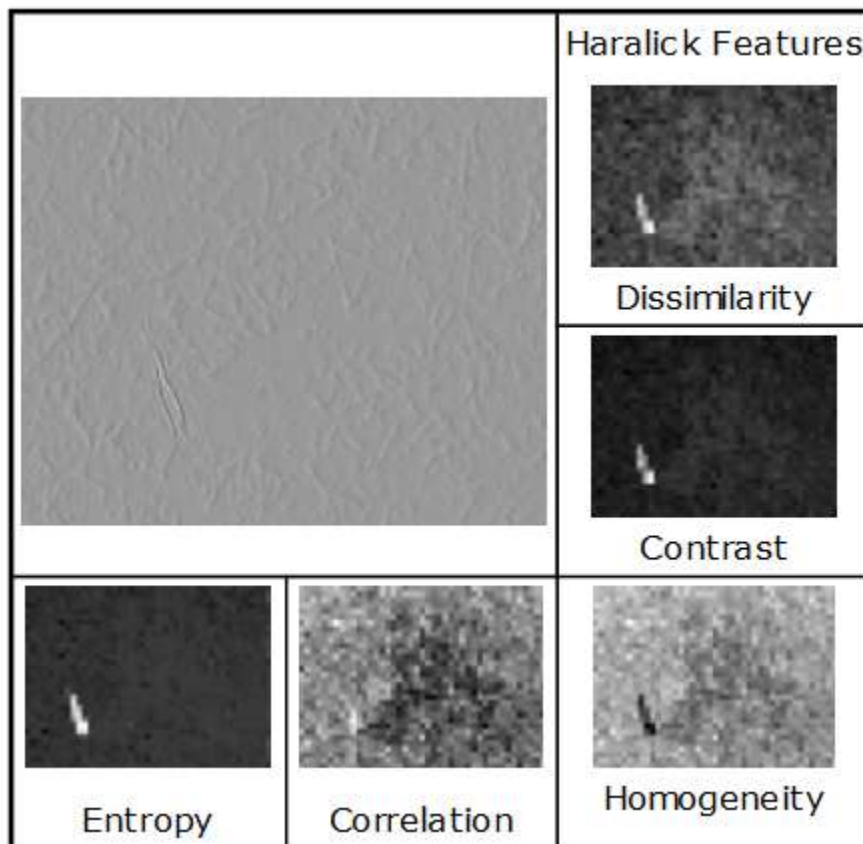
$$Homogeneity = \sum_{i=1}^G \sum_{j=1}^G \frac{P_{i,j}}{1 + (i-j)^2}$$

$$Correlation = \sum_{i=1}^G \sum_{j=1}^G \left(\frac{(i - \mu_i)(j - \mu_j)}{\sqrt{\sigma_i^2 \sigma_j^2}} \right) P_{i,j}$$

where,

- $\mu_i = \sum_{j=1}^G j P_{i,j}$ and $\mu_j = \sum_{i=1}^G i P_{i,j}$ are the GLCM means, and
- $\sigma_i = \sqrt{\sum_{j=1}^G P_{i,j} (j - \mu_i)^2}$ and $\sigma_j = \sqrt{\sum_{i=1}^G P_{i,j} (i - \mu_j)^2}$ are the GLCM variances.

The following figure illustrates the High Low wavelet subband and its five corresponding Haralick feature maps. Note that the Haralick features clearly distinguish the texture defect from the texture, with the entropy and contrast features exhibiting the highest differentiation for this example.



The number of feature vector elements extracted to represent a texture sample is equivalent to the number of selected wavelet subbands multiplied by 5 (the number of Haralick features extracted from each wavelet subband). For example, if an application uses all 8 subbands, the size of the resulting feature vector is 40.

Support Vector Machine Classifier

The final stage of texture defect detection involves classifying pixels as either texture or defect, based on the texture features extracted from a neighborhood around the pixel. This type of classification is an outlier detection or one-class classification problem. In a one-class classification problem, a known class is represented by numerous trained samples while an unknown class is represented by few or no samples. For example, a known class may consist of texture samples and an unknown class may consist of texture defects that vary so greatly in size, shape, or orientation that they are impossible to document.

The texture defect detection uses a one-class ***SVM classifier***. SVM classifiers identify a separating surface, or hyperplane, located at the maximum possible distance from the nearest data point in either of two classes⁴. SVM classifiers have very good generalization capabilities and perform well in high dimensional feature spaces.

Related concepts:

- [Support Vector Machines](#)

Flat Field Correction

This section contains information about flat field correction.

Introduction to Flat Field Correction

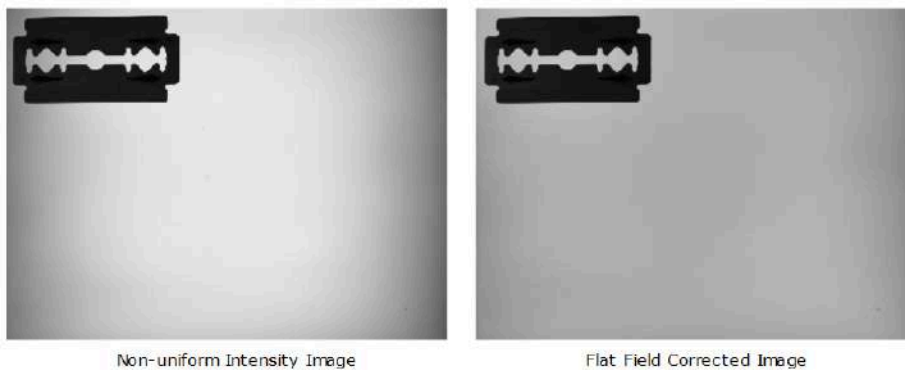
Flat field correction is the process of correcting the non-uniform intensity in images. Non-uniform intensity occurs in the images due to lens light fall off (also known as vignetting), non-linear surfaces, and non-uniform lighting.

4. For more information about texture inspection with SVM classifiers see Jahanbin, S., Bovik, A., Perez, E., Nair, D. "Automated Inspection of Textured Surfaces by Support Vector Machines," SPIE Conference on Optics and Photonics San Diego, California, August 2–6 (2009).

Flat field correction requires flat field images, also called bright field images, and dark field images from the imaging setup. Flat field images are mandatory for correcting the images.

Flat field images need to be captured using the imaging setup with a bright background or the flat field images can be estimated programmatically using a mathematical model. Once flat field images are captured or estimated, they can be used to correct the non-uniform intensity in the images.

The following illustration shows an image with non-uniform intensity and the flat field corrected image.

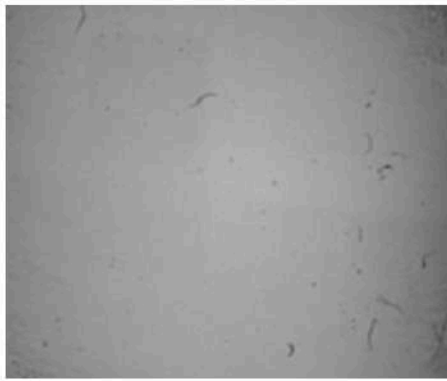


When to Use Flat Field Correction

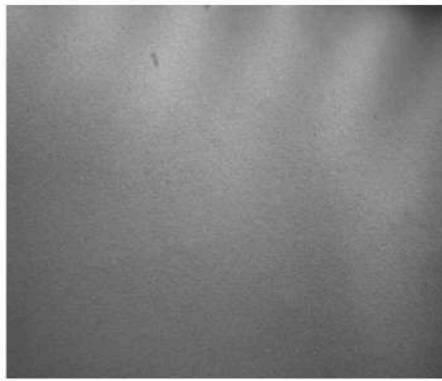
Use flat field correction to do the following:

- Correct non-uniform intensity in an image.
- Pre-process an image before applying edge detection or pattern matching.
- Correct intensity fall off to emphasize a defect in the image.
- Correct sensor dust and impurity.

The following illustration shows sensor dust and lighting issues due to improper line light setup.



Sensor Dust



Improper Lighting Setup

Flat Field Correction Concepts

Flat field correction uses the following equation to correct images:

$$\text{Corrected Image} = \frac{(\text{Image} - \text{Dark Field Image}) * \text{Median of (Flat Field Image} - \text{Dark Field Image)}}{(\text{Flat Field Image} - \text{Dark Field Image})} * \text{Correction Factor}$$

where,

- **Image** is the image to be corrected,
- **Dark Field Image** is the image that captures the dark currents in the sensor,
- **Flat Field Image** is the image intensity profile image,
- **Correction Factor** is a constant to bias the brightness of the corrected image.

Optimized Correction

The flat field correction algorithm provides an option to correct the image faster by storing the following equation component in memory.

$$\text{Flat Field Stored Image} = \frac{\text{Median of (Flat Field Image} - \text{Dark Field Image)}}{(\text{Flat Field Image} - \text{Dark Field Image})} * \text{Correction Factor}$$

When the optimized correction is enabled, the algorithm computes and stores the above component when the flat field image is first used. The stored value is reused for each image thereafter. The flat field correction equation with the optimization enabled:

$$\text{Corrected Image} = (\text{Image} - \text{Dark Field Image}) * \text{Flat Field Stored Image}$$

Flat Field Correction In-Depth Discussion

NI Vision provides two techniques to generate flat field images.

- User-controlled—Capture images with a bright background using an actual imaging setup.
- Estimation—Estimate the flat field image using the Estimate Flat Field Model algorithm.

User-controlled Technique

Use this technique:

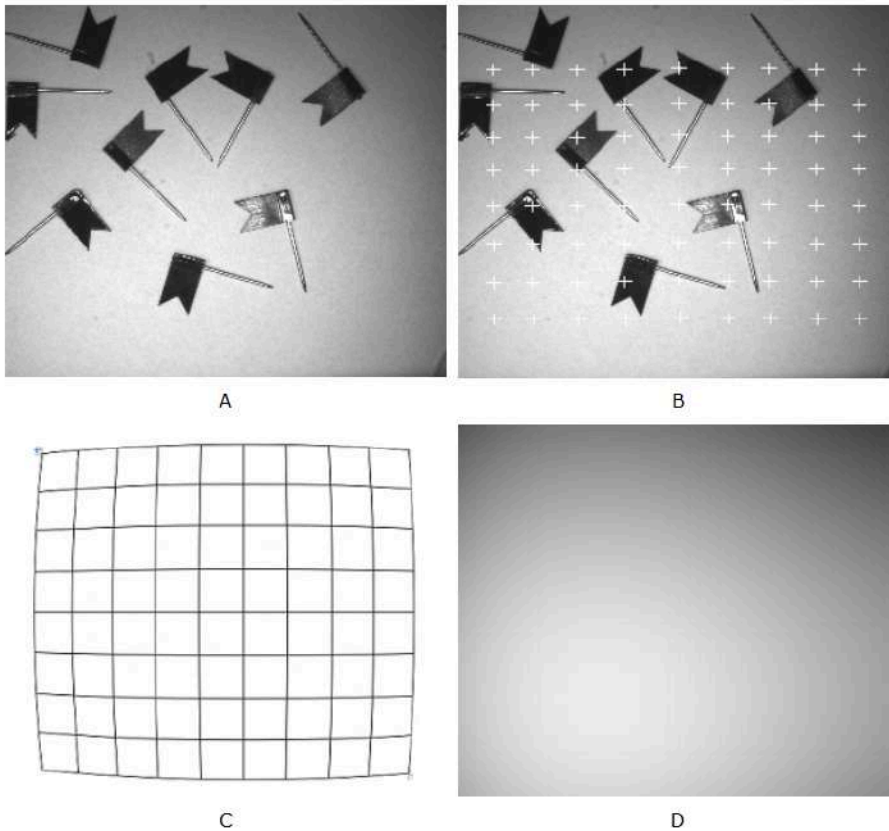
- to achieve an accurate representation of back ground intensity,
- to remove sensor noise, such as dust,
- if the imaging setup is easily accessible in an industrial environment, and recapturing the images are necessary due to lighting changes.

In this technique, use your imaging set up to capture flat field images with a bright background after removing the object under inspection. Typically, multiple images are captured (more than 10), and then averaged to create the flat field image. Capture the dark field images by covering the camera lens.

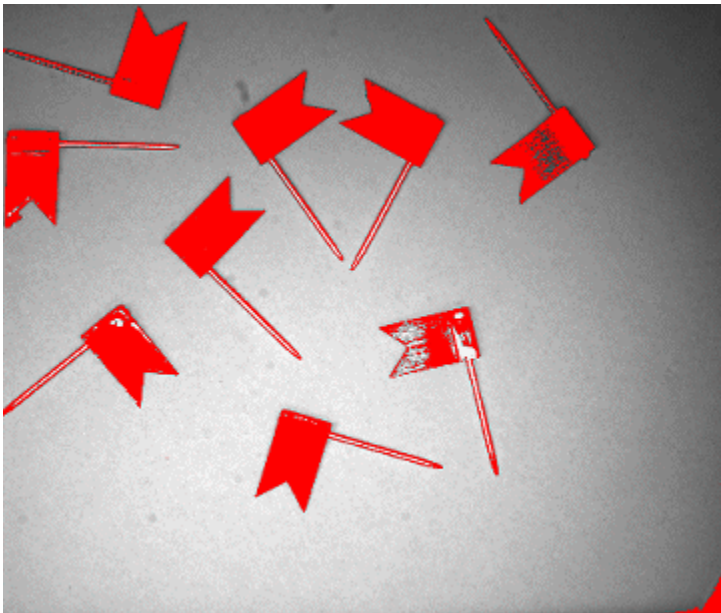
This process should be repeated whenever the imaging setup (lens, light, and position) changes. It is important to capture multiple frames to nullify the texture of the background. Use the IMAQ Compute Median Image VI and the IMAQ Compute Average Image VI to obtain the median or average of multiple images. Pass the flat field and dark field images to the IMAQ Flat Field Correction VI to create a corrected image.

Estimation Technique

The following images illustrate the flat field estimation process. Image A is an image to estimate the flat field image. Image B illustrates the sampling points. Image C illustrates the fitted 2D polynomial model. Image D is the estimated flat field image using the surface fit.



The flat field image can be estimated more accurately by enabling the **Estimate Background?** boolean in the VI. This parameter detects the background region in the image and performs a surface fit using only background pixels. The following figure illustrates the detected background region by masking the foreground objects.



The **Estimate Background?** option provides the following options to detect the

background region:

- **Polynomial**—Uses a polynomial algorithm with a specified Polynomial Degree to estimate the background.
- **Background Correction**—Performs background correction to eliminate non-uniform lighting effects and then performs thresholding using the interclass variance thresholding algorithm.
- **NiBlack**—Computes thresholds for each pixel based on its local statistics using the NiBlack local thresholding algorithm.

Particle Analysis

This section describes conceptual information about particle analysis, including thresholding, morphology, and particle measurements.

Introduction

You can use particle analysis to detect connected regions or groupings of pixels in an image and then make selected measurements of those regions. These regions are commonly referred to as particles. A particle is a contiguous region of nonzero pixels. You can extract particles from a grayscale image by thresholding the image into background and foreground states. Zero valued pixels are in the background state, and all nonzero valued pixels are in the foreground.

Particle analysis consists of a series of processing operations and analysis functions that produce information about particles in an image. Using particle analysis, you can detect and analyze any 2D shape in an image.

When to Use

Use particle analysis when you are interested in finding particles whose spatial characteristics satisfy certain criteria. In many applications where computation is time-consuming, you can use particle filtering to eliminate particles that are of no interest based on their spatial characteristics, and keep only the relevant particles for further analysis.

You can use particle analysis to find statistical information, such as the presence of particles, their number and size, and location. This information allows you to perform many machine vision inspection tasks, such as detecting flaws on silicon wafers, detecting soldering defects on electronic boards, or web inspection applications such as finding structural defects on wood planks or detecting cracks on plastics sheets. You also can locate objects in motion control applications.

In applications where there is a significant variance in the shape or orientation of an object, particle analysis is a powerful and flexible way to search for the object. You can use a combination of the measurements obtained through particle analysis to define a

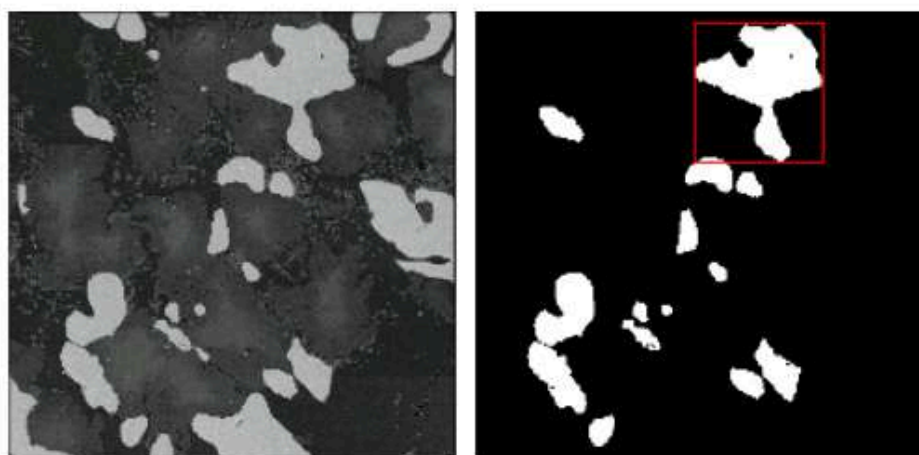
feature set that uniquely defines the shape of the object.

Concepts

A typical particle analysis process scans through an entire image, detects all the particles in the image, and builds a detailed report on each particle. You can use multiple parameters such as perimeter, angle, area, and center of mass to identify and classify these particles. Using multiple parameters can be faster and more effective than pattern matching in many applications.

By using different sets of parameters, you can also uniquely identify a feature in an image. For example, you could use the area of the template particle as a criterion for removing all particles that do not match it within some tolerance. You then can perform a more refined search on the remaining particles using another list of parameter tolerances.

The following figure shows a sample list of parameters that you can obtain in a particle analysis application. The binary image in this example was obtained by thresholding the source image and removing particles that touch the border of the image. You can use these parameters to identify and classify particles. The following figure shows the values obtained for the particle enclosed in a rectangle.



A. Source Image

B. Binary Image

Particle Measurement	Values
Area	2456
Number of Holes	1
Bounding Rect	
Left	127
Top	8
Right	200
Bottom	86
Center of Mass	
X	167.51
Y	37.61
Orientation	82.36°
Dimensions	
Width	73
Height	78

To use particle analysis, first create a binary image using a thresholding process. You then can improve the binary image using morphological transformations and make measurements on the particles in the image.

Image Segmentation

This section contains information about segmenting images using global grayscale thresholding, global color thresholding, local thresholding, and morphological

segmentation. Image segmentation is the process of separating objects from the background and each other so that each object can be identified and characterized. Refer to **particle measurements** for information about characterizing objects after segmentation.

Related concepts:

- [Particle Measurements](#)

Thresholding

Thresholding uses the pixel values in an image to segment the image into two regions: a particle region, which contains the objects under inspection, and a background region.

A range of pixel values is defined, either by the user or automatically, as the threshold. Any pixel value outside the range becomes 0, and any pixel value inside the range becomes 1, or a user-defined value. Thresholding results in a binary image.

When to Use

Use thresholding to create a binary image and focus inspection on specific areas of interest.

Thresholding is often the first step in machine vision applications such as particle analysis, golden template comparison, and binary particle classification.

Vision supports the following thresholding methods:

- **Global Grayscale Thresholding**—Use Global Grayscale Thresholding on grayscale images with uniform lighting. Global Grayscale Thresholding can be performed with the following methods:
 - **Manual Threshold**—Enables the user to manually set the threshold range. Recommended on images with good contrast and uniform lighting.
 - **Clustering**—Thresholds the image into more than two classes. This is the most common automatic thresholding method.
 - **Entropy**—Detects small areas of interest in the image. This method is used for applications such as fault detection.

- **Inter Variance**—Use for images in which classes are not overly disproportionate, and when the object of interest and the background contain a comparable number of pixels. For satisfactory results, the smallest class must be at least 5% of the largest one.
- **Metric**—Calculates a value for each threshold that is determined by the surfaces representing the initial gray scale. Use this method when the object of interest and the background contain a comparable number of pixels.
- **Moments**—Use for images that have poor contrast.
- **Global Color Thresholding**—Use Global Color Thresholding on color images with uniform lighting.
- **Local Thresholding**—Use Local Thresholding to isolate regions of interest in images that exhibit non-uniform lighting changes, such as shadows or a strong illumination gradient. Local Thresholding can be performed with the following methods:
 - **Niblack Algorithm**—Effective for applications such as display inspection and OCR images.
 - **Sauvola Algorithm**—Results in less noise and preserves the shape of the particles.
 - **Modified Sauvola Algorithm**—Less computationally intensive than the Sauvola algorithm.
 - **Background Correction Algorithm**—Reduces noise in large, empty areas.

Global Grayscale Thresholding

Global grayscale thresholding includes manual thresholding and automatic thresholding techniques.

When to Use

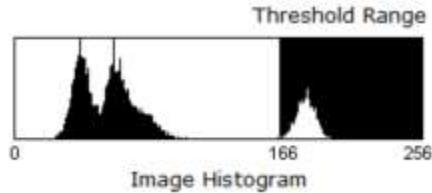
Global thresholding works best when the inspection images exhibit uniform lighting both within each image and across multiple images.

Concepts

Particles are characterized by an intensity range. They are composed of pixels with gray-level values belonging to a given threshold interval (overall luminosity or gray shade). All other pixels are considered to be part of the background.

Thresholding sets all pixels that belong to a range of pixel values, called the threshold interval, to 1 or a user-defined value, and it sets all other pixels in the image to 0. Pixels inside the threshold interval are considered part of a particle. Pixels outside the threshold interval are considered part of the background.

The following figure shows the histogram of an image. All pixels in the image whose values range from 166 to 255 are considered particle pixels.

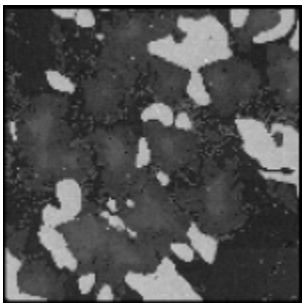


Manual Threshold

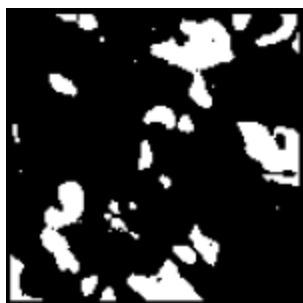
The threshold interval in a manual threshold has two user-defined parameters: lower threshold and upper threshold. All pixels that have gray-level values equal to or greater than the lower threshold and equal to or smaller than the upper threshold are selected as pixels belonging to particles in the image.

Manual Thresholding Example

This example uses the following source image.



Highlighting the pixels that belong to the threshold interval [166, 255] (the brightest areas) produces the following image.



Automatic Threshold

Vision has five automatic thresholding techniques.

- Clustering
- Entropy
- Inter Variance
- Metric
- Moments

In contrast to manual thresholding, these techniques do not require that you set the lower and upper threshold values. These techniques are well suited for conditions in which the light intensity varies from image to image.

Clustering is the only multi-class thresholding method available. Clustering operates on multiple classes so you can create tertiary or higher-level images.

The other four methods—entropy, metric, moments, and interclass variance—are reserved for strictly binary thresholding techniques. The choice of which algorithm to apply depends on the type of image to threshold.

Depending on your source image, it is sometimes useful to invert the original grayscale image before applying an automatic threshold function, such as entropy and moments. This is especially true for cases in which the background is brighter than the foreground.

Clustering

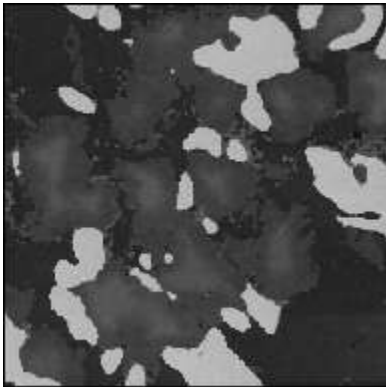
Clustering is the most frequently used automatic thresholding method. Use the clustering method when you need to threshold the image into more than two classes.

Clustering sorts the histogram of the image within a discrete number of classes corresponding to the number of phases perceived in an image. The gray values are determined, and a barycenter is determined for each class. This process repeats until it obtains a value that represents the center of mass for each phase or class.

Example of Clustering

This example uses a clustering technique in two and three phases on an image. Notice that the results from this function are generally independent of the lighting conditions as well as the histogram values from the image.

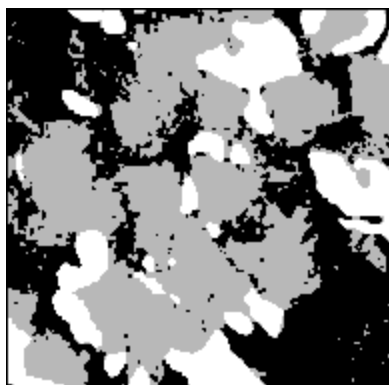
This example uses the following original image.



Clustering in two phases produces the following image.



Clustering in three phases produces the following image.



Entropy

Based on a classical image analysis technique, entropy is best for detecting particles that are present in minuscule proportions on the image. For example, this function would be suitable for fault detection.

Inter Variance

Inter variance is based on discriminant analysis. An optimal threshold is determined by maximizing the interclass variation with respect to the threshold.

Metric

For each threshold, a value determined by the surfaces representing the initial gray scale is calculated. The optimal threshold corresponds to the smallest value.

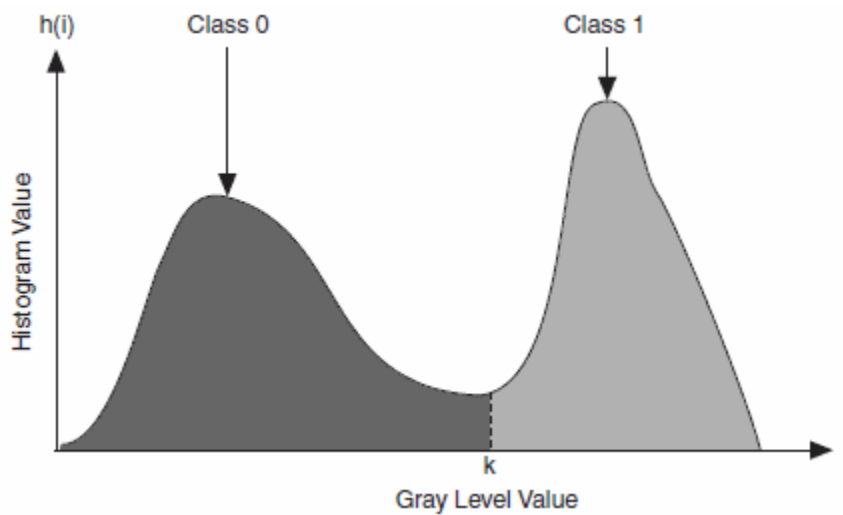
Moments

This technique is suited for images that have poor contrast. The moments method is based on the hypothesis that the observed image is a blurred version of the theoretically binary original. The blurring that is produced from the acquisition process, caused by electronic noise or slight defocalization, is treated as if the statistical moments of average and variance were the same for both the blurred image and the original image. This function recalculates a theoretical binary image.

In-Depth Discussion

All automatic thresholding methods use the histogram of an image to determine the threshold. The following figure explains the notations used to describe the parameters of the histogram. These notations are used throughout this section to show how each

automatic thresholding method calculates the threshold value for an image.



- i represents the gray level value,
- k represents the gray level value chosen as the threshold,
- $h(i)$ represents the number of pixels in the image at each gray level value,
- N represents the total number of gray levels in the image (256 for an 8-bit image),
- n represents the total number of pixels in the image.

Use the automatic thresholding techniques to determine the threshold pixel value k such that all gray-level values less than or equal to k belong to one class 0 and the other gray level values belong to another class 1.

Clustering

The threshold value is the pixel value k for which the following condition is true:

$$\frac{\mu_1 + \mu_2}{2} = k$$

where:

- μ_1 is the mean of all pixel values that lie between 0 and k ,
- and μ_2 is the mean of all the pixel values that lie between $k + 1$ and 255.

Entropy

In this method, the threshold value is obtained by applying information theory to the

histogram data. In information theory, the entropy of the histogram signifies the amount of information associated with the histogram. Let

$$p(i) = \frac{h(i)}{\sum_{i=0}^{N-1} h(i)}$$

represent the probability of occurrence of the gray level i . The entropy of a histogram of an image with gray levels in the range $[0, N - 1]$ is given by

$$H = \sum_{i=0}^{N-1} p(i) \log_2 p(i)$$

If k is the value of the threshold, then the two entropies

$$H_b = - \sum_{i=0}^k p(i) \log_2 p_b(i)$$

$$H_w = - \sum_{i=k+1}^{N-1} p_w(i) \log_2 p_w(i)$$

represent the measures of the entropy (information) associated with the black and white pixels in the image after thresholding. $P_b(i)$ is the probability of the background, and $P_b(w)$ is the probability of the object.

The optimal threshold value is gray-level value that maximizes the entropy in the thresholded image given by

$$H_b + H_w$$

Simplified, the threshold value is the pixel value k at which the following expression is maximized:

$$-\frac{1}{\sum_{i=0}^k h(i)} \sum_{i=0}^k \log_2(h(i) + 1) h(i) - \frac{1}{\sum_{i=k+1}^{N-1} h(i)} \sum_{i=k+1}^{N-1} \log_2(h(i) + 1) h(i) + \log_2 \left(\sum_{i=0}^k h(i) \sum_{i=k+1}^{N-1} h(i) \right)$$

Inter Variance

The threshold value is the pixel value k at which the following expression is maximized:

$$\sigma_B^2(k) = \frac{(\mu_T \omega(k) - \mu(k))^2}{\omega(k)(1 - \omega(k))}$$

where

$$\mu(k) = \sum_{i=0}^k ip(i)$$

$$\mu_T = \sum_{i=0}^{N-1} ip(i)$$

$$\omega(k) = \sum_{i=0}^k p(i)$$

where,

- $\sigma_B^2 k$
is the Interclass Variance.
- $\mu(k)$
is the mean of the class containing bins 0 to k.
- μ_T
is the overall mean.
- $\omega(k)$
is the class probability.

Metric

The threshold value is the pixel value k at which the following expression is minimized:

$$\sum_{i=0}^k h(i) \left((i - \mu_1) \right) + \sum_{i=k+1}^{N-1} h(i) \left((i - \mu_2) \right)$$

where,

- μ_1 is the mean of all pixel values in the image that lie between 0,
- k, and μ_2 is the mean of all the pixel values in the image that lie between k + 1 and 255.

Moments

In this method the threshold value is computed in such a way that the moments of the

image to be thresholded are preserved in the binary output image.

The k th moment m of an image is calculated as

$$m_k = \frac{1}{n} \sum_{i=0}^{N-1} i^k h(i)$$

where n is the total number of pixels in the image.

Global Color Thresholding

Color thresholding converts a color image into a binary image.

When to Use

Threshold a color image when you need to isolate features for analysis and processing or to remove unnecessary features.

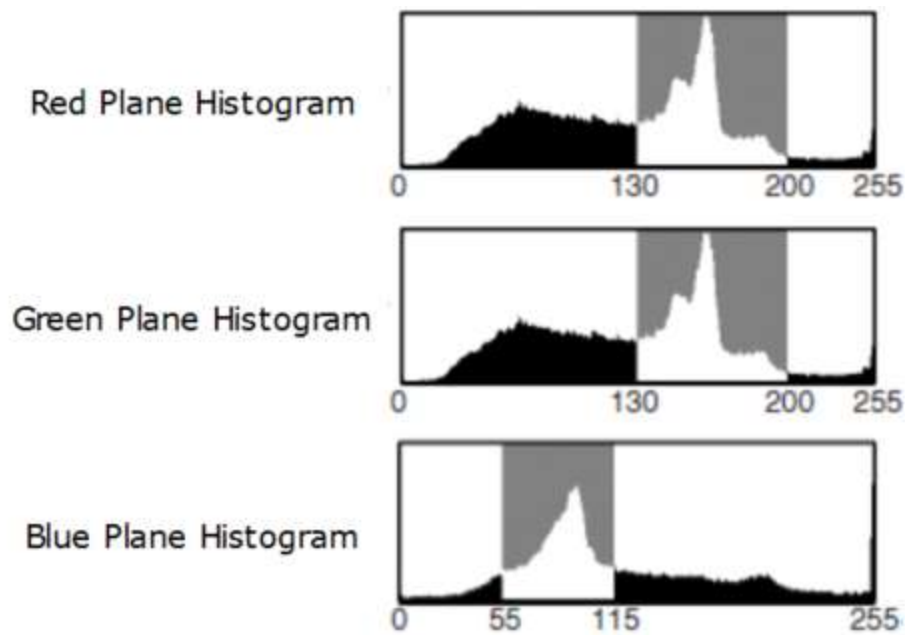


Note Before performing a color threshold, you may need to enhance your image with lookup tables or the equalize function.

Concepts

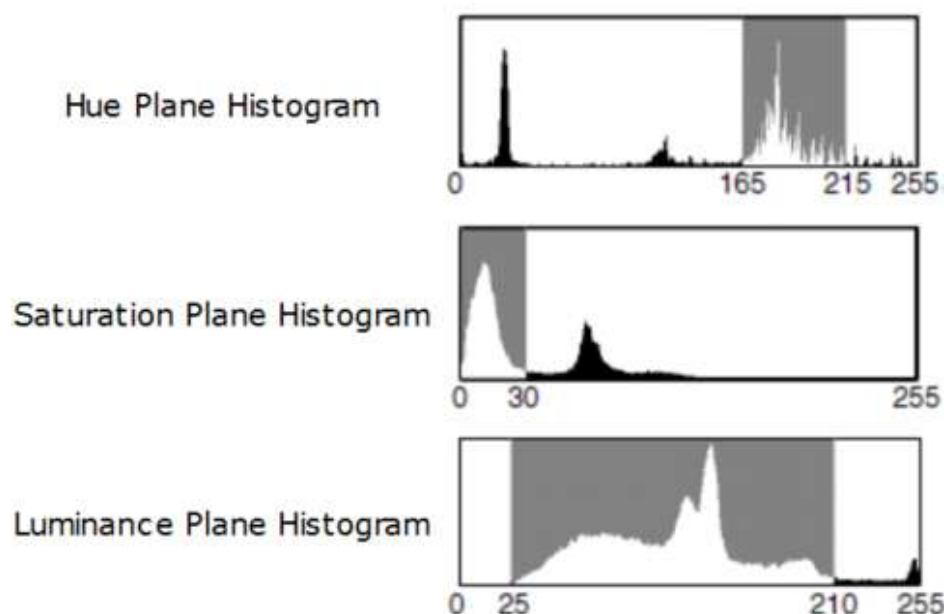
To threshold a color image, specify a threshold interval for each of the three color components. A pixel in the output image is set to 1 if and only if its color components fall within the specified ranges. Otherwise, the pixel value is set to 0.

The following figure shows the histograms of each plane of a color image stored in RGB format. The gray shaded region indicates the threshold range for each of the color planes. For a pixel in the color image to be set to 1 in the binary image, its red value should lie between 130 and 200, its green value should lie between 100 and 150, and its blue value should lie between 55 and 115.



To threshold an RGB image, first determine the red, green, and blue values of the pixels that constitute the objects you want to analyze after thresholding. Then, specify a threshold range for each color plane that encompasses the color values of interest. You must choose correct ranges for all three color planes to isolate a color of interest.

The following figure shows the histograms of each plane of a color image stored in HSL format. The gray shaded region indicates the threshold range for each of the color planes. For a pixel in the color image to be set to 1 in the binary image, its hue value should lie between 165 and 215, its saturation value should lie between 0 and 30, and its luminance value should lie between 25 and 210.



The hue plane contains the main color information in an image. To threshold an HSL image, first determine the hue values of the pixels that you want to analyze after thresholding. In some applications, you may need to select colors with the same hue value but various saturation values. Because the luminance plane contains only information about the intensity levels in the image, you can set the luminance threshold range to include all the luminance values, thus making the thresholding process independent from intensity information.

Local Thresholding

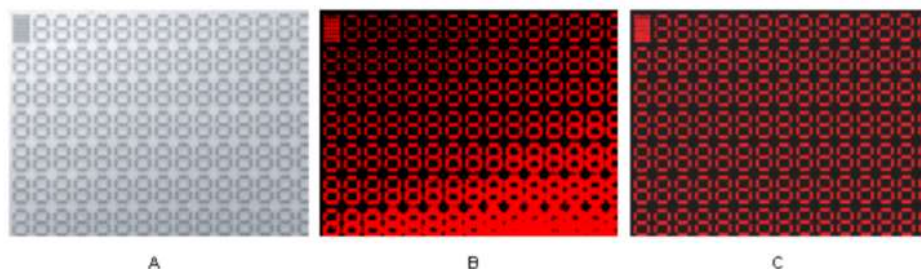
Local thresholding, also known as locally adaptive thresholding, is like global grayscale thresholding in that both create a binary image by segmenting a grayscale image into a particle region and a background region. Unlike global grayscale thresholding, which categorizes a pixel as part of a particle or the background based on a single threshold value derived from the intensity statistics of the entire image, local thresholding categorizes a pixel based on the intensity statistics of its neighboring pixels.

When to Use

Use local thresholding to isolate objects of interest from the background in images that exhibit nonuniform lighting changes. Nonuniform lighting changes, such as those resulting from a strong illumination gradient or shadows, often make global

thresholding ineffective.

The following figure shows the effect of global thresholding and local thresholding on an image with nonuniform lighting changes. Figure A shows the original inspection image of LCD digits. Figure B shows how a global threshold segments the inspection image. Notice that many of the nondigit pixels in the bottom, right corner are erroneously selected as particles. Figure C shows how a local threshold segments the inspection image. Only pixels belonging to LCD digits are selected as particles.



Concepts

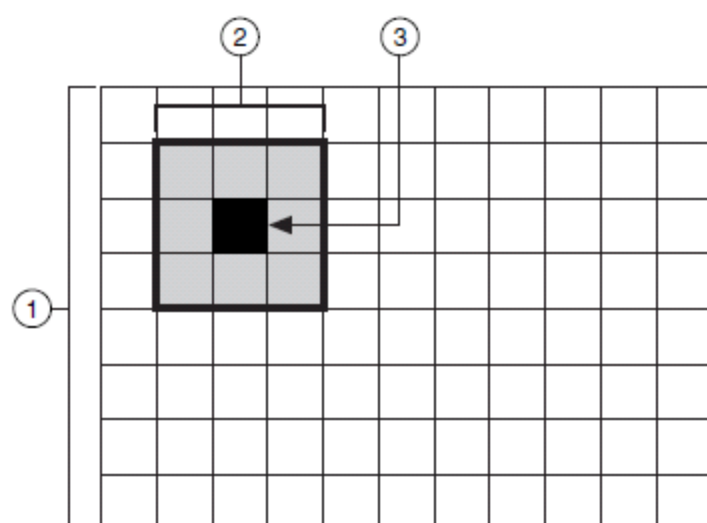
The local thresholding algorithm calculates local pixel intensity statistics—such as range, variance, surface fitting parameters, or their logical combinations—for each pixel in an inspection image. The result of this calculation is the local threshold value for the pixel under consideration. The algorithm compares the original intensity value of the pixel under consideration to its local threshold value and determines whether the pixel belongs to a particle or the background.

A user-defined window specifies which neighboring pixels are considered in the statistical calculation. The default window size is 32×32 .



Note Even-numbered window dimensions and odd-numbered window dimensions produce the same center pixel. For example, in the following figure, the pixel under consideration is the same for a 4×4 local thresholding window as it is for a 3×3 local thresholding window.

The window size should be approximately the size of the smallest object you want to separate from the background. The following figure shows a simplified local thresholding window.



1. Image
2. Local Thresholding Window
3. Pixel under Consideration



Note The pixel intensities of all of the pixels in the window, including the pixel under consideration, are used to calculate the local threshold value.

A typical local thresholding function requires a large amount of computation time. Also, the time a typical local thresholding function takes to complete often varies depending on the window size. This lack of determinism prevents local thresholding from being used in real-time applications. The Vision local thresholding function uses a fully optimized, efficient algorithm implementation whose computation speed is independent of the window size. This significantly reduces the computation cost and makes using the function in a real-time segmentation applications possible.

The following sections describe the algorithms available in the Vision local thresholding function.



Note You must specify whether you are looking for dark objects on a light background or light objects on a dark background regardless of which algorithm you use.

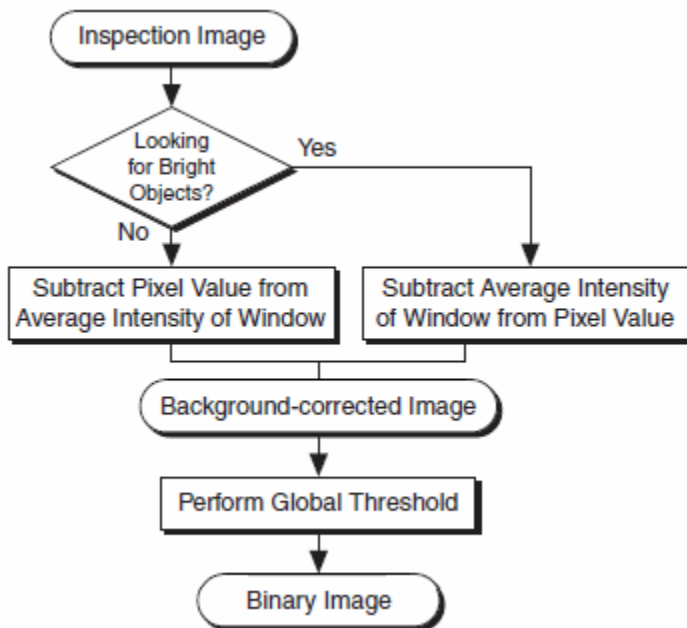
Niblack Algorithm

This algorithm has been experimentally shown to be the best among eleven locally adaptive thresholding algorithms, based on a goal-directed evaluation from OCR and map image segmentation applications. The algorithm is effective for many image thresholding applications, such as display inspection and OCR.

The Niblack algorithm is sensitive to the window size and produces noisy segmentation results in areas of the image with a large, uniform background. To solve this problem, the Vision local thresholding function computes a deviation factor that the algorithm uses to correctly categorize pixels.

Background Correction Algorithm

This algorithm combines the local and global thresholding concepts for image segmentation. The following figure illustrates the background correction algorithm.



The background-corrected image is thresholded using the interclass variance automatic thresholding method described in the automatic threshold section of this chapter.

In-Depth Discussion

The following sections provide an in-depth discussion of the algorithms used by each

thresholding method.

Niblack

In the Niblack algorithm, the local threshold value $T(i, j)$ at pixel (i, j) is calculated as

$$T(i, j) = m(i, j) + k \cdot \omega(i, j)$$

where,

- $m(i, j)$ is the local sample mean,
- k is the deviation Niblack/Sauvola factor,
- $\omega(i, j)$ is the standard deviation.

Each image pixel $I(i, j)$ is categorized as a particle or background pixel based on the following:

if $I(i, j) > T(i, j)$, $I(i, j) = \text{particle}$

else $I(i, j) = \text{background}$



Tip Setting k to 0 to increases the computation speed of the Niblack algorithm.

Sauvola

In the Sauvola algorithm, the windowed standard deviation is normalized by dividing the windowed standard deviation by the dynamic range of the standard deviation (R). This results in less noise and preserves the shape of the particles. In the Sauvola algorithm, the local threshold value $T(i, j)$ at pixel (i, j) is calculated as

$$T_{(i, j)} = m(i, j) * \left(1 + k * \left(1 - \frac{\omega(i, j)}{R} \right) \right)$$

where:

- $m(i, j)$ and $w(i, j)$ are the mean and standard deviation calculated in a window,
- k is the Niblack/Sauvola deviation factor,

- R is the Sauvola deviation range.

Modified Sauvola

In the Modified Sauvola algorithm, the windowed mean deviation is normalized by dividing it by the dynamic range of the standard deviation (R). This method uses mean deviation instead of standard deviation, making it less computationally intensive than the Sauvola algorithm. In the Modified Sauvola algorithm, the local threshold value $T(i, j)$ at pixel (i, j) is calculated as:

$$T_{(i, j)} = m(i, j) * \left(1 + k * \left(1 - \frac{d(i, j)}{R} \right) \right)$$

$$d(i, j) = l(i, j) - m(i, j)$$

where

- $m(i, j)$ is the local mean and $d(i, j)$ are the local mean deviation,
- k is the Niblack/Sauvola deviation factor,
- R is the Sauvola deviation range.

Background Correction

In the background correction algorithm, the background-corrected image $B(i, j)$ is calculated as:

$$B(i, j) = l(i, j) - m(i, j)$$

where:

- $m(i, j)$ is the local mean at pixel (i, j) .

Thresholding Considerations

A critical and frequent problem in segmenting an image into particle and background regions occurs when the boundaries are not sharply demarcated. In such a case, the determination of a correct threshold interval becomes subjective. Therefore, you may want to enhance your images before thresholding to outline where the correct borders lie. You can use lookup tables, filters, FFTs, or equalize functions to enhance your

images. Observing the intensity profile of a line crossing a boundary area is also helpful in selecting a correct threshold value. Finally, keep in mind that morphological transformations can help you retouch the shape of binary particles and, therefore, correct unsatisfactory selections that occurred during thresholding.

Morphological Segmentation

In some image analysis and machine vision applications—such as industrial defect inspection or biomedical imaging—segmentation based on thresholding or edge detection is not sufficient because the image quality is insufficient or the objects under inspection touch or overlap. In such applications, morphological segmentation is an effective method of image segmentation. Morphological segmentation partitions an image based on the topographic surface of the image. The image is separated into non-overlapping regions with each region containing a unique particle.

When to Use

Thresholding can segment objects from the background only if the objects are well separated from each other and have intensity values that differ significantly from the background. Binary morphology operators, such as close or open, often return inaccurate results when segmenting overlapping particles.

Use morphological segmentation to segment touching or overlapping objects from each other and from the background. Also, use morphological segmentation when the objects have intensity values similar to the background.



Note The morphological segmentation process described in the following section works best when the objects under inspection are convex.

Concepts

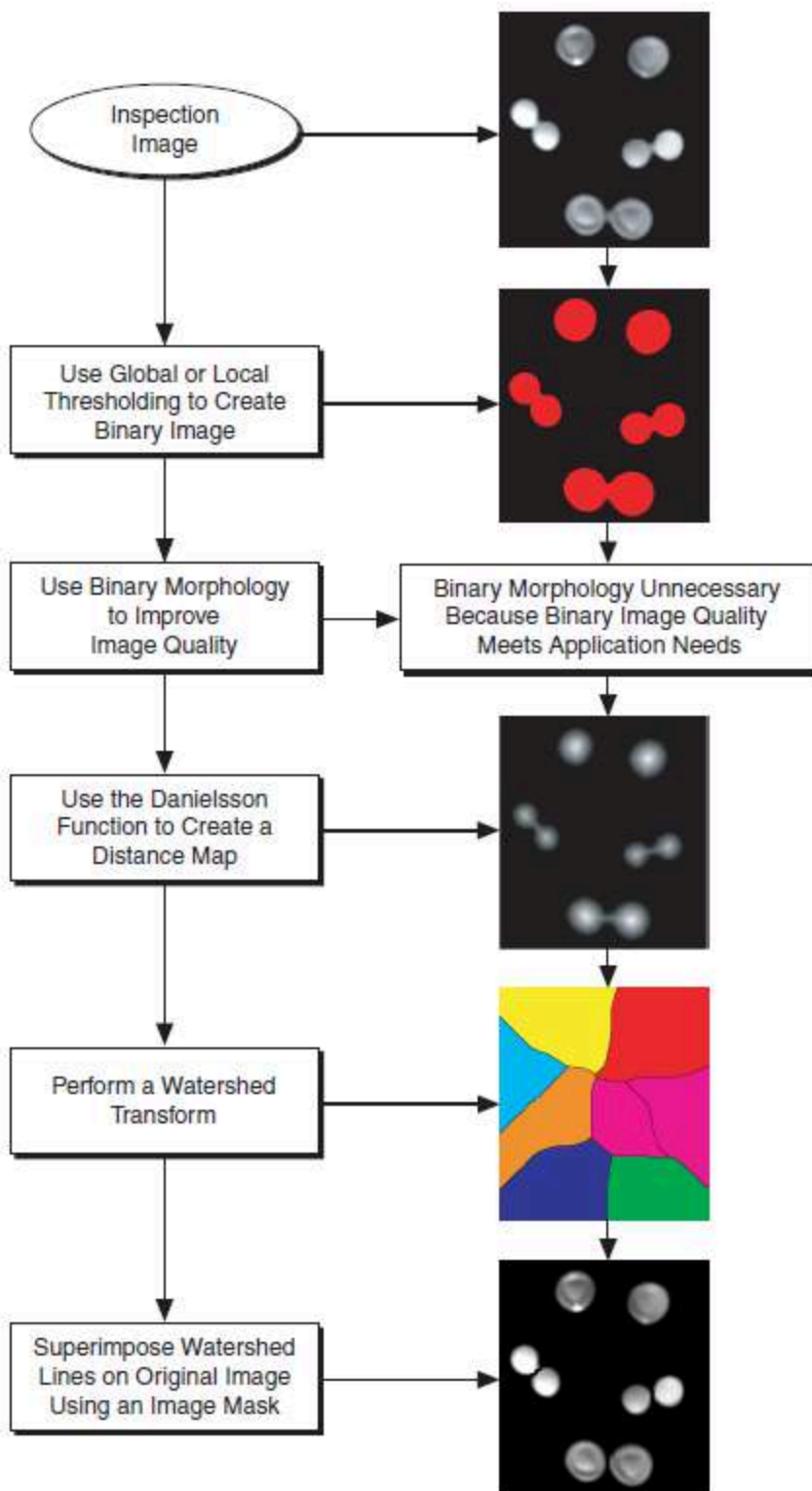
Morphological segmentation is a multiple-step process involving several Vision functions. The following list describes each morphological segmentation step and where to find more information about each step.

1. Use a global or local threshold to create a binary image. Refer to ***global grayscale thresholding***, ***global color thresholding***, or ***local***

thresholding for more information about thresholding.

2. If necessary, use **binary morphology** operations to improve the quality of the image by filling holes in particles or remove extraneous noise from the image.
3. Use the **Danielsson function** to transform the binary image into a grayscale distance map in which each particle pixel is assigned a gray-level value equal to its shortest Euclidean distance from the particle border.
4. Perform a watershed transform on the distance map to find the watershed separation lines.
5. Superimpose the watershed lines on the original image using an **image mask**.

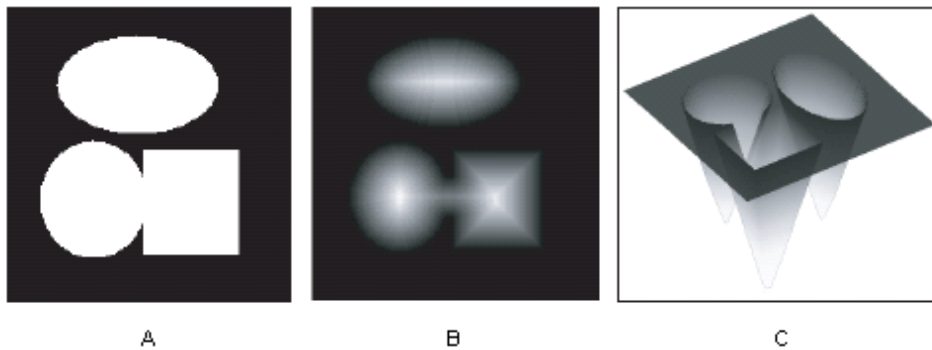
The following figure summarizes the morphological segmentation process and shows an example of each step.



Watershed Transform

In geography, a watershed is an area of land from which all rain that falls on the land flows into a specific body of water. In imaging, the watershed transform algorithm

considers the objects under inspection to be the bodies of water. The following figure illustrates this concept.

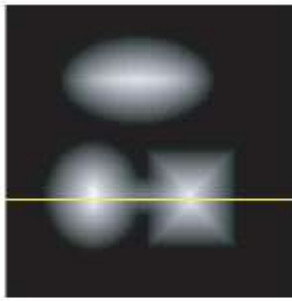


- Figure A shows an inspection image after it has been thresholded.
- Figure B shows the distance map of objects in the image using the gradient palette.
- Figure C shows the topographic surface of the distance map.

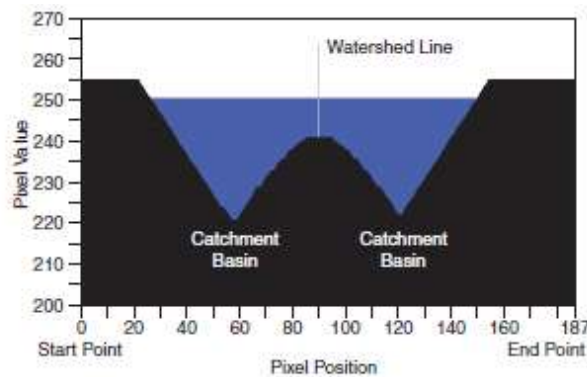
Each object from the inspection image forms a deep, conical lake called a catchment basin. The pixels to which the distance map function assigned the highest value represent the deepest parts of each catchment basin. The image background represents the land surrounding the catchment basins.

To understand how a watershed transform works, imagine that the catchment basins are dry. If rain were to fall evenly across the image, the basins would fill up at the same rate. Eventually, the water in the basins represented by the circle and square would merge, forming one lake. To prevent the two lakes from becoming one, the watershed transform algorithm builds a dam, or watershed line, where the waters would begin to mix.

Figure A shows the same distance map as Figure B with a line through the bottom two objects. Figure B shows the intensities of the pixels along the line in figure A. Notice the watershed line preventing the waters from the two catchment basins from mixing.

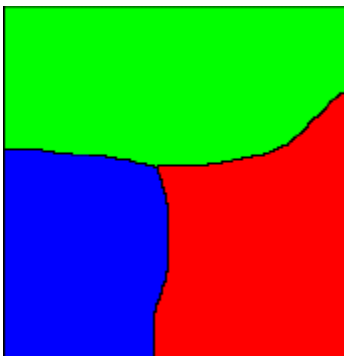


A



B

As the rainfall continues, the rising water in all three lakes would begin to flood the land. The watershed transform algorithm builds dams on the land to prevent the flood waters from each lake from merging. The following figure shows the watershed transform image after segmentation is complete. The water from each catchment basin is represented by a different pixel value. The black lines represent the watershed lines.



In-Depth Discussion

Vincent and Soille's Algorithm

The Vincent and Soille's algorithm fills catchment basins from the bottom up. Imagine that a hole is located in each local minimum. When the topographic surface is immersed in water, water starts filling all the catchment basins, minima of which are under the water level. If two catchment basins are about to merge as a result of further immersion, the algorithm builds a vertical dam up to the highest surface altitude. The dam represents the watershed line. The core algorithm of the Vision watershed transform function is based on Vincent and Soille's algorithm. The concept behind the Vision implementation of Vincent and Soille's algorithm is to sort the pixels in decreasing order of their grayscale values, followed by a flooding step consisting of a

fast breadth-first scanning of all pixels in the order of their grayscale values.

Related concepts:

- [Thresholding](#)
- [Binary Morphology](#)
- [Advanced Morphology Operations](#)
- [Image Masks](#)

Binary Morphology

This section contains information about element structuring, connectivity, and primary and advanced binary morphology operations.

Introduction

Binary morphological operations extract and alter the structure of particles in a binary image. You can use these operations during your inspection application to improve the information in a binary image before making particle measurements, such as the area, perimeter, and orientation.

A binary image is an image containing particle regions with pixel values of 1 and a background region with pixel values of 0. Binary images are the result of the thresholding process. Because thresholding is a subjective process, the resulting binary image may contain unwanted information, such as noise particles, particles touching the border of images, particles touching each other, and particles with uneven borders. By affecting the shape of particles, morphological functions can remove this unwanted information, thus improving the information in the binary image.

Structuring Elements

Morphological operators that change the shape of particles process a pixel based on its number of neighbors and the values of those neighbors. A neighbor is a pixel whose value affects the values of nearby pixels during certain image processing functions. Morphological transformations use a 2D binary mask called a structuring element to define the size and effect of the neighborhood on each pixel, controlling the effect of

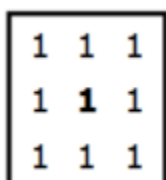
the binary morphological functions on the shape and the boundary of a particle.

When to Use

Use a structuring element when you perform any primary binary morphology operation or the Separation advanced binary morphology operation. You can modify the size and the values of a structuring element to alter the shape of particles in a specific way. However, study the basic morphology operations before defining your own structuring element.

Concepts

The size and contents of a structuring element specify which pixels a morphological operation takes into account when determining the new value of the pixel being processed. A structuring element must have an odd-sized axis to accommodate a center pixel, which is the pixel being processed. The contents of the structuring element are always binary, composed of 1 and 0 values. The most common structuring element is a 3×3 matrix containing values of 1. This matrix, shown below, is the default structuring element for most binary and grayscale morphological transformations.

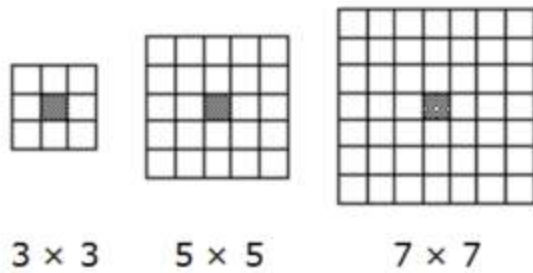


1	1	1
1	1	1
1	1	1

Three factors influence how a structuring element defines which pixels to process during a morphological transformation: the size of the structuring element, the values of the structuring element sectors, and the shape of the pixel frame.

Structuring Element Size

The size of a structuring element determines the size of the neighborhood surrounding the pixel being processed. The coordinates of the pixel being processed are determined as a function of the structuring element. In the following figure, the coordinates of the pixels being processed are (1, 1), (2, 2), and (3, 3), respectively. The origin (0, 0) is always the top, left corner pixel.



Using structuring elements requires an **image border**. A 3×3 structuring element requires a minimum border size of 1. In the same way, structuring elements of 5×5 and 7×7 require a minimum border size of 2 and 3, respectively. Bigger structuring elements require corresponding increases in the image border size.



Note Vision images have a default border size of 3. This border size enables you to use structuring elements as large as 7×7 without any modification. If you plan to use structuring elements larger than 7×7 , specify a correspondingly larger border when creating your image.

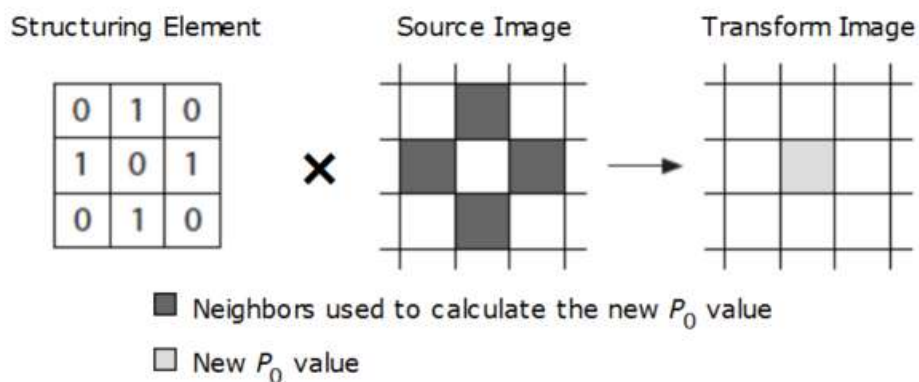
The size of the structuring element determines the speed of the morphological transformation. The smaller the structuring element, the faster the transformation.

Structuring Element Values

The binary values of a structuring element determine which neighborhood pixels to consider during a transformation in the following manner:

- If the value of a structuring element sector is 1, the value of the corresponding source image pixel affects the central pixel's value during a transformation.
- If the value of a structuring element sector is 0, the morphological function disregards the value of the corresponding source image pixel.

The following figure illustrates the effect of structuring element values during a morphological function. A morphological transformation using a structuring element alters a pixel P_0 so that it becomes a function of its neighboring pixel values.



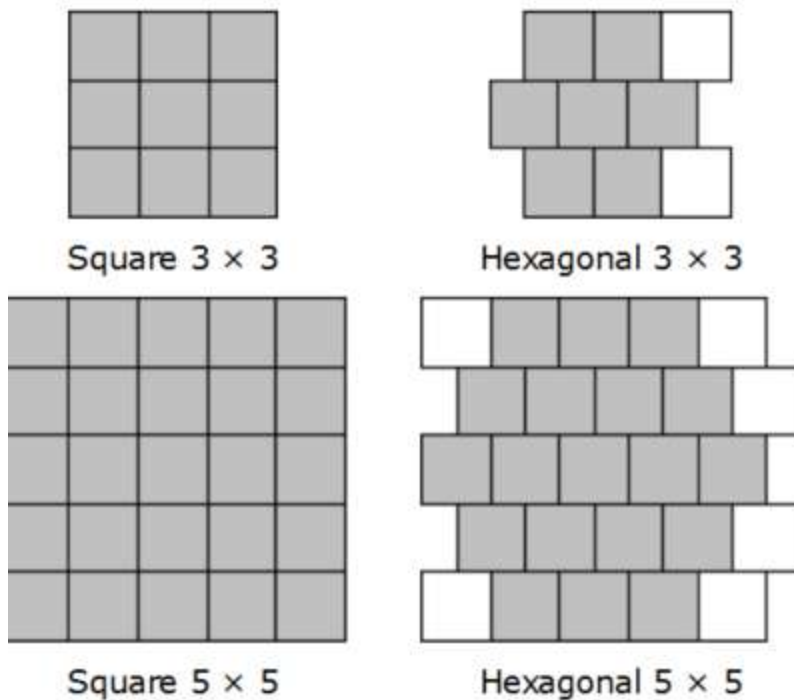
Pixel Frame Shape

A digital image is a 2D array of pixels arranged in a rectangular grid. Morphological transformations that extract and alter the structure of particles allow you to process pixels in either a square or hexagonal configuration. These pixel configurations introduce the concept of a pixel frame. Pixel frames can either be aligned (square) or shifted (hexagonal). The pixel frame parameter is important for functions that alter the value of pixels according to the intensity values of their neighbors. Your decision to use a square or hexagonal frame affects how Vision analyzes the image when you process it with functions that use this frame concept. Vision uses the square frame by default.



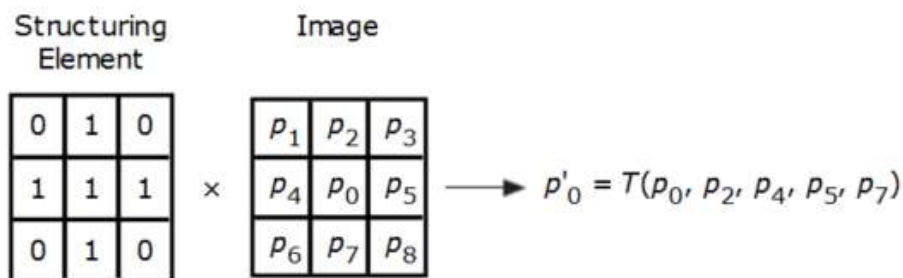
Note Pixels in the image do not physically shift in a horizontal pixel frame. Functions that allow you to set the pixel frame shape merely process the pixel values differently when you specify a hexagonal frame.

The following figure illustrates the difference between a square and hexagonal pixel frame when a 3×3 and a 5×5 structuring element are applied.

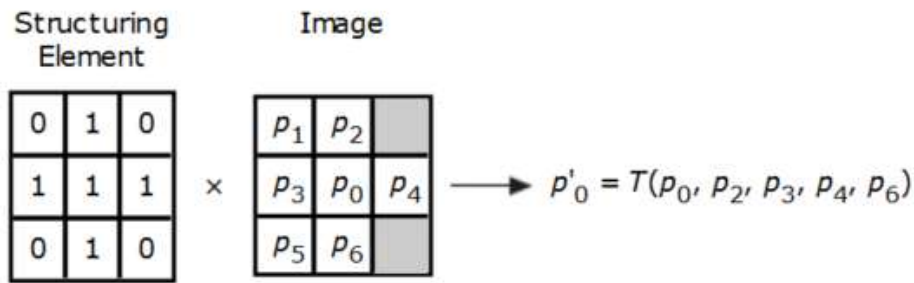


If a morphological function uses a 3×3 structuring element and a hexagonal frame mode, the transformation does not consider the elements $[2, 0]$ and $[2, 2]$ when calculating the effect of the neighbors on the pixel being processed. If a morphological function uses a 5×5 structuring element and a hexagonal frame mode, the transformation does not consider the elements $[0, 0]$, $[4, 0]$, $[4, 1]$, $[4, 3]$, $[0, 4]$, and $[4, 4]$.



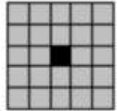
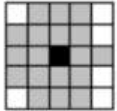
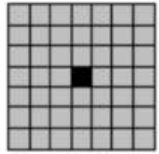
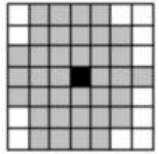
The following figure illustrates a morphological transformation using a 3×3 structuring element and a rectangular frame mode.



The following figure illustrates a morphological transformation using a 3×3 structuring element and a hexagonal frame mode.



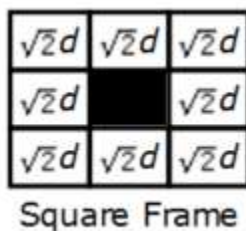
The following table illustrates the effect of the pixel frame shape on a neighborhood given three structuring element sizes. The gray boxes indicate the neighbors of each black center pixel.

Structuring Element Size	Square Pixel Frame	Hexagonal Pixel Frame
3×3		
5×5		
7×7		

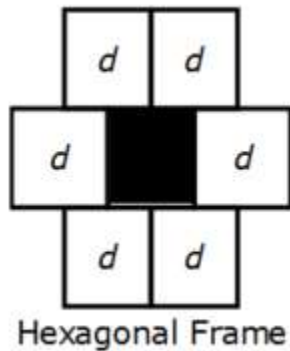
Square Frame

In a square frame, pixels line up normally. The following figure shows a pixel in a square frame surrounded by its eight neighbors. If d is the distance from the vertical and horizontal neighbors to the central pixel, then the diagonal neighbors are located at a distance of

$\sqrt{2}d$
from the central pixel.



In a hexagonal frame, the even lines of an image shift half a pixel to the right. Therefore, the hexagonal frame places the pixels in a configuration similar to a true circle. The following figure shows a pixel in a hexagonal frame surrounded by its six neighbors. Each neighbor is an equal distance d from the central pixel, which results in highly precise morphological measurements.



Related concepts:

- [Image Borders](#)

Connectivity

After you identify the pixels belonging to a specified intensity threshold, Vision groups them into particles. This grouping process introduces the concept of connectivity. You can set the pixel connectivity in some functions to specify how Vision determines whether two adjoining pixels are included in the same particle.

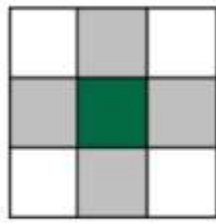
When to Use

Use connectivity-4 when you want Vision to consider pixels to be part of the same particle only when the pixels touch along an adjacent edge. Use connectivity-8 when you want Vision to consider pixels to be part of the same particle even if the pixels touch only at a corner.

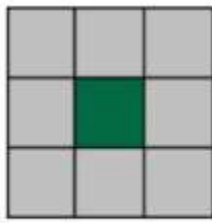
Concepts

With connectivity-4, two pixels are considered part of the same particle if they are horizontally or vertically adjacent. With connectivity-8, two pixels are considered part of the same particle if they are horizontally, vertically, or diagonally adjacent. The

following figure illustrates the two types of connectivity.

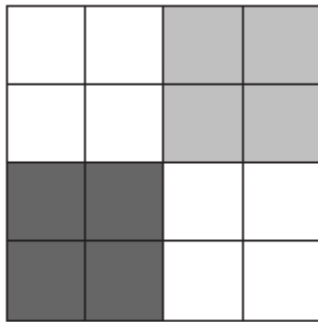


Connectivity-4

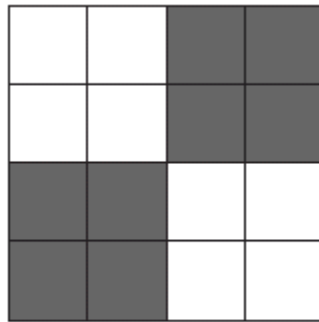


Connectivity-8

The following figure illustrates how connectivity-4 and connectivity-8 affect the way the number of particles in an image are determined. In figure A, the image has two particles with connectivity-4. In figure B, the same image has one particle with connectivity-8.



A



B

In-Depth Discussion

In a rectangular pixel frame, each pixel P_0 has eight neighbors, as shown in the following graphic. From a mathematical point of view, the pixels P_1 , P_3 , P_5 , and P_7 are closer to P_0 than the pixels P_2 , P_4 , P_6 , and P_8 .

$$\begin{bmatrix} P_8 & P_1 & P_2 \\ P_7 & P_0 & P_3 \\ P_6 & P_5 & P_4 \end{bmatrix}$$

If D is the distance from P_0 to P_1 , then the distances between P_0 and its eight neighbors can range from D to

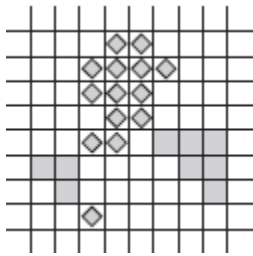
$\sqrt{2}$

D, as shown in the following figure.

$$\begin{bmatrix} \sqrt{2}D & D & \sqrt{2}D \\ D & 0 & D \\ \sqrt{2}D & D & \sqrt{2}D \end{bmatrix}$$

Connectivity-4

A pixel belongs to a particle if it is located a distance of D from another pixel in the particle. In other words, two pixels are considered to be part of the same particle if they are horizontally or vertically adjacent. They are considered as part of two different particles if they are diagonally adjacent. In the following figure, the particle count equals 4.

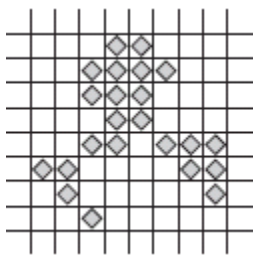


Connectivity-8

A pixel belongs to a particle if it is located a distance of D or

 $\sqrt{2}$

D from another pixel in the particle. In other words, two pixels are considered to be part of the same particle if they are horizontally, vertically, or diagonally adjacent. In the following figure, the particle count equals 1.



Primary Morphology Operations

Primary morphological operations work on binary images to process each pixel based on its neighborhood. Each pixel is set either to 1 or 0, depending on its neighborhood information and the operation used. These operations always change the overall size and shape of particles in the image.

When to Use

Use the primary morphological operations for expanding or reducing particles, smoothing the borders of objects, finding the external and internal boundaries of particles, and locating particular configurations of pixels.

You also can use these transformations to prepare particles for quantitative analysis, to observe the geometry of regions, and to extract the simplest forms for modeling and identification purposes.

Concepts

The primary morphology functions apply to binary images in which particles have been set to 1 and the background is equal to 0. They include three fundamental binary processing functions: erosion, dilation, and hit-miss. The other transformations are combinations of these three functions.

This section describes the following primary morphology transformations:

- Erosion
- Dilation
- Opening
- Closing
- Inner gradient
- Outer gradient
- Hit-miss
- Thinning
- Thickening
- Proper-opening
- Proper-closing
- Auto-median



Note In the following descriptions, the term pixel denotes a pixel equal to 1, and the term particle denotes a group of pixels equal to 1.

Erosion and Dilation Functions

An erosion eliminates pixels isolated in the background and erodes the contour of particles according to the template defined by the structuring element.

For a given pixel P_0 , the structuring element is centered on P_0 . The pixels masked by a coefficient of the structuring element equal to 1 are then referred as P_i .

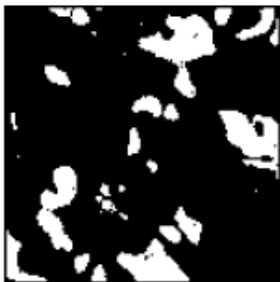
- If the value of one pixel P_i is equal to 0, then P_0 is set to 0, else P_0 is set to 1.
- If $\text{AND}(P_i) = 1$, then $P_0 = 1$, else $P_0 = 0$.

A dilation eliminates tiny holes isolated in particles and expands the particle contours according to the template defined by the structuring element. This function has the opposite effect of an erosion because the dilation is equivalent to eroding the background.

For any given pixel P_0 , the structuring element is centered on P_0 . The pixels masked by a coefficient of the structuring element equal to 1 then are referred to as P_i .

- If the value of one pixel P_i is equal to 1, then P_0 is set to 1, else P_0 is set to 0.
- If $\text{OR}(P_i) = 1$, then $P_0 = 1$, else $P_0 = 0$.

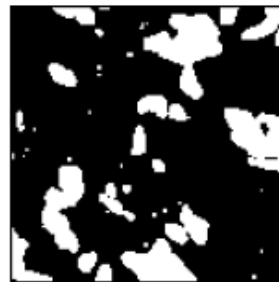
The following figure illustrates the effects of erosion and dilation. Figure A is the binary source image. Figure B represents the source image after erosion, and figure C shows the source image after dilation.



A

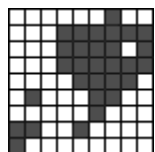


B



C

The following figure is the source image for the examples in the following tables, in which gray cells indicate pixels equal to 1.



The following tables show how the structuring element can control the effects of erosion or dilation, respectively. The larger the structuring element, the more templates can be edited and the more selective the effect.

Structuring Element	After Erosion	Description
		<p>A pixel is cleared if it is equal to 1 and if its three upper-left neighbors do not equal 1.</p> <p>The erosion truncates the upper-left particle borders.</p>
		<p>A pixel is cleared if it is equal to 1 and if its lower and right neighbors do not equal 1.</p> <p>The erosion truncates the bottom and right particle borders but retains the corners.</p>

Structuring Element	After Erosion	Description
		<p>A pixel is set to 1 if it is equal to 1 or if one of its three upper-left neighbors equals 1.</p> <p>The dilation expands the lower-right particle borders.</p>
		<p>A pixel is set to 1 if it is equal to 1 or if its lower or right neighbor equals 1.</p>

Structuring Element	After Erosion	Description
		The dilation expands the upper and left particle borders.

Opening and Closing Functions



The opening function is an erosion followed by a dilation. This function removes small particles and smooths boundaries. This operation does not significantly alter the area and shape of particles because erosion and dilation are dual transformations, in which borders removed by the erosion function are restored during dilation. However, small particles eliminated during the erosion are not restored by the dilation. If I is an image,



$\text{opening}(I) = \text{dilation}(\text{erosion}(I)).$

The closing function is a dilation followed by an erosion. This function fills tiny holes and smooths boundaries. This operation does not significantly alter the area and shape of particles because dilation and erosion are morphological complements, where borders expanded by the dilation function are then reduced by the erosion function. However, erosion does not restore any tiny holes filled during dilation. If I is an image,

$\text{closing}(I) = \text{erosion}(\text{dilation}(I)).$

The following figures illustrate examples of the opening and closing function.

	<table><tr><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	1	1	1	1	1	1	1	1	1		
1	1	1										
1	1	1										
1	1	1										
Original Image	Structuring Element	After Opening	After Closing									

<table><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr></table>	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		<table><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr></table>	0	0	1	0	0	0	1	1	1	0	1	1	1	1	1	0	1	1	1	0	0	0	1	0	0	
1	1	1	1	1																																																	
1	1	1	1	1																																																	
1	1	1	1	1																																																	
1	1	1	1	1																																																	
1	1	1	1	1																																																	
0	0	1	0	0																																																	
0	1	1	1	0																																																	
1	1	1	1	1																																																	
0	1	1	1	0																																																	
0	0	1	0	0																																																	
Structuring Element	After Opening	Structuring Element	After Closing																																																		

Inner Gradient Function

The internal edge subtracts the eroded image from its source image. The remaining pixels correspond to the pixels eliminated by the erosion process. If I is an image,

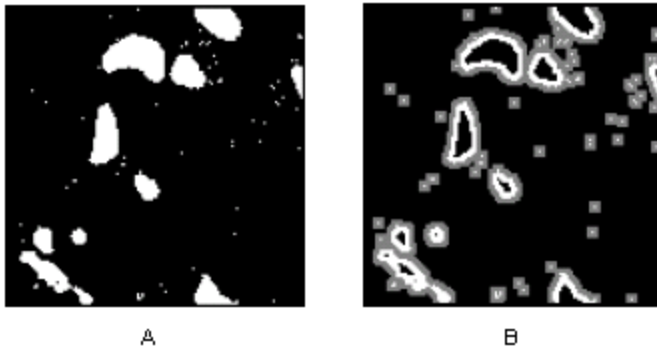
$$\text{internal edge}(I) = I - \text{erosion}(I) = \text{XOR}(I, \text{erosion}(I)).$$

Outer Gradient Function

The external edge subtracts the source image from the dilated image of the source image. The remaining pixels correspond to the pixels added by the dilation process. If I is an image,

$$\text{internal edge}(I) = \text{dilation}(I) - I = \text{XOR}(I, \text{dilation}(I)).$$

Figure A shows the binary source image. Figure B shows the image produced from an extraction using a 5×5 structuring element. The superimposition of the internal edge is shown in white, and the external edge is shown in gray. The thickness of the extended contours depends on the size of the structuring element.



Hit-Miss Function

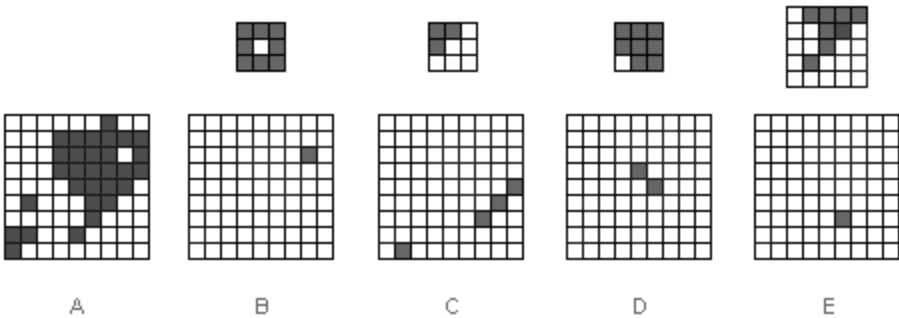
The hit-miss function locates particular configurations of pixels. This function extracts each pixel located in a neighborhood exactly matching the template defined by the structuring element. Depending on the configuration of the structuring element, the hit-miss function can locate single isolated pixels, cross-shape or longitudinal patterns, right angles along the edges of particles, and other user-specified shapes. The larger the size of the structuring element, the more specific the researched template can be. Refer to the following table for strategies on using the hit-miss function.

In a structuring element with a central coefficient equal to 0, a hit-miss function changes all pixels set to 1 in the source image to the value 0.

For a given pixel P_0 , the structuring element is centered on P_0 . The pixels masked by the structuring element are then referred to as P_i .

- If the value of each pixel P_i is equal to the coefficient of the structuring element placed on top of it, then the pixel P_0 is set to 1, else the pixel P_0 is set to 0.
- In other words, if the pixels P_i define the exact same template as the structuring element, then $P_0 = 1$, else $P_0 = 0$.

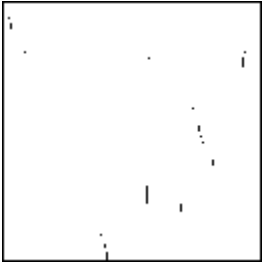
Figures B, C, D, and E show the result of three hit-miss functions applied to the same source image, represented in figure A. Each hit-miss function uses a different structuring element, which is specified above each transformed image. Gray cells indicate pixels equal to 1.



A second example of the hit-miss function shows how, when given the binary image shown in the following figure, the function can locate various patterns specified in the structuring element. The results are displayed in the following table.



Strategy	Structuring Element	Resulting Image																									
<p>Use the hit-miss function to locate pixels isolated in a background.</p> <p>The structuring element on the right extracts all pixels equal to 1 that are surrounded by at least two layers of pixels that are equal to 0.</p>	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0																							
0	0	0	0	0																							
0	0	1	0	0																							
0	0	0	0	0																							
0	0	0	0	0																							
<p>Use the hit-miss function to locate single pixel holes in particles.</p> <p>The structuring element on the right extracts all pixels equal to 0 that are surrounded by at least one layer of pixels that are equal to 1.</p>	<table><tr><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	1	1	1	1	0	1	1	1	1																	
1	1	1																									
1	0	1																									
1	1	1																									

Strategy	Structuring Element	Resulting Image									
<p>Use the hit-miss function to locate pixels along a vertical left edge.</p> <p>The structuring element on the right extracts pixels surrounded by at least one layer of pixels equal to 1 to the left and pixels that are equal to 0 to the right.</p>	<table> <tr> <td>1</td><td>1</td><td>0</td></tr> <tr> <td>1</td><td>1</td><td>0</td></tr> <tr> <td>1</td><td>1</td><td>0</td></tr> </table>	1	1	0	1	1	0	1	1	0	
1	1	0									
1	1	0									
1	1	0									

Thinning Function

The thinning function eliminates pixels that are located in a neighborhood matching a template specified by the structuring element. Depending on the configuration of the structuring element, you also can use thinning to remove single pixels isolated in the background and right angles along the edges of particles. A larger structuring element allows for a more specific template.

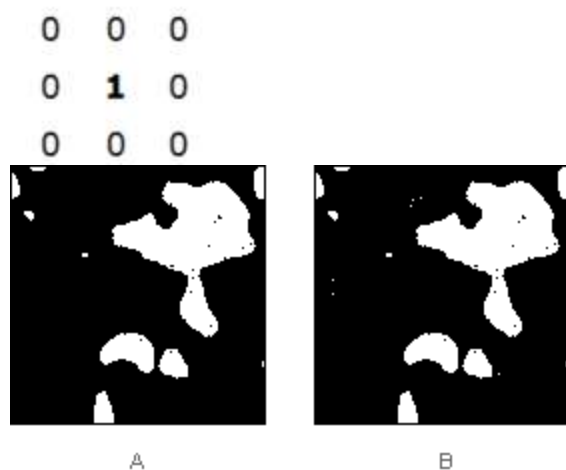
The thinning function extracts the intersection between a source image and its transformed image after a hit-miss function. In binary terms, the operation subtracts its hit-miss transformation from a source image.

Do not use this function when the central coefficient of the structuring element is equal to 0. In such cases, the hit-miss function can change only the value of certain pixels in the background from 0 to 1. However, the subtraction of the thinning function then resets these pixels back to 0.

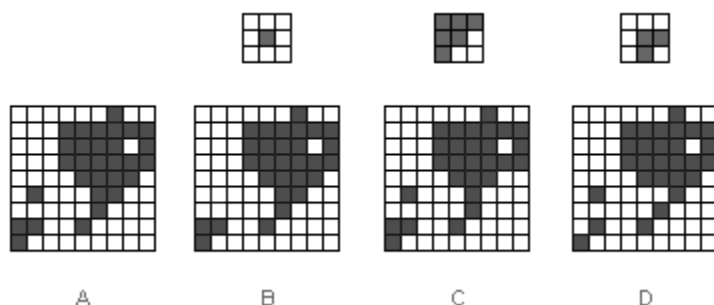
If I is an image,

$$\text{thinning}(I) = I - \text{hit-miss}(I) = \text{XOR}(I, \text{hit-miss}(I)).$$

Figure A shows the binary source image used in the following example of thinning. Figure B illustrates the resulting image, in which single pixels in the background are removed from the image. This example uses the following structuring element:



Another thinning example uses the source image shown in figure A. Figures B, C, and D show the results of three thinnings applied to the source image. Each thinning uses a different structuring element, which is specified above each transformed image. Gray cells indicate pixels equal to 1.



Thickening Function

The thickening function adds to an image those pixels located in a neighborhood that matches a template specified by the structuring element. Depending on the configuration of the structuring element, you can use thickening to fill holes and smooth right angles along the edges of particles. A larger structuring element allows for a more specific template.

The thickening function extracts the union between a source image and its transformed image, which was created by a hit-miss function using a structuring element specified for thickening. In binary terms, the operation adds a hit-miss transformation to a source image.

Do not use this function when the central coefficient of the structuring element is

equal to 1. In such cases, the hit-miss function can turn only certain particle pixels from 1 to 0. However, the addition of the thickening function resets these pixels to 1.

If I is an image,

$\text{thickening}(I) = I + \text{hit-miss}(I) = \text{OR}(I, \text{hit-miss}(I))$.

Figure A represents the binary source file used in the following thickening example. Figure B shows the result of the thickening function applied to the source image, which filled single pixel holes using the following structuring element:

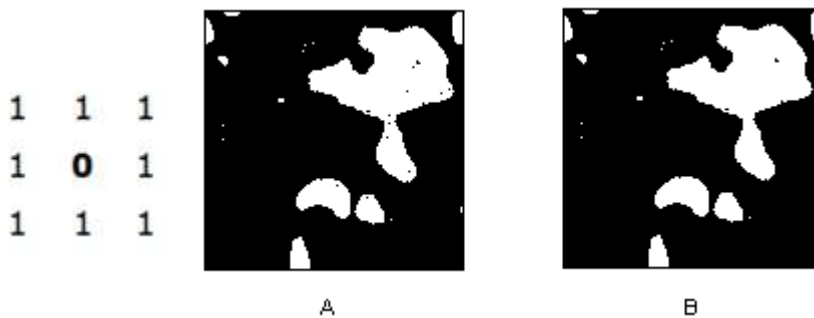
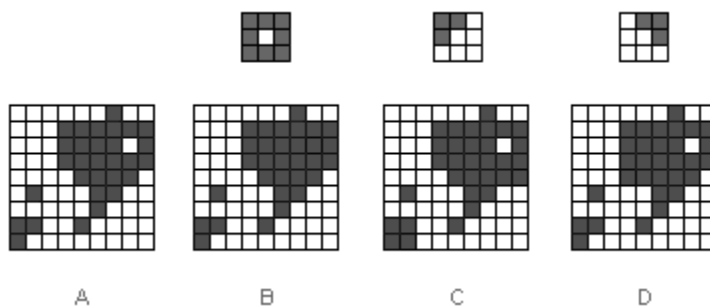


Figure A represents the source image for another thickening example. Figures B, C, and D show the results of three thickenings as applied to the source image. Each thickening uses a different structuring element, which is specified on top of each transformed image. Gray cells indicate pixels equal to 1.



Proper-Opening Function

The proper-opening function is a finite and dual combination of openings and closings. It removes small particles and smooths the contour of particles according to the template defined by the structuring element.

If I is the source image, the proper-opening function extracts the intersection between

the source image I and its transformed image obtained after an opening, followed by a closing, and then followed by another opening:

$$\text{proper-opening}(I) = \text{AND}(I, \text{OCO}(I))$$

or

$$\text{proper-opening}(I) = \text{AND}(I, \text{DEEDDE}(I))$$

where:

- I is the source image,
- E is an erosion,
- D is a dilation,
- O is an opening,
- C is a closing,
- $F(I)$ is the image obtained after applying the function F to the image I ,
- $GF(I)$ is the image obtained after applying the function F to the image I followed by the function G to the image I .

Proper-Closing Function

The proper-closing function is a finite and dual combination of closings and openings. It fills tiny holes and smooths the inner contour of particles according to the template defined by the structuring element.

If I is the source image, the proper-closing function extracts the union of the source image I and its transformed image obtained after a closing, followed by an opening, and then followed by another closing:

$$\text{proper-closing}(I) = \text{OR}(I, \text{COC}(I))$$

or

$$\text{proper-closing}(I) = \text{OR}(I, \text{EDDEED}(I))$$

where:

- I is the source image,

- E is an erosion,
- D is a dilation,
- O is an opening,
- C is a closing,
- $F(I)$ is the image obtained after applying the function F to the image I,
- $GF(I)$ is the image obtained after applying the function F to the image I followed by the function G to the image I.

Auto-Median Function

The auto-median function is a dual combination of openings and closings. It generates simpler particles that contain fewer details.

If I is the source image, the auto-median function extracts the intersection between the proper-opening and proper-closing of the source image I:

$$\text{auto-median}(I) = \text{AND}(\text{OCO}(I), \text{COC}(I))$$

or

$$\text{auto-median}(I) = \text{AND}((\text{DEEDDEI}), \text{EDDEED}(I))$$

where:

- I is the source image,
- E is an erosion,
- D is a dilation,
- O is an opening,
- C is a closing,
- $F(I)$ is the image obtained after applying the function F to the image I,
- $GF(I)$ is the image obtained after applying the function F to the image I followed by the function G to the image I.

Advanced Morphology Operations

The advanced morphology operations are built upon the primary morphological operators and work on particles as opposed to pixels. Each of the operations have been developed to perform specific operations on the particles in a binary image.

When to Use

Use the advanced morphological operations to fill holes in particles, remove particles that touch the border of the image, remove unwanted small and large particles, separate touching particles, find the convex hull of particles, and more.

You can use these transformations to prepare particles for quantitative analysis, observe the geometry of regions, extract the simplest forms for modeling and identification purposes, and so forth.

Concepts

The advanced morphology functions are conditional combinations of fundamental transformations, such as binary erosion and dilation. The functions apply to binary images in which a threshold of 1 has been applied to particles and where the background is equal to 0.



Note In this section of the manual, the term pixel denotes a pixel equal to 1, and the term particle denotes a group of pixels equal to 1.

Border Function

The ***border function*** removes particles that touch the border of the image. These particles may have been truncated during the digitization of the image, and their elimination helps to avoid erroneous particle measurements and statistics.

Hole Filling Function

The ***hole filling function*** fills the holes within particles.

Labeling Function

The ***labeling function*** assigns a different gray-level value to each particle. The image produced is not a binary image, but a labeled image using a number of gray-level values equal to the number of particles in the image plus the gray level 0 used in the background area.

The labeling function identifies particles using either connectivity-4 or connectivity-8 criteria. For more information on connectivity, refer to the **connectivity** section.

Lowpass and Highpass Filters

The lowpass filter removes small particles according to their widths as specified by a parameter called filter size. For a given filter size N , the lowpass filter eliminates particles whose widths are less than or equal to $(N - 1)$ pixels. These particles disappear after $(N - 1) / 2$ erosions.

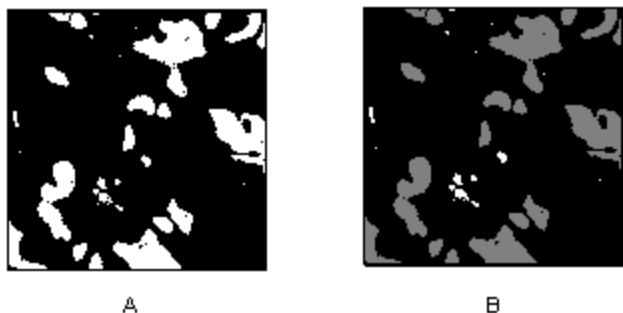
The highpass filter removes large particles according to their widths as specified by a parameter called filter size. For a given filter size N , the highpass filter eliminates particles with widths greater than or equal to N pixels. These particles do not disappear after $(N / 2 + 1)$ erosions.

Both the highpass and lowpass morphological filters use erosions to select particles for removal. Since erosions or filters cannot discriminate particles with widths of $2k$ pixels from particles with widths of $2k - 1$ pixels, a single erosion eliminates both particles that are 2 pixels wide and 1 pixel wide.

The following table shows the effect of lowpass and highpass filtering.

Filter Size (N)	Highpass Filter	Lowpass Filter
N is an even number ($N = 2k$)	<ul style="list-style-type: none"> Removes particles with a width greater than or equal to $2k$ Uses $k - 1$ erosions 	<ul style="list-style-type: none"> Removes particles with a width less than or equal to $2k - 1$ Uses $k - 1$ erosions
N is an odd number ($N = 2k + 1$)	<ul style="list-style-type: none"> Removes particles with a width greater than or equal to $2k + 1$ Uses k erosions 	<ul style="list-style-type: none"> Removes particles with a width less than or equal to $2k$ Uses k erosions

Figure A represents the binary source image used in this example. Figure B shows how, for a given filter size, a highpass filter produces the following image. Gray particles and white particles are filtered out by a lowpass and highpass filter, respectively.



Separation Function

The separation function breaks narrow isthmuses and separates touching particles with respect to a user-specified filter size. This operation uses erosions, labeling, and conditional dilations.

For example, after thresholding an image, two gray-level particles overlapping one another might appear as a single binary particle. You can observe narrowing where the original particles have intersected. If the narrowing has a width of M pixels, a separation using a filter size of $(M + 1)$ breaks it and restores the two original particles. This applies to all particles that contain a narrowing shorter than N pixels.

For a given filter size N , the separation function segments particles with a narrowing shorter than or equal to $(N - 1)$ pixels. These particles are divided into two parts after $(N - 1) / 2$ erosions.

The above definition is true when N is an odd number, but should be modified slightly when N is an even number, due to the use of erosions in determining whether a narrowing should be broken or kept. The function cannot discriminate a narrowing with a width of $2k$ pixels from a narrowing with a width of $(2k - 1)$ pixels, therefore, one erosion breaks both a narrowing that is two pixels wide as well as a narrowing that is one pixel wide.

The precision of the separation is limited to the elimination of constrictions that have a width smaller than an even number of pixels:

- If N is an even number ($2k$), the separation breaks a narrowing with a width smaller than or equal to $(2k - 2)$ pixels. It uses $(k - 1)$ erosions.
- If N is an odd number ($2k + 1$), the separation breaks a narrowing with a width smaller than or equal to $2k$. It uses k erosions.

Skeleton Functions

A skeleton function applies a succession of thinnings until the width of each particle becomes equal to one pixel. The skeleton functions are both time- and memory-consuming. They are based on conditional applications of thinnings and openings that use various configurations of structuring elements.

L-Skeleton uses the following type of structuring element:

```

0  ?  1
0  1  1
0  ?  1

```

M-Skeleton uses the following type of structuring element:

```

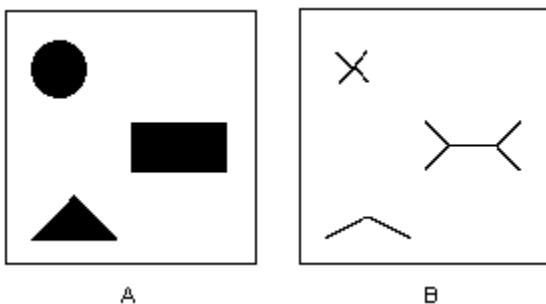
?  ?  1
0  1  1
?  ?  17

```

Skiz is an L-Skeleton performed on an inverse of the image.

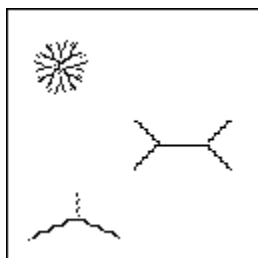
L-Skeleton Function

The L-skeleton function indicates the L-shaped structuring element skeleton function. Using the source image in figure A, the L-skeleton function produces the image in figure B.



M-Skeleton Function

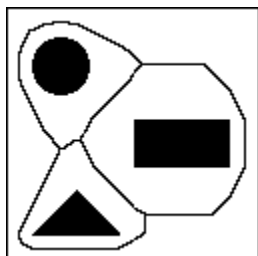
The M-skeleton function extracts a skeleton with more dendrites or branches. Using the source image from figure A, the M-skeleton function produces the image shown in the following figure.



Skiz Function

The skiz (skeleton of influence zones) function behaves like an L-skeleton function applied to the background regions instead of the particle regions. It produces median lines that are at an equal distance from the particles.

Using the source image from figure A, the skiz function produces the image in the following figure, which is shown superimposed on the source image.



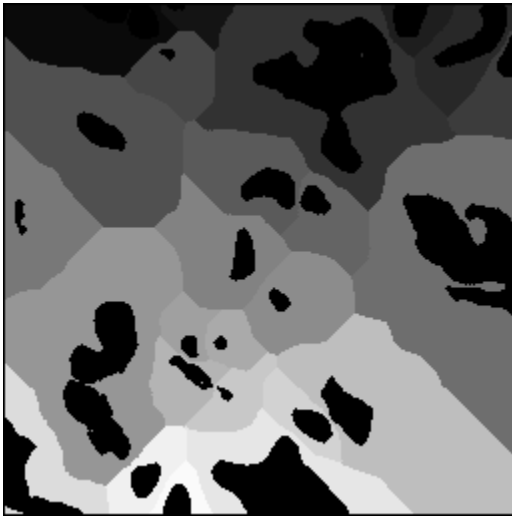
Segmentation Function

The segmentation function is applied only to labeled images. It partitions an image into segments that are centered around a particle such that they do not overlap or leave empty zones. Empty zones are caused by dilating particles until they touch one another.



Note The segmentation function is time-consuming. Reduce the image to its minimum significant size before selecting this function.

In the following figure, binary particles, which are shown in black, are superimposed on top of the segments, which are shown in gray shades.



When applied to an image with binary particles, the transformed image turns red because it is entirely composed of pixels set to 1.

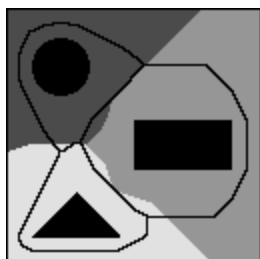
Comparisons Between Segmentation and Skiz Functions

The segmentation function extracts segments that contain only one particle. A segment represents the area in which this particle can be moved without intercepting another particle, assuming that all particles move at the same speed.

The edges of these segments give a representation of the external skeletons of the particles. Unlike the skiz function, segmentation does not involve median distances.

You can obtain segments using successive dilations of particles until they touch each other and cover the entire image. The final image contains as many segments as there were particles in the original image. However, if you consider the inside of closed skiz lines as segments, you may produce more segments than particles originally present in the image. Consider the upper-right region in the following figure. This image shows the following features:

- original particles in black,
- segments in shades of gray,
- skiz lines.



Distance Function

The distance function assigns a gray-level value to each pixel equal to the shortest distance to the particle border. This distance may be equal to the distance to the outer particle border or to a hole within the particle.



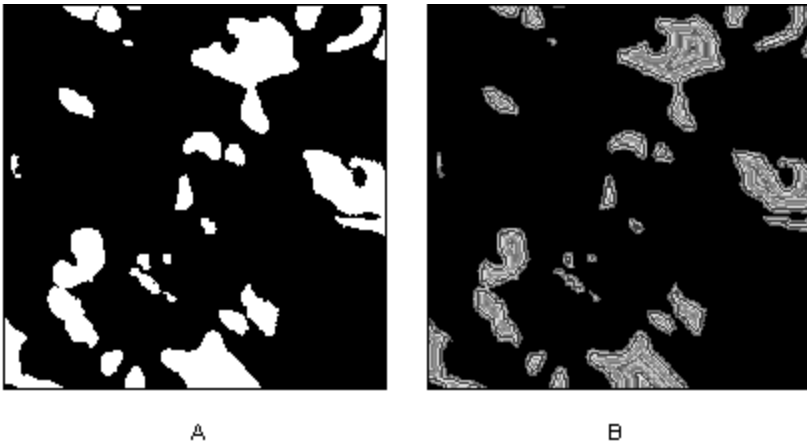
Tip Use the Danielsson function instead of the distance function when possible.

Danielsson Function

The Danielsson function also creates a distance map but is a more accurate algorithm than the classical distance function. Because the destination image is 8-bit, its pixels cannot have a value greater than 255. Any pixels with a distance to the edge greater than 255 are set to 255.

For example, a circle with a radius of 300 yields 255 concentric rings whose pixel values range from 1 to 255 from the perimeter of the circle inward. The rest of the circle is filled with a solid circle whose pixel value is 255 and radius is 45.

Figure A shows the source threshold image used in the following example. The image is sequentially processed with a lowpass filter, hole filling, and the Danielsson function. The Danielsson function produces the distance map image shown in figure B.

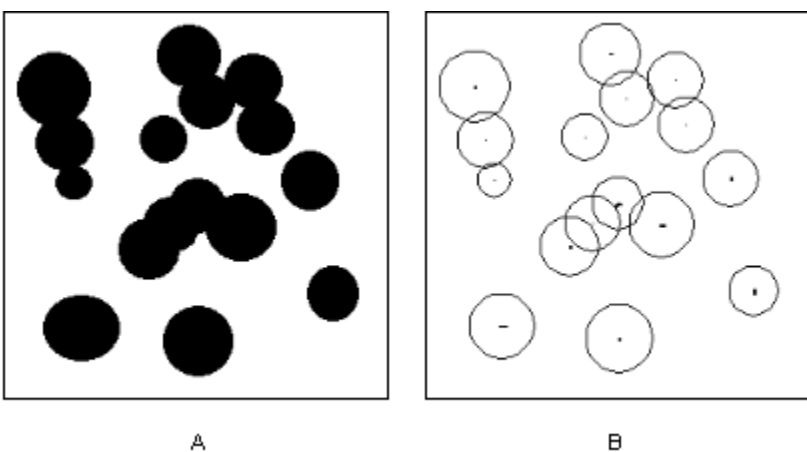


View the resulting image with a binary palette. In this palette, each level corresponds to a different color. Thus, you easily can determine the relation of a set of pixels to the border of a particle. The first layer, which forms the border, is red. The second layer, closest to the border, is green, the third layer is blue, and so forth.

Circle Function

The circle function separates overlapping circular particles using the Danielsson coefficient to reconstitute the form of a particle, provided that the particles are essentially circular. The particles are treated as a set of overlapping discs that are then separated into separate discs. This allows you to trace circles corresponding to each particle.

Illustration A shows the source image for the following example. Figure B shows the image after the circle function is applied to the image.

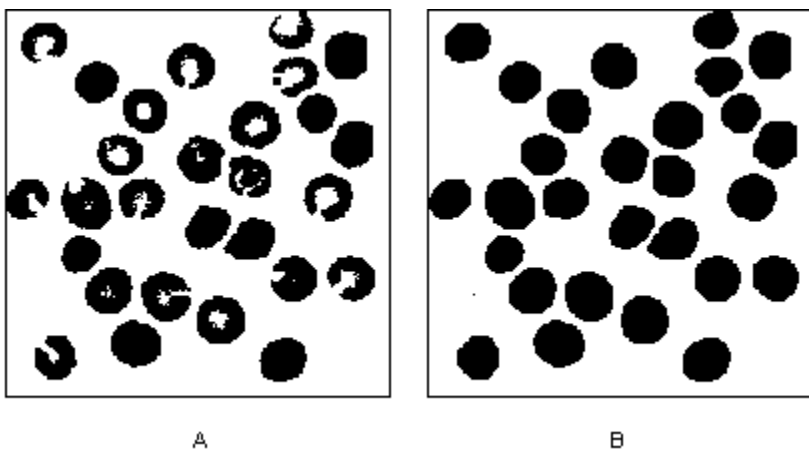


Convex Hull Function

The convex hull function is useful for closing particles so that measurements can be made on the particle, even when the particle contour is discontinuous.

The convex hull function calculates a convex envelope around each particle, effectively closing the particle. The image to which you apply a convex hull function must be binary.

Figure A shows the original labeled image used in this example. Figure B shows the results after the convex hull function is applied to the image.



Related concepts:

- [Connectivity](#)

Morphological Reconstruction

Morphological reconstruction is useful for constructing an image from small components or for removing features from an image, without altering the shape of the objects in the image. Morphological reconstruction works on grayscale images and binary images. Use morphological reconstruction for applications such as:

- Segmenting magnetic resonance images (MRI) of structures inside the body.
- H-dome extraction for detecting clustered microcalcifications in digital mammograms.
- Removing shadows from images.
- Identifying language scripts.

- Finding the connected paths in a network or map.

Concepts

The morphological reconstruction process is based on a source image, a marker image, and marker points.

- **Source Image**—The source image, which in some research papers is referred to as the mask image, is the reference image used in the morphological reconstruction.
- **Marker Image**—The reconstruction process occurs on the marker image, which is created by applying **dilations** or **erosions** on the source image. The marker image can also be taken from existing images. The marker image should have the same dimensions as the source image.
- **Marker Points**—Marker points are user-specified points in the image that specify where the reconstruction process should start.

Reconstruction by Dilation

Reconstruction by dilation reconstructs bright regions in grayscale images and reconstructs particles in binary images. Starting at the marker points, neighboring pixels are reconstructed by spreading the brightness value. Reconstruction by dilation starts with the maximal gray valued pixels of the marker and reconstructs the neighboring pixels ranging from 0 to the maximal valued pixel. Refer to **Primary Morphology Operations** for more information about dilation.

Reconstruction by Erosion

Reconstruction by erosion reconstructs dark regions in a grayscale image and holes in a binary image. Starting at the marker points, neighboring pixels are reconstructed by spreading the darkness value. Reconstruction by erosion starts with the minimal valued pixels of the marker and reconstructs the neighboring pixels ranging from the minimal valued pixel to the image maximum value (for example, the image maximum value is 255 for U8 images). Refer to **Primary Morphology Operations** for more information about erosion.

Connectivity

Grayscale morphological reconstruction uses a **structuring element** to determine

the type of connectivity. In general, a 3 x 3 structuring element is used to specify the connectivity. Higher order kernels are supported, but grayscale morphological reconstruction is optimized for the 3 x 3 connectivity-4 and connectivity-8 kernels.

Binary morphological reconstruction supports ***connectivity-4*** and ***connectivity-8***. Image border calculations are handled internally, unless the marker image and the destination image are same size. In this case, the image should have a minimum border size of half the kernel size.

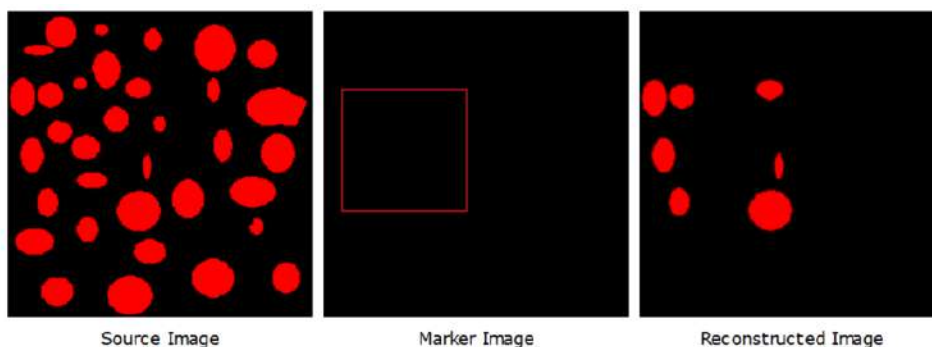
Reconstruction with ROI

You can limit morphological reconstruction to the area bounded by an ROI. The image reconstruction happens inside the ROI of the marker image and the parts of the image outside of the ROI will remain unchanged. If marker points are used instead of a marker image, the points inside the ROI are used to reconstruct the image.

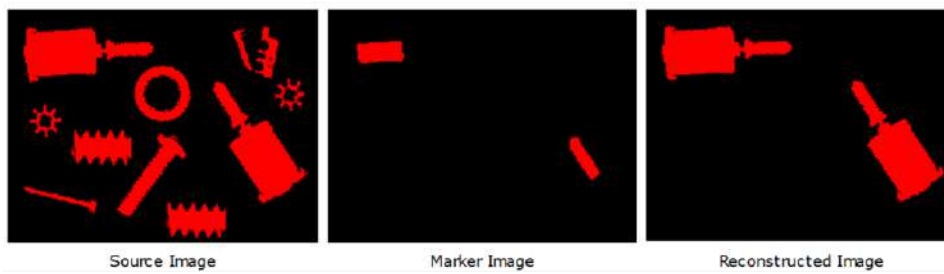
Binary Morphological Reconstruction

Use binary morphological reconstruction to find connected particles or holes in a binary image. It can also be used to extract the particles or holes which are connected a set of pixels in a marker image.

The following example extracts the particles connected to a rectangle boundary.



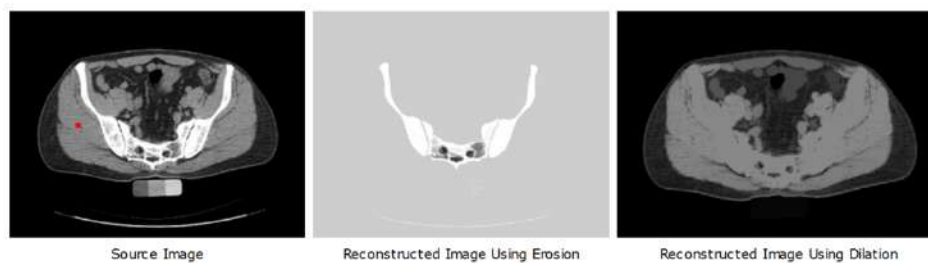
The following example extracts the particles based on morphological properties.



Grayscale Morphological Reconstruction

Use grayscale morphological reconstruction to segment an image based on its grayscale values. This type of reconstruction is useful for counting the number of objects in an image, removing shadows, and identifying language scripts.

The following examples segment the image based on its grayscale values. The marker point, shown in red in the source image, is where the segmentation process begins.



Creating Marker Images

The following examples show how to generate a marker image for morphological reconstruction based on morphology and based on H-Dome extraction.

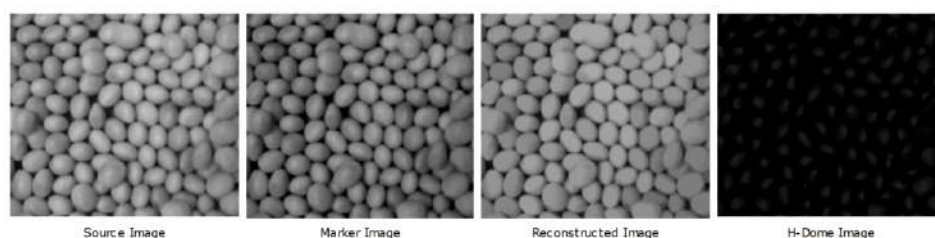
Creating a Marker Image Based on Morphological Properties

When performing reconstruction based on morphological properties, the goal is to extract the particles which are thicker than the specified value. Figure A shows the source image. Apply a **distance transform** on the image. Figure B shows the result of the distance transform of the source image. Threshold the image based on the specified thickness value, eliminating all particles that are narrower than the thickness value. Figure C shows the threshold output, which is used as the marker image.



Creating a Marker Image for H-Dome Extraction

Grayscale morphological reconstruction can be used to count the number of objects in a given image. Create the marker image by subtracting a constant value from the pixels in the source image. The number of H-Domes in the output gives an estimate of the total number of seeds in the source image. In the following example, a constant value of 30 is used. The H-Dome image is created by subtracting the reconstructed image from the source image.



In-Depth Discussion

The function $OP(q)$ forms the basic operation of morphological reconstruction. $OP(q)$ is applied to each pixel during reconstruction. In general, if $OP(q)$ is applied multiple times on all the pixels of an image, the resultant will be the reconstructed image.

Reconstruction by dilation is characterized by the equation:

$$OP(q) = \text{Min}(\text{Max}(M(q), K), I(q))$$

Reconstruction by erosion is characterized by the equation:

$$OP(q) = \text{Max}(\text{Min}(M(q), K), I(q))$$

where:

- M is the marker image,

- K is the kernel,
- I is the source image.

The reconstruction process uses two algorithms: the downhill filter and the hybrid reconstruction algorithm.

Downhill Filter

The downhill filter is used for binary, U8, U16, and I16 images. It operates through a controlled process of region growing by ordered aggregation of surface pixels onto an expanding shell. The maximum value of the marker image is determined and the reconstruction is applied to all neighboring pixels of the maximal valued pixel until all pixels are reconstructed.

Hybrid Algorithm

The hybrid algorithm is used on the SGL grayscale images. Hybrid reconstruction combines a sequential and a queue-based algorithm. The sequential technique applies the reconstruction of pixels by scanning the image in raster and anti-raster order. The queue-based technique starts with the regional maximas and applies the Breadth First Search (BFS) on them.

Related concepts:

- [Primary Morphology Operations](#)
- [Structuring Elements](#)
- [Connectivity](#)
- [Advanced Morphology Operations](#)

Particle Measurements

This section contains tables that list and describe the NI Vision particle measurements. The tables include definitions, symbols, and equations for particle measurements.



Note Some equation symbols may be defined inside tables later in the chapter.

Introduction 13

A particle is a group of contiguous nonzero pixels in an image. Particles can be characterized by measurements related to their attributes, such as particle location, area, and shape.

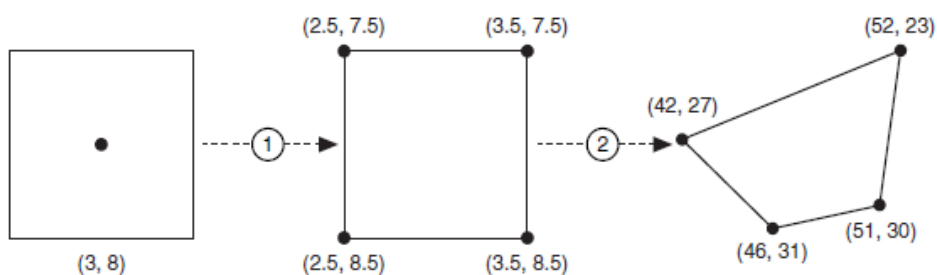
When to Use

Use particle measurements when you want to make shape measurements on particles in a binary image.

Pixel Measurements versus Real-World Measurements

In addition to making conventional pixel measurements, Vision particle analysis functions can use calibration information attached to an image to make measurements in calibrated real-world units. In applications that do not require the display of corrected images, you can use the calibration information attached to the image to make real-world measurements directly without using time-consuming image correction.

In pixel measurements, a pixel is considered to have an area of one square unit, located entirely at the center of the pixel. In calibrated measurements, a pixel is a polygon with corners defined as plus or minus one half a unit from the center of the pixel. The following illustrates this concept.



1. Point to Polygon
2. Pixel Coordinates to Real-World Coordinates

A pixel at $(3, 8)$ is a square with corners at $(2.5, 7.5)$, $(3.5, 7.5)$, $(3.5, 8.5)$, and $(2.5, 8.5)$. To make real-world measurements, the four corner coordinates are transformed from pixel coordinates into real-world coordinates. Using real-world coordinates, the area

and moments of the pixel can be integrated. Similarly, the area and moments of an entire particle can be computed using the calibrated particle contour points.

Particle Measurements

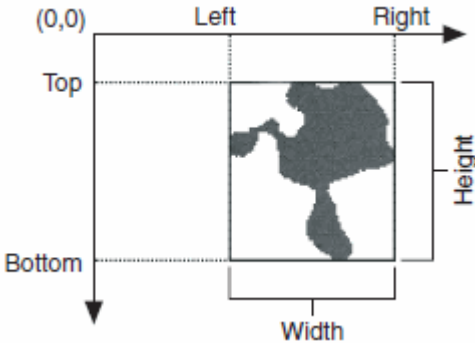
This section contains tables that list and describe the Vision particle measurements. The tables include definitions, symbols, and equations for particle measurements.

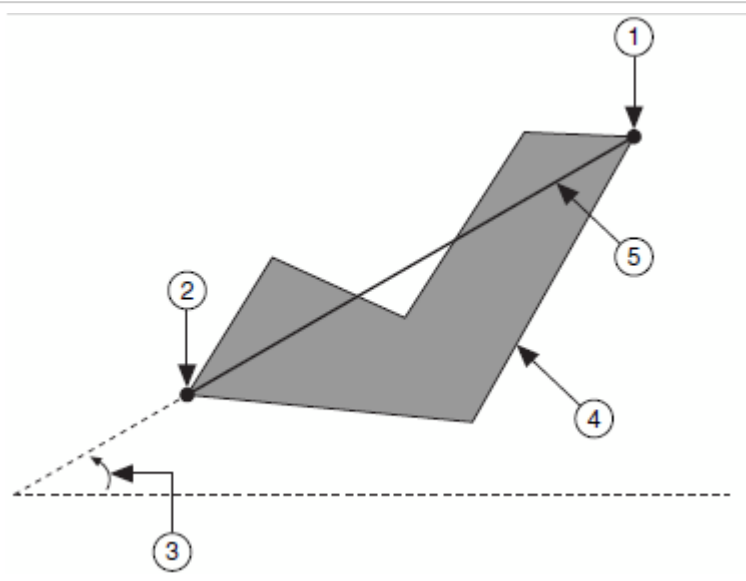



Note Some equation symbols may be defined inside tables later in the section.

Particle Concepts

The following table contains concepts relating to particle measurements.

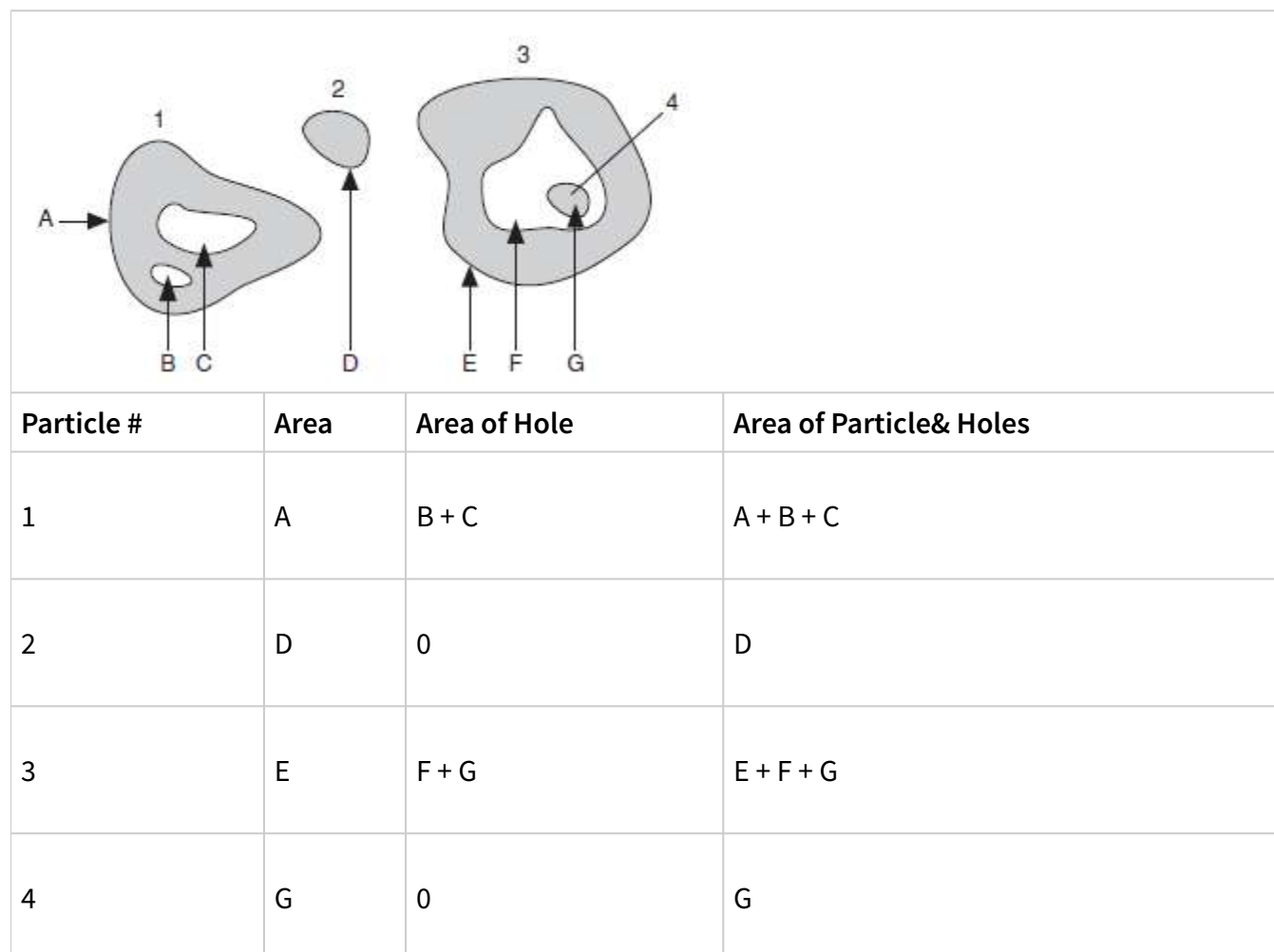
Concept	Definition
Bounding Rect	<p>Smallest rectangle with sides parallel to the x-axis and y-axis that completely encloses the particle.</p> 
Perimeter	<p>Length of a boundary of a region. Because the boundary of a binary image is comprised of discrete pixels, Vision subsamples the boundary points to approximate a smoother, more accurate perimeter. Boundary points are the pixel corners that form the boundary of the particle. Refer to the introduction for an illustration of pixel corners.</p>
Particle	<p>Contiguous region of zero-valued pixels completely surrounded by pixels with nonzero</p>

Concept	Definition
hole	values. Refer to the particle holes section for more information.
Angle	Degrees of rotation measured counter-clockwise from the x-axis, such that $0 \leq \theta < 180$.
Equivalent Rect	Rectangle with the same perimeter and area as the particle.
Equivalent Ellipse	Ellipse with the same perimeter and area as the particle.
Max Feret Diameter	<div><p>Line segment connecting the two perimeter points that are the furthest apart.</p><p>The diagram shows a gray, irregularly shaped particle. A solid line segment connects two points on its perimeter, representing the Max Feret Diameter. Point 1 is at the top-right end of this segment, and point 2 is at the bottom-left end. A dashed line extends from point 2, and an arrow labeled 3 indicates the orientation of the Max Feret Diameter relative to the horizontal. Other points on the particle's perimeter are labeled 4 and 5.</p><ol style="list-style-type: none">1. Max Feret Diameter Start—Highest, leftmost of the two points defining the Max Feret Diameter2. Max Feret Diameter End—Lowest, rightmost of the two points defining the Max Feret Diameter3. Max Feret Diameter Orientation</div>

Concept	Definition
	<div> <div>4. Particle Perimeter</div> <div>5. Max Feret Diameter</div> </div>
Convex Hull	<p>Smallest convex polygon containing all points in the particle. The following figure illustrates two particles, shown in gray, and their respective convex hulls, the areas enclosed by black lines.</p> 
Max Horiz. Segment Length	Longest row of contiguous pixels in the particle. This measurement is always given as a pixel measurement.
Sum	Moments of various orders relative to the x-axis and y-axis.
Moment of Inertia	Moments about the particle center of mass. Provides a representation of the pixel distribution in a particle with respect to the particle center of mass. Moments of inertia are shift invariant.
Norm. Moment of Inertia	Moment of Inertia normalized with regard to the particle area. Normalized moments of inertia are shift and scale invariant.
Hu Moment	Moments derived from the Norm. Moment of Inertia measurements. Hu Moments are shift, scale, and rotation invariant.

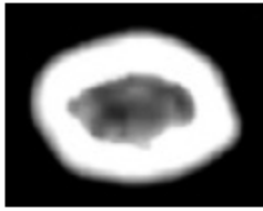
Particle Holes

A particle hole is a contiguous region of zero-valued pixels completely surrounded by pixels with nonzero values. A particle located inside a hole of a bigger particle is identified as a separate particle. The area of a hole that contains a particle includes the area covered by that particle.



Hole measurements are valuable when analyzing particles similar to the one in figure A. For example, if you threshold a cell with a dark nucleus (figure A) so that the nucleus appears as a hole in the cell (figure B), you can make the following cell measurements:

- Area of Hole—Returns the size of the nucleus.
- Area of Particle and Hole—Returns the size of the entire cell.
- Area of Hole/Area of Particle and Hole—Returns the percentage of the cell that the nucleus occupies.



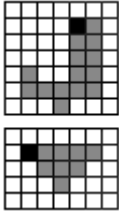
A



B

Coordinates

The following table lists the Vision particle measurements relating to coordinates.

Measurement	Definition	Symbol	Equation
Center of Mass	Point representing the average position of the total particle mass, assuming every point in the particle has a constant density. The center of mass can be located outside the particle if the particle is not convex.	—	—
First Pixel	Highest, leftmost particle pixel. The first pixel is always given as a pixel measurement. The black squares in the following figure represent the first pixel of each particle. 	—	—
Center of Mass x	X-coordinate of the particle Center of Mass.	x	$\frac{\Sigma_x}{A}$
Center of Mass y	Y-coordinate of the particle Center of Mass.	y	$\frac{\Sigma_y}{A}$
First Pixel x	X-coordinate of the first particle pixel.	—	—

Measurement	Definition	Symbol	Equation
First Pixel y	Y-coordinate of the first particle pixel.	—	—
Bounding Rect Left	X-coordinate of the leftmost particle point.	B _L	—
Bounding Rect Top	Y-coordinate of highest particle point.	B _T	—
Bounding Rect Right	X-coordinate of the rightmost particle point.	B _R	—
Bounding Rect Bottom	Y-coordinate of the lowest particle point.	B _B	—
Max Feret Diameter Start x	X-coordinate of the Max Feret Diameter Start.	F _{x1}	—
Max Feret Diameter Start y	Y-coordinate of the Max Feret Diameter Start.	F _{y1}	—
Max Feret Diameter End x	X-coordinate of the Max Feret Diameter End.	F _{x2}	—
Max Feret Diameter End y	Y-coordinate of the Max Feret Diameter End.	F _{y2}	—

Measurement	Definition	Symbol	Equation
Max Horiz. Segment Length Left	X-coordinate of the leftmost pixel in the Max Horiz. Segment. Max Horiz. Segment Length Left is always given as a pixel measurement.	—	—
Max Horiz. Segment Length Right	X-coordinate of the rightmost pixel in the Max Horiz. Segment. Max Horiz. Segment Length Right is always given as a pixel measurement.	—	—
Max Horiz. Segment Length Row	Y-coordinate for all of the pixels in the Max Horiz. Segment. Max Horiz. Segment Length Row is always given as a pixel measurement.	—	—

Lengths

the following table lists the Vision particle relating to length.

Measurement	Definition	Symbol	Equation
Bounding Rect Width	Distance between Bounding Rect Left and Bounding Rect Right .	W	$B_R - B_L$
Bounding Rect Height	Distance between Bounding Rect Top and Bounding Rect Bottom.	H	$B_B - B_T$
Bounding Rect Diagonal	Distance between opposite corners of the Bounding Rect.	—	$\sqrt{W^2 + H^2}$
Perimeter	Length of the outer boundary of the particle. Because the boundary is comprised of discrete	P	—

Measurement	Definition	Symbol	Equation
	pixels, Vision subsamples the boundary points to approximate a smoother, more accurate perimeter.		
Convex Hull Perimeter	Perimeter of the Convex Hull.	P _{CH}	—
Hole Perimeter	Sum of the perimeters of each hole in the particle.	—	—
Max Feret Diameter	Distance between the Max Feret Diameter Start and the Max Feret Diameter End.	F	$\sqrt{(F_{y2} - F_{y1})^2 + (F_{x2} - F_{x1})^2}$
Equivalent Ellipse Major Axis	Length of the major axis of the Equivalent Ellipse.	E _{2a}	$\sqrt{\frac{p^2}{2\pi^2} + \frac{2A}{\pi}} + \sqrt{\frac{p^2}{2\pi^2} - \frac{2A}{\pi}}$
Equivalent Ellipse Minor Axis	Length of the minor axis of the Equivalent Ellipse.	E _{2b}	$\sqrt{\frac{p^2}{2\pi^2} + \frac{2A}{\pi}} - \sqrt{\frac{p^2}{2\pi^2} - \frac{2A}{\pi}}$
Equivalent Ellipse Minor Axis (Feret)	Length of the minor axis of the ellipse with the same area as the particle, and Major Axis equal in length to the Max Feret Diameter.	EF _{2b}	$\frac{4A_{CH}}{\pi \cdot F}$
Equivalent Rect Long Side	Longest side of the Equivalent Rect.	R _a	$\frac{1}{4}P + \sqrt{P^2 - 16A}$

Measurement	Definition	Symbol	Equation
Equivalent Rect Short Side	Shortest side of the Equivalent Rect.	R_b	$\frac{1}{4}P - \sqrt{P^2 - 16A}$
Equivalent Rect Diagonal	Distance between opposite corners of the Equivalent Rect.	R_d	$R_a^2 + R_b^2$
Equivalent Rect Short Side (Feret)	Shortest side of the rectangle with the same area as the particle, and longest side equal in length to the Max Feret Diameter.	RF_b	$\frac{A_{CH}}{F}$
Average Horiz. Segment Length	Average length of a horizontal segment in the particle. Sum of the horizontal segments that do not superimpose any other horizontal segment. Average Horiz. Segment Length is always given as a pixel measurement.	—	$\frac{A}{S_H}$
Average Vert. Segment Length	Average length of a vertical segment in the particle. Sum of the vertical segments that do not superimpose any other vertical segment. Average Vert. Segment Length is always given as a pixel measurement.	—	$\frac{A}{S_V}$
Hydraulic Radius	Particle area divided by the particle perimeter.	—	$\frac{A}{P}$
Waddel Disk Diameter	Diameter of a disk with the same area as the particle.	—	$2\sqrt{\frac{A}{\pi}}$

Ellipses

- **Equivalent Ellipse Major Axis**—Total length of the major axis of the ellipse that has the same area and same perimeter as a particle. This length is equal to $2a$.

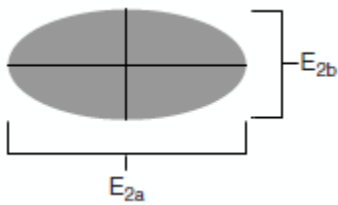
This definition gives the following set of equations:

$$\text{Area} = \pi ab$$

$$\text{Perimeter} = \pi \sqrt{2(a^2 + b^2)}$$

where:

- $a = 1/2 E_{2a}$
- $b = 1/2 E_{2b}$



For a given area and perimeter, only one solution (a, b) exists.

- **Equivalent Ellipse Minor Axis**—Total length of the minor axis of the ellipse that has the same area and same perimeter as a particle. This length is equal to $2b$.
- **Ellipse Ratio**—Ratio of the major axis of the equivalent ellipse to its minor axis, which is defined as

$$\frac{\text{ellipse major axis}}{\text{ellipse minor axis}} + \frac{a}{b}$$

The more elongated the equivalent ellipse, the higher the ellipse ratio. The closer the equivalent ellipse is to a circle, the closer the ellipse ratio is to 1.

Rectangles

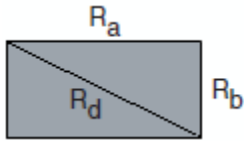
- **Equivalent Rect Long Side**—Length of the long side (R_a) of the rectangle that has the same area and same perimeter as a particle.

This definition gives the following set of equations:

$$A = \text{Area} = R_a R_b$$

$$A = \text{Area} = R_a R_b$$

$$P = \text{Perimeter} = 2(R_a + R_b)$$



This set of equations can be expressed so that the sum $R_a + R_b$ and the product $R_a R_b$ become functions of the parameters **Particle Area** and **Particle Perimeter**. R_a and R_b then become the two solutions of the following polynomial equation:

$$2x^2 - Px + 2A = 0$$

Notice that for a given area and perimeter, only one solution (R_a, R_b) exists.

- **Equivalent Rect Short Side**—Length of the short side of the rectangle that has the same area and same perimeter as a particle. This length is equal to R_b .
- **Equivalent Rect Diagonal**—Distance between opposite corners of the Equivalent Rect:

$$\sqrt{R_a^2 + R_b^2}$$

- **Rectangle Ratio**—Ratio of the long side of the equivalent rectangle to its short side, which is defined as:

$$\frac{\text{rectangle long side}}{\text{rectangle short side}} = \frac{R_a}{R_b}$$

The more elongated the equivalent rectangle, the higher the rectangle ratio.

The closer the equivalent rectangle is to a square, the closer to 1 the rectangle ratio.

Hydraulic Radius

A disk with radius R has a hydraulic radius equal to

$$\frac{\text{disk area}}{\text{disk perimeter}} = \frac{\pi R^2}{2\pi R} = \frac{R}{2}$$

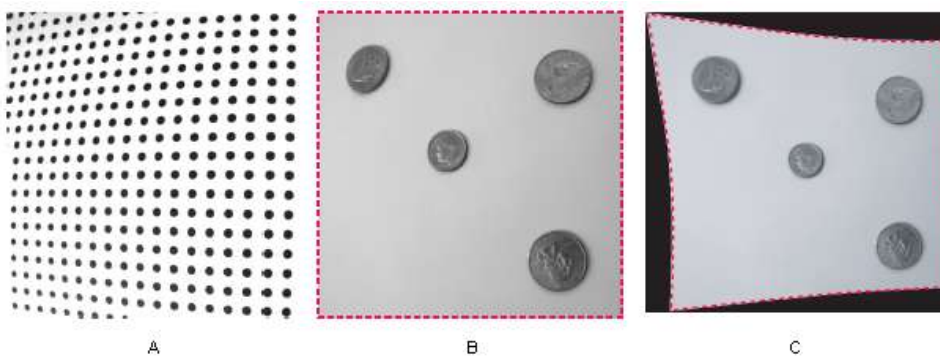
Areas

The following table lists the Vision particle area measurements.

Measurement	Definition	Symbol	Equation
Area	Area of the particle.	A	—
Area of Hole	Sum of the areas of each hole in the particle.	A_H	—
Area of Particle & Holes	Area of a particle that completely covers the image.	A_T	$A + A_H$
Convex Hull Area	Area of the particle Convex Hull.	A_{CH}	—
Image Area	Area of the image.	A_I	—

Image Area

Figure A shows an image of a calibration grid. The image exhibits nonlinear distortion. Figure B shows an image of coins taken with the same camera setup used in figure A. The dashed line around figure B defines the image area in pixels. Figure C illustrates the image of coins after image correction. The dashed line around figure C defines the image area in calibrated units.



Quantities

The following table lists the Vision particle measurements relating to quantity.

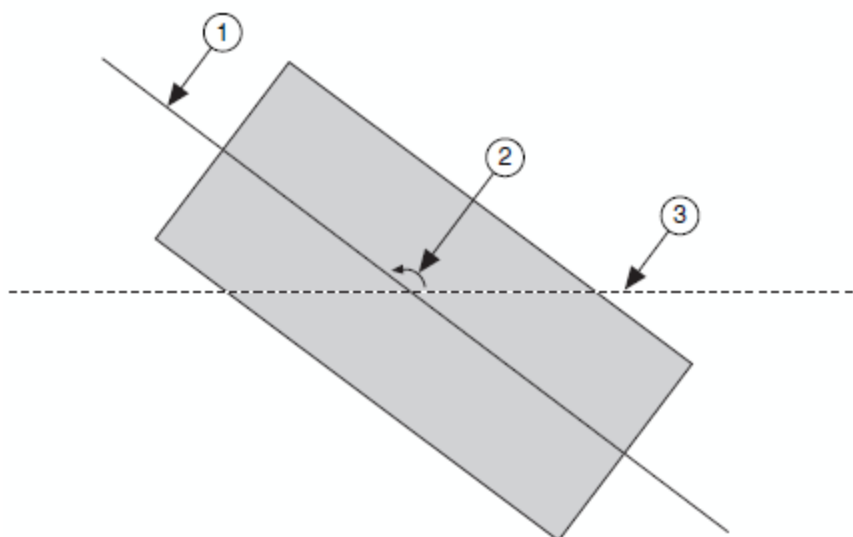
Measurement	Definition	Symbol
Number of Holes	Number of holes in the particle.	—
Number of Horiz. Segments	Number of horizontal segments in the particle. Number of Horiz. Segments is always given as a pixel measurement.	S_H
Number of Vert. Segments	Number of vertical segments in the particle. Number of Vert. Segments is always given as a pixel measurement.	S_V

Angles

The following table lists the Vision particle angle measurements. The equations are given in radians. The results are given in degrees that are mapped into the range 0 to 180, such that $0 \leq \theta < 180$.

Measurement	Definition	Equation
Orientation	The angle of the line that passes through the particle Center of Mass about which the particle has the lowest moment of inertia.	$\frac{1}{2} \operatorname{atan} \left(\frac{\frac{2I_{xy}}{I_{xx} - I_{yy}}}{1} \right)$
Max Feret Diameter Orientation	The angle of the Max Feret Diameter.	$\operatorname{atan} \left(\frac{F_{y1} - F_{y2}}{F_{x1} - F_{x2}} \right)$

The Orientation angle is measured counterclockwise from the horizontal axis, as shown in the following figure. The value can range from 0° to 180° . Angles outside this range are mapped into the range. For example, a 190° angle is considered to be a 10° angle.



1. Line with Lowest Moment of Inertia
2. Orientation in Degrees
3. Horizontal Axis



Note Refer to the max feret diameter entry in the for an illustration of Max Feret Diameter Orientation.

Ratios

The following table lists the Vision particle ratio measurements.

Measurement	Definition	Equation
% Area/Image Area	Percentage of the particle Area covering the Image Area.	$\frac{A}{A_I} \cdot 100 \%$
% Area/(Area of Particle & Holes)	Percentage of the particle Area in relation to the Area of its Particle & Holes	$\frac{A}{A_T} \cdot 100 \%$
Ratio of Equivalent Ellipse Axes	Equivalent Ellipse Major Axis divided by Equivalent Ellipse Minor Axis.	$\frac{E_{2a}}{E_{2b}}$
Ratio of Equivalent Rect Sides	Equivalent Rect Long Side divided by Equivalent Rect Short Side.	$\frac{R_a}{R_b}$

Factors

The following table lists the Vision particle factor measurements.

Measurement	Definition	Equation
Elongation Factor	Max Feret Diameter divided by Equivalent Rect Short Side (Feret). The more elongated the shape of a particle, the higher its elongation factor.	$\frac{F}{RF_b}$
Compactness Factor	Area divided by the product of Bounding Rect Width and Bounding Rect Height. The compactness factor belongs to the interval [0, 1].	$\frac{A}{W \cdot H}$
Heywood Circularity Factor	Perimeter divided by the circumference of a circle with the same area. The closer the shape of a particle is to a disk, the closer the Heywood circularity factor is to 1.	$\frac{P}{2\sqrt{\pi A}}$
Type Factor	Factor relating area to moment of inertia.	$\frac{A^2}{4 \prod \sqrt{I_{xx} \cdot I_{yy}}}$

Sums

The following table lists the Vision particle sum measurements.

Measurement	Symbol
Sum x	Σ_x
Sum y	Σ_y
Sum xx	Σ_{xx}
Sum xy	Σ_{xy}
Sum yy	Σ_{yy}
Sum xxx	Σ_{xxx}
Sum xxy	Σ_{xxy}
Sum xyy	Σ_{xyy}
Sum yyy	Σ_{yyy}

Moments

The following table lists the Vision particle moment measurements.

Measurement	Symbol	Equation
Moment of Inertia xx	I_{xx}	$\sum_{xx} - \frac{\sum_x^2}{A}$
Moment of Inertia xy	I_{xy}	$\sum_{xy} - \frac{\sum_x \cdot \sum_y}{A}$
Moment of Inertia yy	I_{yy}	$\sum_{yy} - \frac{\sum_y^2}{A}$
Moment of Inertia xxx	I_{xxx}	$\sum_{xxx} - 3)\bar{x}\sum_{xx} + 2)\bar{x}^2\sum_x$
Moment of Inertia xxy	I_{xxy}	$\sum_{xxy} - 2)\bar{x}\sum_{xy} -)\bar{y}\sum_{xx} + 2)\bar{x}^2\sum_y$
Moment of Inertia xyy	I_{xyy}	$\sum_{xyy} - 2)\bar{y}\sum_{xy} - \bar{x}\sum_{yy} + 2)\bar{y}^2\sum_x$
Moment of Inertia yyy	I_{yyy}	$\sum_{yyy} - 3)\bar{y}\sum_{yy} + 2)\bar{y}^2\sum_y$
Norm. Moment of Inertia xx	N_{xx}	$\frac{I_{xx}}{A^2}$
Norm. Moment of Inertia xy	N_{xy}	$\frac{I_{xy}}{A^2}$
Norm. Moment of Inertia yy	N_{yy}	$\frac{I_{yy}}{A^2}$
Norm. Moment of Inertia xxx	N_{xxx}	$\frac{I_{xxx}}{A^{5/2}}$
Norm. Moment of	N_{xxy}	$\frac{I_{xxy}}{A^{5/2}}$

Measurement	Symbol	Equation
Inertia xxy		
Norm. Moment of Inertia xyy	N_{xyy}	$\frac{I_{xyy}}{A^5/2}$
Norm. Moment of Inertia yyy	N_{yyy}	$\frac{I_{yyy}}{A^5/2}$
Hu Moment 1	H_1	$N_{xx} + N_{yy}$
Hu Moment 2	H_2	$(N_{xx} + N_{yy})^2 + 4N_{xy}^2$
Hu Moment 3	H_3	$(N_{xxx} - N_{xyy})^2 + (3N_{xxy} - N_{yyy})^2$
Hu Moment 4	H_4	$(N_{xxx} - N_{xyy})^2 + (N_{xxy} - N_{yyy})^2$
Hu Moment 5	H_5	$(N_{xxx} - 3N_{xyy})(N_{xxx} + 3N_{xyy})\left((N_{xxx} + 3N_{xyy})^2 - 3(N_{xxy} + 3N_{yyy})^2\right) + 3(N_{xxy} - N_{yyy})(N_{xxy} - N_{yyy})\left((3N_{xxx} + N_{xyy})\right)$
Hu Moment 6	H_6	$(N_{xx} - N_{yy})\left((N_{xxx} + N_{xyy})^2 - (N_{xxy} + N_{yyy})^2\right) + 4N_{xy}(N_{xxx} + N_{xyy})^2 - (N_{xxy} + N_{yyy})$
Hu Moment 7	H_7	$(3N_{xxy} - N_{yyy})(N_{xxx} + N_{xyy})\left((N_{yyy} + N_{xyy})^2 - 3(N_{xxy} + 3N_{yyy})^2\right) + (3N_{xxy} - N_{yyy})(N_{xxy} + N_{yyy})\left(3(N_{xxx} + N_{xyy})\right)$

Related concepts:

- [Introduction 13](#)

Machine Vision

This section describes conceptual information about high-level operations commonly used in machine vision applications such as edge detection, pattern matching, dimensional measurements, color inspection, binary particle classification, optical character recognition, and instrument reading.

Edge Detection

This section describes edge detection techniques and tools that locate edges, such as the rake, concentric rake, spoke, and caliper.

Introduction

Edge detection finds edges along a line of pixels in the image. Use the edge detection tools to identify and locate discontinuities in the pixel intensities of an image. The discontinuities are typically associated with abrupt changes in pixel intensity values that characterize the boundaries of objects in a scene.

To detect edges in an image, specify a search region in which to locate edges. You can specify the search region interactively or programmatically. When specified interactively, you can use one of the line ROI tools to select the search path you want to analyze. You also can programmatically fix the search regions based either on constant values or the result of a previous processing step. For example, you may want to locate edges along a specific portion of a part that has been previously located using particle analysis or pattern matching algorithms. The edge detection software analyzes the pixels along this region to detect edges. You can configure the edge detection tool to find all edges, find the first edge, the best edge, or find the first and last edges in the region.

When to Use

Edge detection is an effective tool for many machine vision applications. It provides your application with information about the location of object boundaries and the presence of discontinuities.

Use edge detection in the following three application areas: gauging, detection, and alignment.

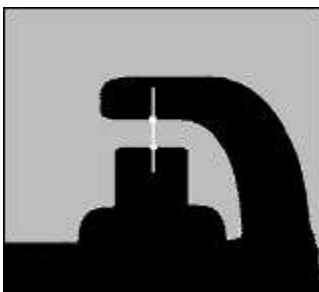
Gauging

Gauging applications make to make critical dimensional measurements, such as length, distance, diameter, angle, and quantity, to determine if the product under inspection is manufactured correctly. Depending on whether the gauged parameters fall inside or outside of the user-defined tolerance limits, the component or part is either classified or rejected.

Gauging is often used both inline and offline in production. During inline processes, each component is inspected as it is manufactured. Visual inline gauging inspection is a widely used inspection technique in applications such as mechanical assembly verification, electronic packaging inspection, container inspection, glass vial inspection, and electronic connector inspection.

Similarly, gauging applications often measure the quality of products offline. First, a sample of products is extracted from the production line. Next, measured distances between features on the object are studied to determine if the sample falls within a tolerance range. You can measure the distances separating the different edges located in an image, as well as positions measured using particle analysis or pattern matching techniques. Edges also can be combined in order to derive best fit lines, projections, intersections, and angles. Use edge locations to compute estimations of shape measurements such as circles, ellipses, or polygons.

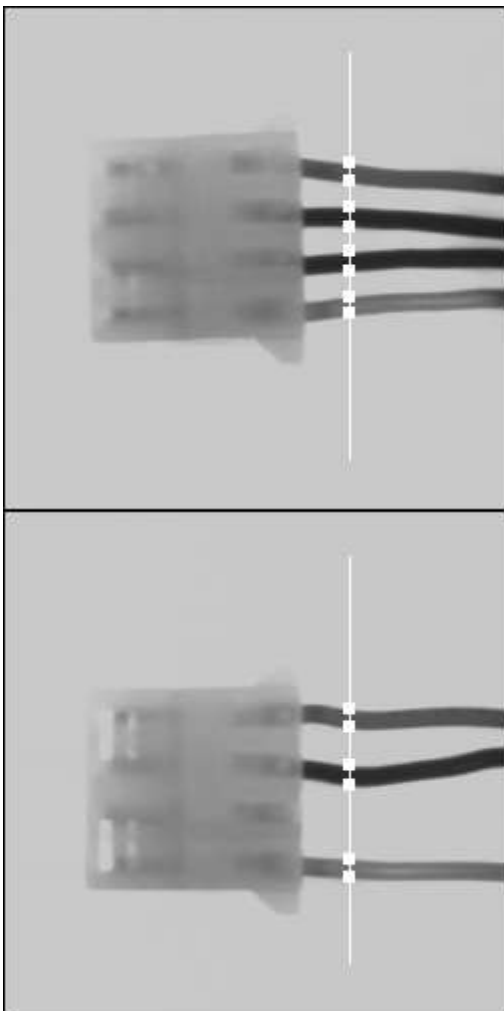
The following figure shows a gauging application using edge detection to measure the length of the gap in a spark plug.



Detection

Part present/not present applications are typical in electronic connector assembly and mechanical assembly applications. The objective of the application is to determine if a part is present or not present using line profiles and edge detection. An edge along the line profile is defined by the level of contrast between background and foreground and the slope of the transition. Using this technique, you can count the number of edges along the line profile and compare the result to an expected number of edges. This method offers a less numerically intensive alternative to other image processing methods such as image correlation and pattern matching.

The following figure shows a simple detection application in which the number of edges detected along the search line profile determines if a connector has been assembled properly. Detection of eight edges indicates that there are four wires. Any other edge count means that the part has been assembled incorrectly.

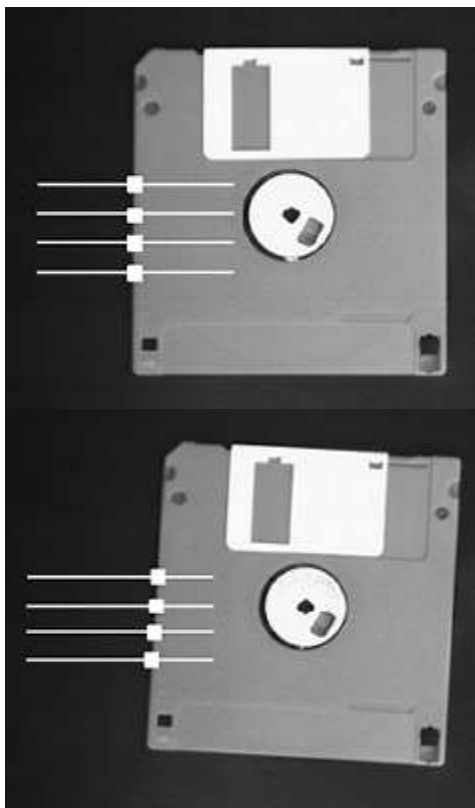


Use edge detection to detect structural defects, such as cracks, or cosmetic defects, such as scratches, on a part. If the part is of uniform intensity, these defects show up as sharp changes in the intensity profile. Edge detection identifies these changes.

Alignment

Alignment determines the position and orientation of a part. In many machine vision applications, the object that you want to inspect may be at different locations in the image. Edge detection finds the location of the object in the image before you perform the inspection, so that you can inspect only the regions of interest. The position and orientation of the part also can be used to provide feedback information to a positioning device, such as a stage.

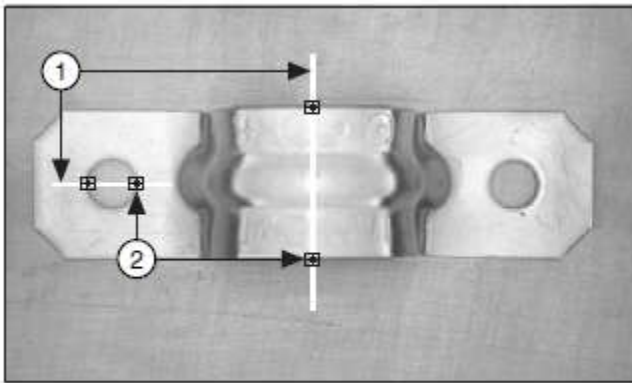
Figure 11-3 shows the detection of the left boundary of a disk in the image. You can use the location of the edges to determine the orientation of the disk.



Concepts

Definition of an Edge

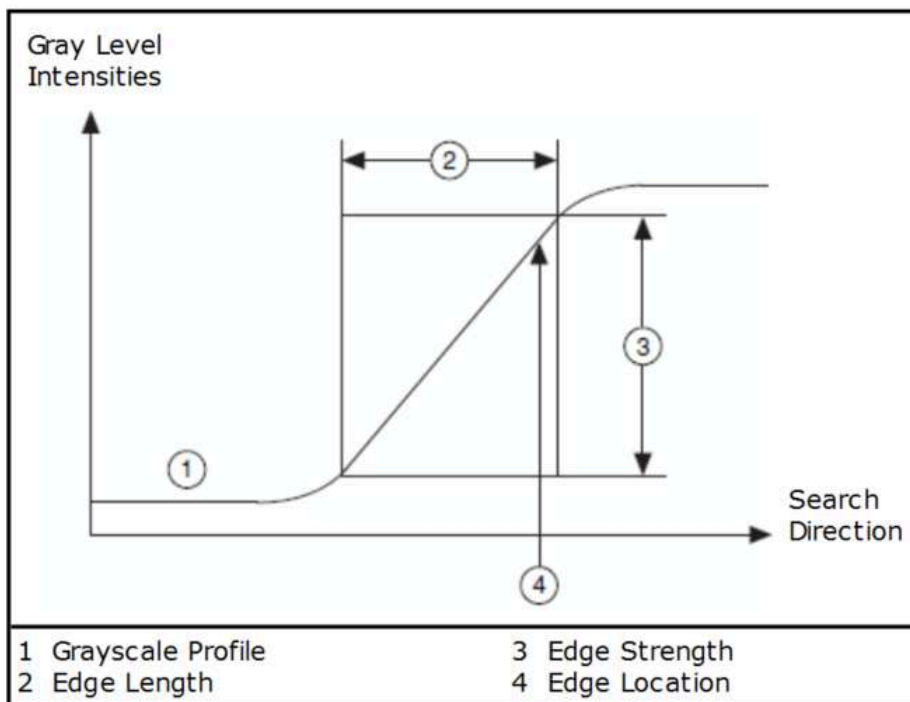
An edge is a significant change in the grayscale values between adjacent pixels in an image. In Vision, edge detection works on a 1D profile of pixel values along a search region, as shown in the following figure. The 1D search region can take the form of a line, the perimeter of a circle or ellipse, the boundary of a rectangle or polygon, or a freehand region. The software analyzes the pixel values along the profile to detect significant intensity changes. You can specify characteristics of the intensity changes to determine which changes constitute an edge.



1. Search Lines
2. Edges

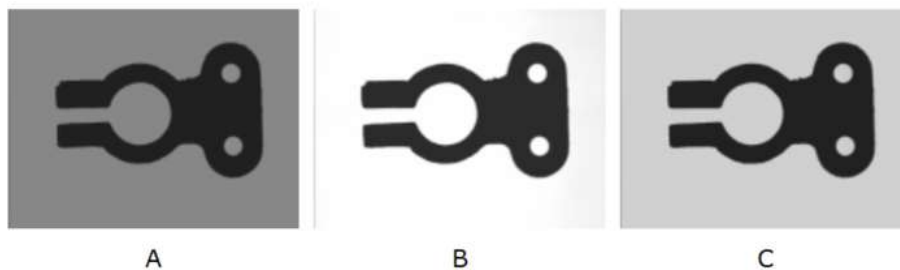
Characteristics of an Edge

The following figure illustrates a common model that is used to characterize an edge.



The following list includes the main parameters of this model.

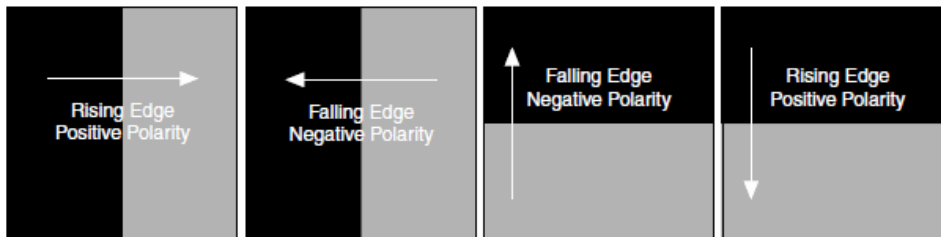
- **Edge strength**—Defines the minimum difference in the grayscale values between the background and the edge. The edge strength is also called the edge contrast. The following figure shows an image that has different edge strengths. The strength of an edge can vary for the following reasons:
 - Lighting conditions—If the overall light in the scene is low, the edges in image will have low strengths. The following figure illustrates a change in the edge strength along the boundary of an object relative to different lighting conditions.
 - Objects with different grayscale characteristics—The presence of a very bright object causes other objects in the image with lower overall intensities to have edges with smaller strengths.



- **Edge length**—Defines the distance in which the desired grayscale difference

between the edge and the background must occur. The length characterizes the slope of the edge. Use a longer edge length, defined by the size of the kernel used to detect edges, to detect edges with a gradual transition between the background and the edge.

- **Edge location**—The x, y location of an edge in the image.
- **Edge polarity**—Defines whether an edge is rising or falling. A rising edge is characterized by an increase in grayscale values as you cross the edge. A falling edge is characterized by a decrease in grayscale values as you cross the edge. The polarity of an edge is linked to the search direction. The following figure shows examples of edge polarities.



Edge Detection Methods

Vision offers two ways to perform edge detection. Both methods compute the edge strength at each pixel along the 1D profile. An edge occurs when the edge strength is greater than a minimum strength. Additional checks find the correct location of the edge. You can specify the minimum strength by using the Minimum Edge Strength or Threshold Level parameter in the software.

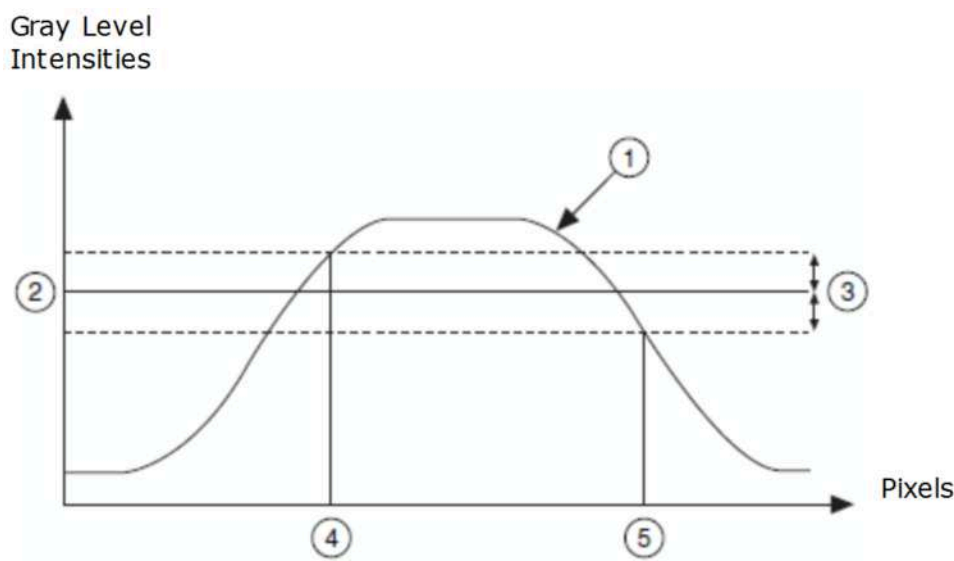
Simple Edge Detection

The software uses the pixel value at any point along the pixel profile to define the edge strength at that point. To locate an edge point, the software scans the pixel profile pixel by pixel from the beginning to the end. A rising edge is detected at the first point at which the pixel value is greater than a threshold value plus a hysteresis value. Set this threshold value to define the minimum edge strength required for qualifying edges. Use the hysteresis value to declare different edge strengths for the rising and falling edges. When a rising edge is detected, the software looks for a falling edge. A falling edge is detected when the pixel value falls below the specified threshold value. This process is repeated until the end of the pixel profile. The first edge along the profile can be either a rising or falling edge.

The simple edge detection method works well when there is little noise in the image

and when there is a distinct demarcation between the object and the background.

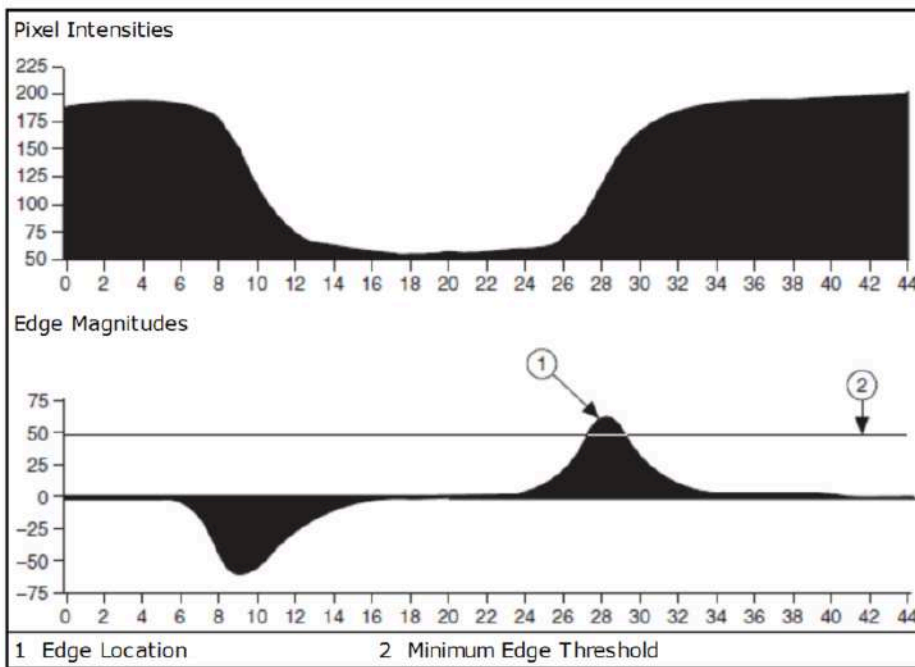
The following figure illustrates the simple edge model.



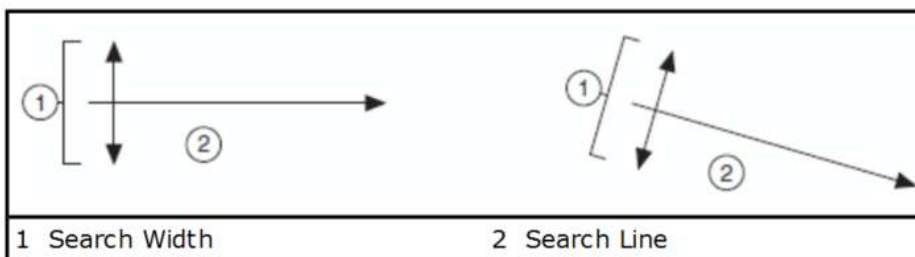
1. Grayscale Profile
2. Threshold Value
3. Hysteresis
4. Rising Edge Location
5. Falling Edge Location

Advanced Edge Detection

The edge detection algorithm uses a kernel operator to compute the edge strength. The kernel operator is a local approximation of a Fourier transform of the first derivative. The kernel is applied to each point in the search region where edges are to be located. For example, for a kernel size of 5, the operator is a ramp function that has 5 entries in the kernel. The entries are $\{-2, -1, 0, 1, 2\}$. The width of the kernel size is user-specified and should be based on the expected sharpness, or slope, of the edges to be located. The following figure shows the pixel data along a search line and the equivalent edge magnitudes computed using a kernel of size 5. Peaks in the edge magnitude profile above a user-specified threshold are the edge points detected by the algorithm.



To reduce the effect of noise in image, the edge detection algorithm can be configured to extract image data along a search region that is wider than the pixels in the image. The thickness of the search region is specified by the search width parameter. The data in the extracted region is averaged in a direction perpendicular to the search region before the edge magnitudes and edge locations are detected. A search width greater than 1 also can be used to find a “best” or “average” edge location or a poorly formed object. The following figure shows how the search width is defined.



Subpixel Accuracy

When the resolution of the image is high enough, most measurement applications make accurate measurements using pixel accuracy only. However, it is sometimes difficult to obtain the minimum image resolution needed by a machine vision application because of limits on the size of the sensors available or the price. In these cases, you need to find edge positions with subpixel accuracy.

Subpixel analysis is a software method that estimates the pixel values that a higher resolution imaging system would have provided. In the edge detection algorithm, the subpixel location of an edge is calculated using a parabolic fit to the edge-detected data points. At each edge position of interest, the peak or maximum value is found along with the value of one pixel on each side of the peak. The peak position represents the location of the edge to the nearest whole pixel.

Using the three data points and the coefficients a , b , and c , a parabola is fitted to the data points using the expression $ax^2 + bx + c$.

The procedure for determining the coefficients a , b , and c in the expression is as follows:

Let the three points which include the whole pixel peak location and one neighbor on each side be represented by (x_0, y_0) , (x_1, y_1) , and (x_2, y_2) . We will let $x_0 = -1$, $x_1 = 0$, and $x_2 = 1$ without loss of generality. We now substitute these points in the equation for a parabola and solve for a , b , and c . The result is:

$$a = \frac{(y_0 + y_2 - 2y_1)}{2}$$

$$b = \frac{(y_2 - y_0)}{2}$$

$$c = y_1$$

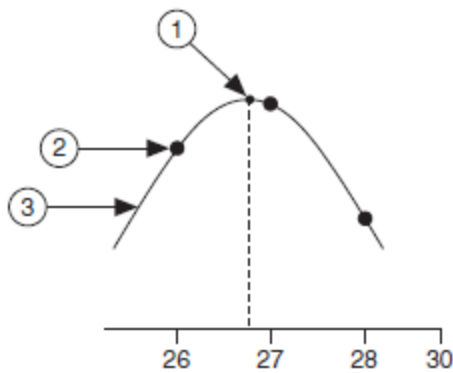
1, which is not needed and can be ignored.

The maximum of the function is computed by taking the first derivative of the parabolic function and setting the result equal to 0. Solving for x yields:

$$x = \frac{-b}{2a}$$

This provides the subpixel offset from the whole pixel location where the estimate of the true edge location lies.

The following illustrates how a parabolic function is fitted to the detected edge pixel location using the magnitude at the peak location and the neighboring pixels. The subpixel location of an edge point is estimated from the parabolic fit.



1. Interpolated Peak Location
2. Neighboring Pixel
3. Interpolating Function

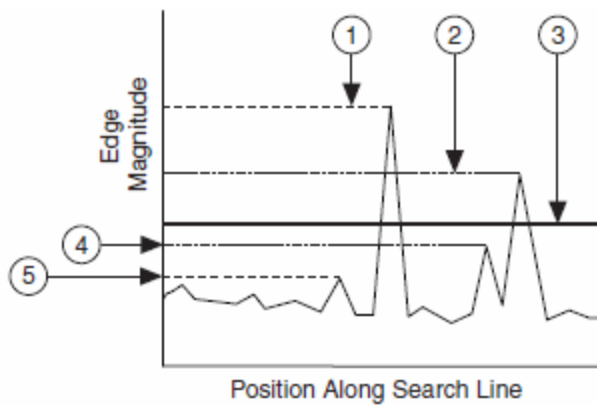
With the imaging system components and software tools currently available, you can reliably estimate 1/25 subpixel accuracy. However, results from an estimation depend heavily on the imaging setup, such as lighting conditions, and the camera lens. Before resorting to subpixel information, try to improve the image resolution. Refer to ***system setup and calibration*** for more information about improving images.

Signal-to-Noise Ratio

The edge detection algorithm computes the signal-to-noise ratio for each detected edge point. The signal-to-noise ratio can be used to differentiate between a true, reliable, edge and a noisy, unreliable, edge. A high signal-to-noise ratio signifies a reliable edge, while a low signal-to-noise ratio implies the detected edge point is unreliable.

In the edge detection algorithm, the signal-to-noise ratio is computed differently depending on the type of edges you want to search for in the image.

When looking for the first, first and last, or all edges along search lines, the noise level associated with a detected edge point is the strength of the edge that lies immediately before the detected edge and whose strength is less than the user-specified minimum edge threshold, as shown in the following figure.

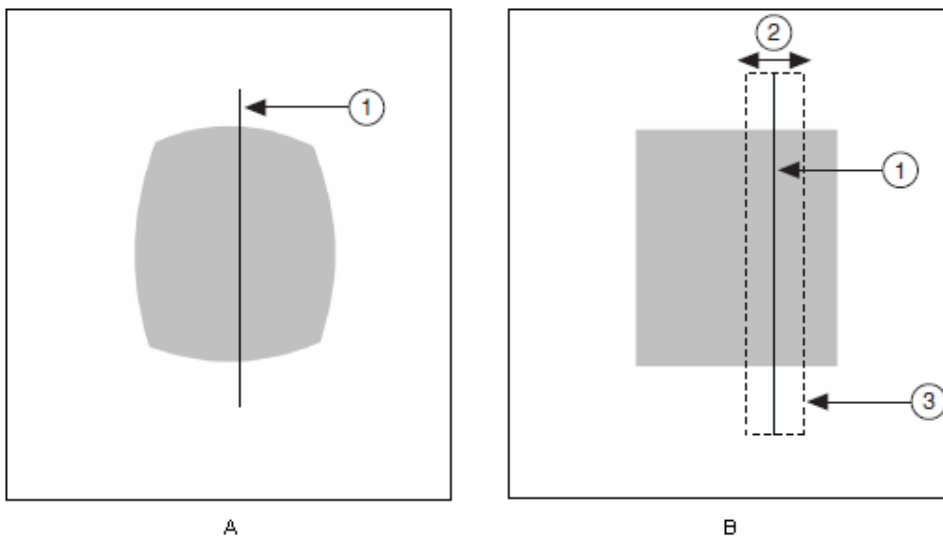


1. Edge 1 Magnitude
2. Edge 2 Magnitude
3. Threshold Level
4. Edge 2 Noise
5. Edge 1 Noise

When looking for the best edge, the noise level is the strength of the second strongest edge before or after the detected edge, as shown in the following figure.

Calibration Support for Edge Detection

The edge detection algorithm uses calibration information in the edge detection process if the original image is calibrated. For simple calibration, edge detection is performed directly on the image and the detected edge point locations are transformed into real-world coordinates. For perspective and non-linear distortion calibration, edge detection is performed on a corrected image. However, instead of correcting the entire image, only the area corresponding to the search region used for edge detection is corrected. Figure A and Figure B illustrate the edge detection process for calibrated images. Figure A shows an uncalibrated distorted image. Figure B shows the same image in a corrected image space.



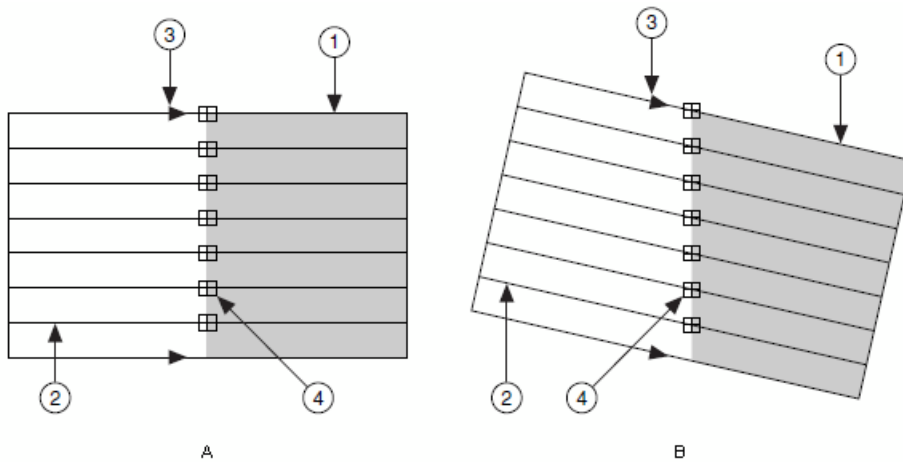
1. Search Line
2. Search Width
3. Corrected Area

Information about the detected edge points is returned in both pixels and real-world units. Refer to ***system setup and calibration*** for more information about calibrating images.

Extending Edge Detection to 2D Search Regions The edge detection tool in NI Vision works on a 1D profile. The rake, spoke, and concentric rake tools extend the use of edge detection to two dimensions. In these edge detection variations, the 2D search area is covered by a number of search lines over which the edge detection is performed. You can control the number of the search lines used in the search region by defining the separation between the lines.

Rake

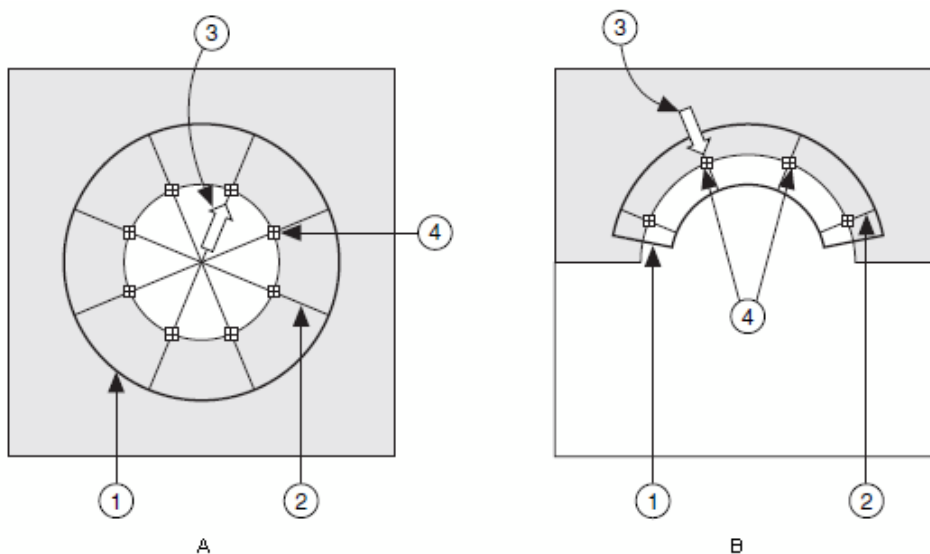
A Rake works on a rectangular search region, along search lines that are drawn parallel to the orientation of the rectangle. Control the number of lines in the area by specifying the search direction as left to right or right to left for a horizontally oriented rectangle. Specify the search direction as top to bottom or bottom to top for a vertically oriented rectangle. The following figure illustrates the basics of the rake function.



1. Search Area
2. Search Line
3. Search Direction
4. Edge Points

Spoke

A Spoke works on an annular search region, along search lines that are drawn from the center of the region to the outer boundary and that fall within the search area. Control the number of lines in the region by specifying the angle between each line. Specify the search direction as either from the center outward or from the outer boundary to the center. The following figure illustrates the basics of the spoke function.

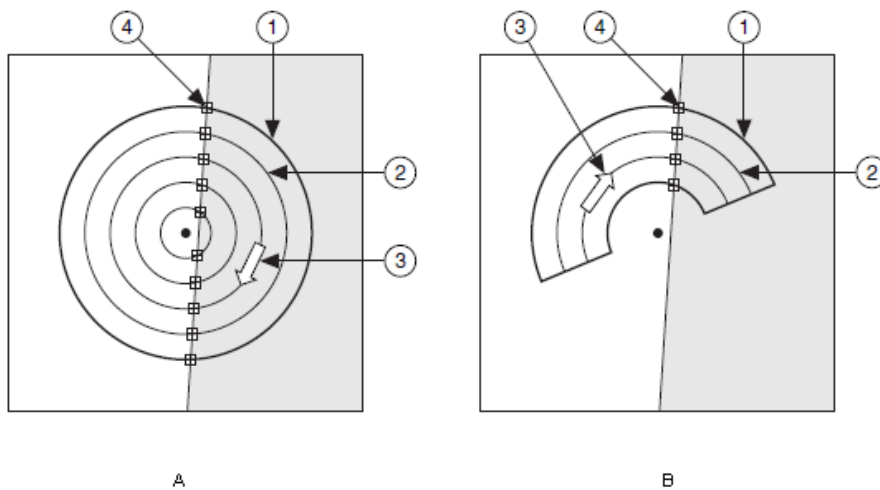


1. Search Area
2. Search Line

3. Search Direction
4. Edge Points

Concentric Rake

A Concentric Rake works on an annular search region. It is an adaptation of the rake to an annular region. The following illustrates the basics of the concentric rake. Edge detection is performed along search lines that occur in the search region and that are concentric to the outer circular boundary. Control the number of concentric search lines that are used for the edge detection by specifying the radial distance between the concentric lines in pixels. Specify the direction of the search as either clockwise or anti-clockwise.



1. Search Area
2. Search Line
3. Search Direction
4. Edge Points

Finding Straight Edges

Finding straight edges is another extension of edge detection to 2D search regions. Finding straight edges involves finding straight edges, or lines, in an image within a 2D search region. Straight edges are located by first locating 1D edge points in the search region and then computing the straight lines that best fit the detected edge points. Straight edge methods can be broadly classified into two distinct groups based on how the 1D edge points are detected in the image.

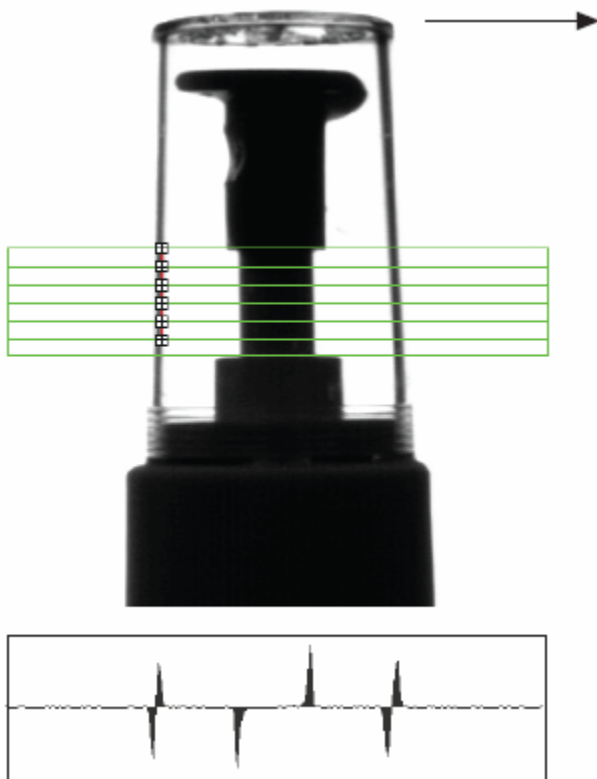
Rake-Based Methods

A Rake is used to detect edge points within a rectangular search region. Straight lines are then fit to the edge points. Three different options are available to determine the edge points through which the straight lines are fit.

First Edges

A straight line is fit through the first edge point detected along each search line in the Rake. The method used to fit the straight line is described in ***dimensional measurements***. The following figure shows an example of the straight edge detected on an object using the first dark to bright edges in the Rake along with the computed edge magnitudes along one search line in the Rake.

The following illustration shows the search direction:

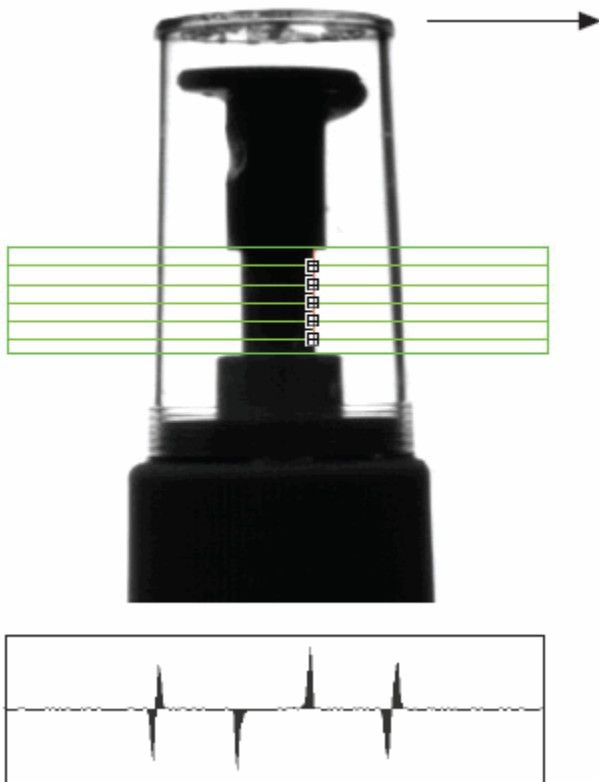


Best Edges

A straight line is fit through the best edge point along each search line in the Rake. The method used to fit the straight line is described in ***dimensional measurements***.

The following figure shows an example of the straight edge detected on an object using the best dark to bright edges in the Rake along with the computed edge magnitudes along one search line in the Rake.

The following illustration shows the search direction:



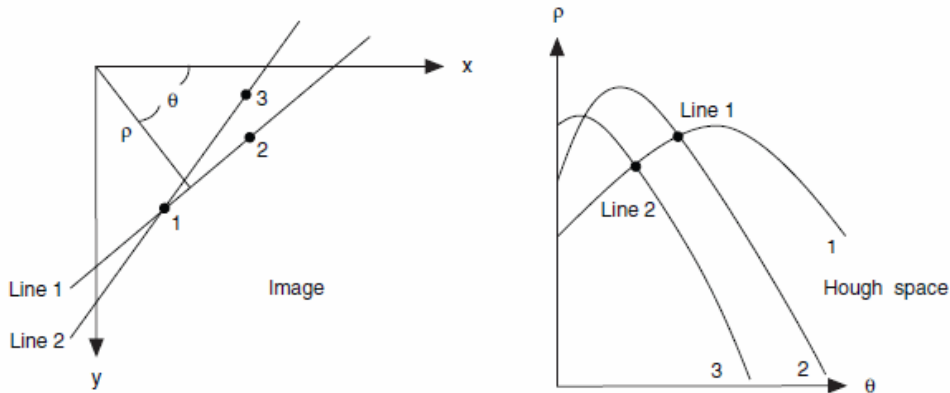
Hough-Based Methods

In this method, a Hough transform is used to detect the straight edges, or lines, in an image. The Hough transform is a standard technique used in image analysis to find curves that can be parameterized, such as straight lines, polynomials, and circles. For detecting straight lines in an image, Vision uses the parameterized form of the line:

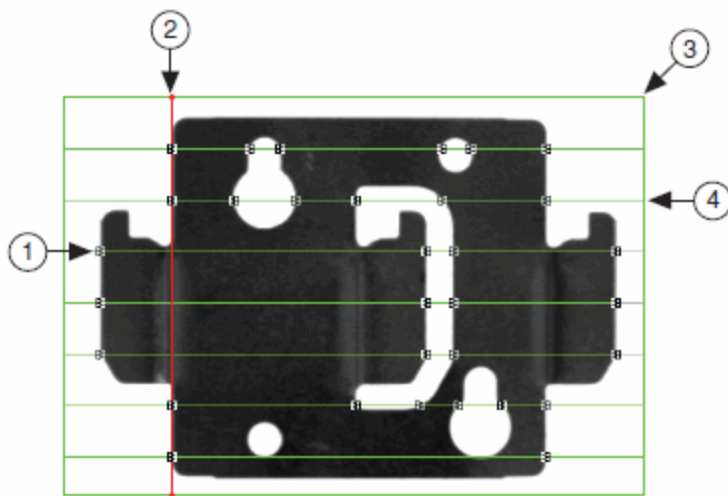
$$\rho = x \cos \theta + y \sin \theta$$

where, ρ is the perpendicular distance from the origin to the line and θ is the angle of the normal from the origin to the line. Using this parameterization, a point (x, y) in the image is transformed into a sinusoidal curve in the (ρ, θ) , or Hough space. The following figure illustrates the sinusoidal curves formed by three image points in the Hough space. The curves associated with colinear points in the image, intersect at a

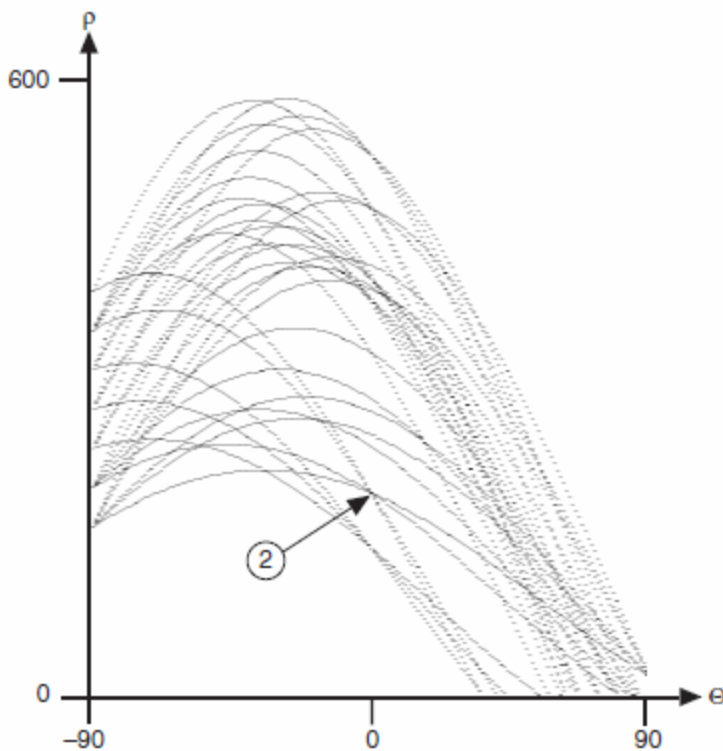
unique point in the Hough space. The coordinates (ρ, θ) of the intersection are used to define an equation for the corresponding line in the image. For example, the intersection point of the curves formed by points 1 and 2 represent the equation for Line1 in the image.



The following figure illustrates how NI Vision uses the Hough transform to detect straight edges in an image. The location (x, y) of each detected edge point is mapped to a sinusoidal curve in the (ρ, θ) space. The Hough space is implemented as a two-dimensional histogram where the axes represent the quantized values for ρ and θ . The range for ρ is determined by the size of the search region, while the range for θ is determined by the angle range for straight lines to be detected in the image. Each edge location in the image maps to multiple locations in the Hough histogram, and the count at each location in the histogram is incremented by one. Locations in the histogram with a count of two or more correspond to intersection points between curves in the (ρ, θ) space. Figure B shows a two-dimensional image of the Hough histogram. The intensity of each pixel corresponds to the value of the histogram at that location. Locations where multiple curves intersect appear darker than other locations in the histogram. Darker pixels indicate stronger evidence for the presence of a straight edge in the original image because more points lie on the line. The following figure also shows the line formed by four edge points detected in the image and the corresponding intersection point in the Hough histogram.



A



B

1. Edge Point
2. Straight Edge
3. Search Region
4. Search Line

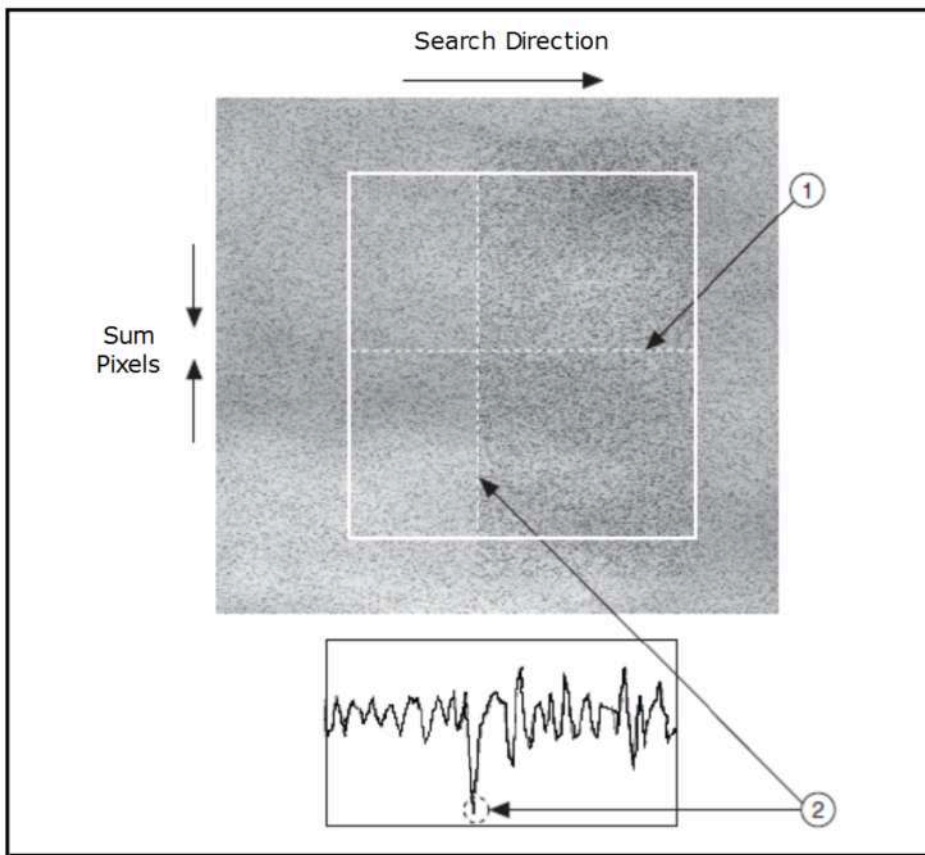
Straight edges in the image are detected by identifying local maxima, or peaks in the Hough histogram. The local maxima are sorted in descending order based on the histogram count. To improve the computational speed of the straight edge detection

process, only a few of the strongest peaks are considered as candidates for detected straight edges. For each candidate, a score is computed in the original image for the line that corresponds to the candidate. The line with the best score is returned as the straight edge. The Hough-based method also can be used to detect multiple straight edges in the original image. In this case, the straight edges are returned based on their scores.

Projection-Based Methods

The projection-based method for detecting straight edges is an extension of the 1D edge detection process discussed in the advanced edge detection section. The following figure illustrates the projection-based straight edge detection process. The algorithm takes in a search region, search direction, and an angle range. The algorithm first either sums or finds the medians of the data in a direction perpendicular to the search direction. Vision then detects the edge position on the summed profile using the 1D edge detection function. The location of the edge peak is used to determine the location of the detected straight edge in the original image.

To detect the best straight edge within an angle range, the same process is repeated by rotating the search ROI through a specified angle range and using the strongest edge found to determine the location and angle of the straight edge.



1. Projection Axis
2. Best Edge Peak and Corresponding Line in the Image

The projection-based method is very effective for locating noisy and low-contrast straight edges.

The projection-based method also can detect multiple straight edges in the search region. For multiple straight edge detection, the strongest edge peak is computed for each point along the projection axis. This is done by rotating the search region through a specified angle range and computing the edge magnitudes at every angle for each point along the projection axis. The resulting peaks along the projection axis correspond to straight edges in the original image.

Straight Edge Score

NI Vision returns an edge detection score for each straight edge detected in an image. The score ranges from 0 to 1000 and indicates the strength of the detected straight edge.

The edge detection score is defined as:

$$s_{\frac{c}{m+n}}$$

where:

- s is the edge detection score,
- c is the sum of the gradients at the edge points that match the specified edge polarity,
- m is the number of edge points on the straight line that match the specified edge polarity,
- and n is the number of edge points on the straight line that do not match the specified edge polarity.

Related concepts:

- [Spatial Calibration](#)
- [Dimensional Measurements](#)

Pattern Matching

This section contains information about pattern matching.

Introduction

Pattern matching quickly locates regions of a grayscale image that match a known reference pattern, also referred to as a model or template.



Note A template is an idealized representation of a feature in the image.

Refer to the ***pattern matching techniques*** section for the definition of an image feature.

When using pattern matching, you create a template that represents the object for which you are searching. Your machine vision application then searches for instances of the template in each acquired image, calculating a score for each match. This score relates how closely the template resembles the located matches.

Pattern matching finds template matches regardless of lighting variation, blur, noise, and geometric transformations such as shifting or rotation of the template.

Related concepts:

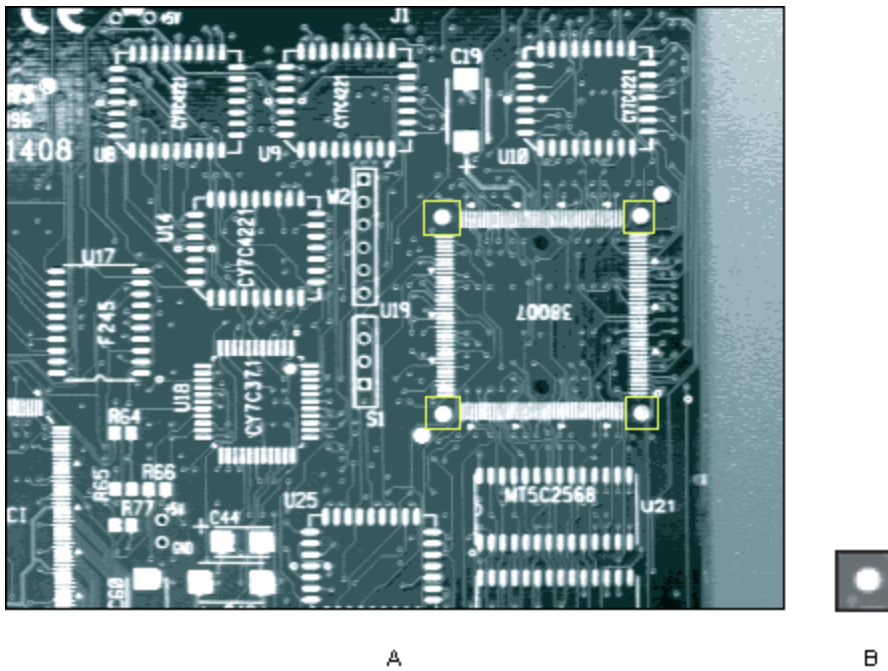
- [Pattern Matching Techniques](#)

When to Use

Pattern matching algorithms are some of the most important functions in machine vision because of their use in varying applications. You can use pattern matching in the following three general applications:

- **Alignment**—Determines the position and orientation of a known object by locating fiducials. Use the fiducials as points of reference on the object.
- **Gauging**—Measures lengths, diameters, angles, and other critical dimensions. If the measurements fall outside set tolerance levels, the component is rejected. Use pattern matching to locate the object you want to gauge.
- **Inspection**—Detects simple flaws, such as missing parts or unreadable print.

Pattern matching provides your application with the number of instances and the locations of template matches within an inspection image. For example, you can search an image containing a printed circuit board (PCB) for one or more fiducials. The machine vision application uses the fiducials to align the board for chip placement from a chip mounting device. Figure A shows part of a PCB. Figure B shows a common fiducial used in PCB inspections or chip pick-and-place applications.



Gauging applications first locate and then measure, or gauge, the dimensions of an object in an image. If the measurement falls within a tolerance range, the object passes inspection. If it falls outside the tolerance range, the object is rejected.

Searching for and finding image features is the key processing task that determines the success of many gauging applications, such as inspecting the leads on a quad pack or inspecting an antilock-brake sensor. In real-time applications, search speed is critical.

In general, pattern matching works well on images where the template is primarily characterized by grayscale information. Templates containing texture, or that have dense, intricate data with no discernible pattern, are the most successful.

Limitations

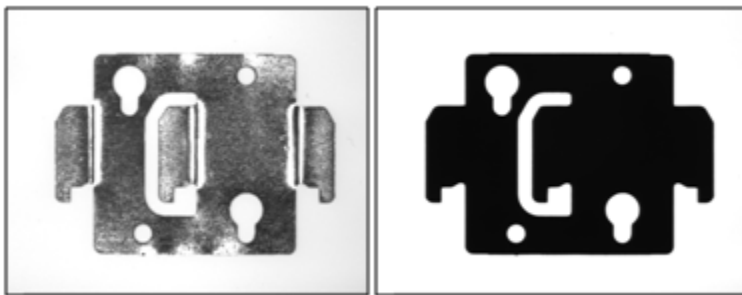
Pattern matching provides a fast, general purpose algorithm to locate an object in a image. However, pattern matching is not well suited to applications where the object to be detected is scaled (Figure A) or if more than 10% of the image is occluded (Figure B). Non-uniform lighting (Figure C) of search images can reduce the effectivity of pattern matching. Applications that are likely to encounter these conditions should use **Geometric Matching** instead.



A



B



C

Related concepts:

- [Geometric Matching](#)

What to Expect from a Pattern Matching Tool

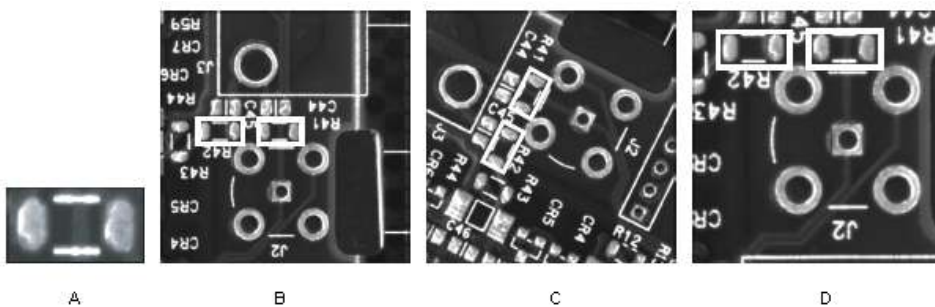
Because pattern matching is the first step in many machine vision applications, it must work reliably under various conditions. In automated machine vision applications, the visual appearance of materials or components under inspection can change because

of varying factors such as part orientation, scale changes, and lighting changes. The pattern matching tool must maintain its ability to locate the reference patterns despite these changes. The following sections describe common situations in which the pattern matching tool needs to return accurate results.

Pattern Orientation and Multiple Instances

A pattern matching algorithm needs to locate the reference pattern in an image even if the pattern in the image is rotated or scaled. When a pattern is rotated or scaled in the image, the pattern matching tool can detect the following items in the image:

- The pattern in the image.
- The position of the pattern in the image.
- The orientation of the pattern.
- Multiple instances of the pattern in the image, if applicable.



- Figure A shows a template image.
- Figure B shows a template match shifted in the image.
- Figure C shows a template match rotated in the image.
- Figure D shows a template match scaled in the image.

Figures B, C, and D also illustrate multiple instances of the template.

Ambient Lighting Conditions

A pattern matching algorithm needs the ability to find the reference pattern in an image under conditions of uniform lighting changes in the lighting across the image. The following figure illustrates the typical conditions under which pattern matching works correctly.

- Figure A shows the original template image.

- Figure B shows a template match under bright light.
- Figure C shows a template match under poor lighting.



Blur and Noise Conditions

A pattern matching algorithm needs the ability to find patterns that have undergone some transformation because of blurring or noise. Blurring usually occurs because of incorrect focus or depth of field changes. Refer to ***system setup and calibration*** for more information about depth of field.

The following figure illustrates typical blurring and noise conditions under which pattern matching works correctly.

- Figure A shows the original template image.
- Figure B shows the changes on the image caused by blurring.
- Figure C shows the changes on the image caused by noise.



Related concepts:

- [Spatial Calibration](#)

Pattern Matching Techniques

Vision implements two pattern matching methods - pyramidal matching and image understanding (low discrepancy sampling). Both methods use normalized cross-correlation as a core technique.

The pattern matching process consists of two stages: learning and matching. During the learning stage, the algorithm extracts gray value and/or edge gradient information

from the template image. The algorithm organizes and stores the information in a manner that facilitates faster searching in the inspection image. In Vision, the information learned during this stage is stored as part of the template image.

During the matching stage, the pattern matching algorithm extracts gray value and/or edge gradient information from the inspection image (corresponding to the information learned from the template). Then, the algorithm finds matches by locating regions in the inspection image where the highest cross-correlation is observed.

Normalized Cross-Correlation

Normalized cross-correlation is the most common method for finding a template in an image. Because the underlying mechanism for correlation is based on a series of multiplication operations, the correlation process is time consuming. Technologies such as MMX allow for parallel multiplications and reduce overall computation time. To increase the speed of the matching process, reduce the size of the image and restrict the region of the image in which the matching occurs. Pyramidal matching and image understanding are two ways to increase the speed of the matching process.

Challenges in Scale-Invariant and Rotation-Invariant Matching

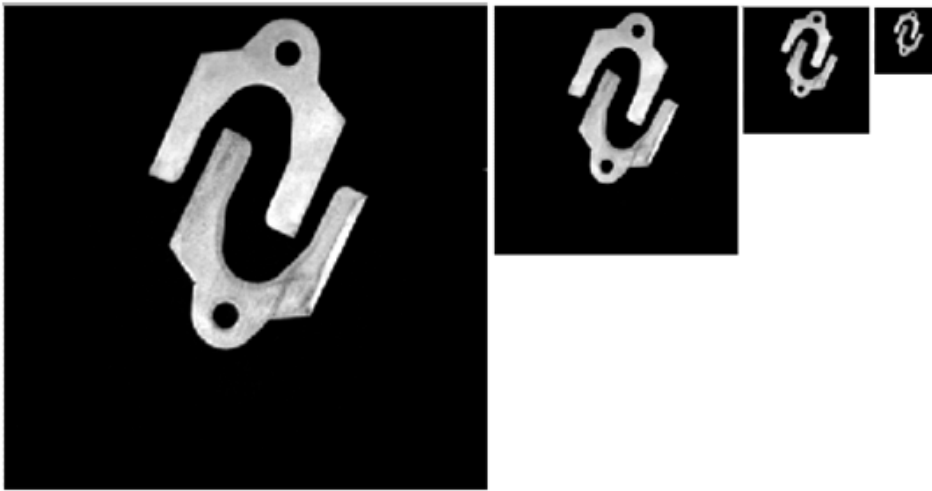
Normalized cross-correlation is a good technique for finding patterns in an image when the patterns in the image are not scaled or rotated. Typically, cross-correlation can detect patterns of the same size up to a rotation of 5° to 10° . Extending correlation to detect patterns that are invariant to scale changes and rotation is difficult.

For scale-invariant matching, you must repeat the process of scaling or resizing the template and then perform the correlation operation. This adds a significant amount of computation to your matching process. Normalizing for rotation is even more difficult. If a clue regarding rotation can be extracted from the image, you can simply rotate the template and perform the correlation. However, if the nature of rotation is unknown, looking for the best match requires exhaustive rotations of the template.

By employing a coarse-to-fine approach to matching, and by using pyramids or image understanding, we can eliminate a significant amount of computation and achieve usable search times for handling rotated patterns. However, scale-invariant matching is not supported even when using pyramids or image understanding.

Pyramidal Matching

You can improve the computation time of pattern matching by reducing the size of the image and the template. In pyramidal matching, both the image and the template are sampled to smaller spatial resolutions using Gaussian pyramids. This method samples every other pixel and thus the image and the template can both be reduced to one-fourth of their original sizes for every successive pyramid level.



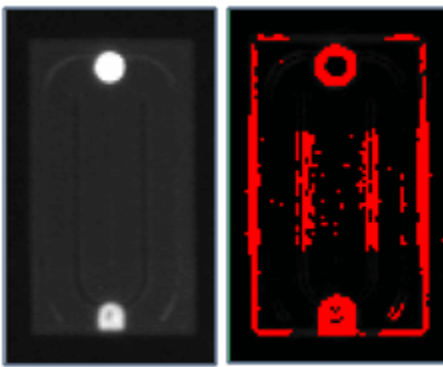
In the learning phase, the algorithm automatically computes the maximum pyramid level that can be used for the given template, and learns the data needed to represent the template and its rotated versions across all pyramid levels. The algorithm attempts to find an optimal pyramid level (based on an analysis of template data) which would give the fastest and most accurate match. Two kinds of data can be used - gray value (based on pixel intensities) and gradients (based on select edge information).

Gray Value Method

This method makes use of the normalized pixel gray values as features. Doing so ensures that no information is left out, which is helpful when the template does not contain structured information, but has intricate textures or dense edges. However, this method has the disadvantage of suffering when faced with occlusion and non-uniform illumination changes. Despite these limitations, the method works over a wide variety of scenarios and is suitable for general use.

Gradient Method

This method makes use of filtered edge pixels as features. An edge image is computed from the supplied grayscale image and a gradient intensity threshold is computed based on image analysis of the template. All edge vectors which are stronger than the threshold are retained as features. Matching is based on vector correlation rather than normalized cross-correlation. This method is more resistant to occlusion and lighting intensity changes as compared to the gray value based method, and is often faster, since less data must be computed. However, as the strength and reliability of edges reduces at very low resolutions, this method requires the user to work at higher resolutions compared to the Gray Value method.



Coarse-to-Fine Matching

The matching phase makes use of a coarse-to-fine approach, starting our search at the lowest resolution possible (the highest pyramid level). Since the sizes of the search image and template have been significantly reduced at this resolution, we can carry out an exhaustive correlation-based search. However, the sub-sampling process introduces some loss of details, and the match locations are not completely reliable. This problem is offset by maintaining a collection of promising candidate match locations with the best scores, rather than choosing the exact number of matches to look for.

We then iterate through each of the lower levels of the pyramid, refining our choice at each stage by re-computing correlation scores. This approach limits all subsequent searches to small localized regions around the best match candidates, achieving a significant speed boost.

When searching for rotated matches, performing an exhaustive match for all possible

rotations (from 0 to 360 degrees) is still prohibitively expensive, even at the lowest resolutions. Consequently, we first exhaustively find the best locations at a coarse angle step. The best locations among these coarse locations are then refined at a finer angle step size. After this, we follow the same method as above by refining the match location as well as angle over the subsequent lower pyramid levels.

Tips and Tricks

Follow these recommendations to obtain the best performance from pyramidal matching:

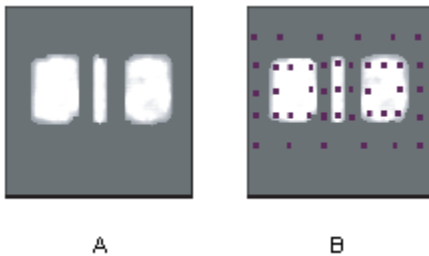
- Use the highest possible pyramid level while choosing the **Max Match Pyramid Level** setting for the fastest execution times. In most cases, matching at level 0 or level 1 might be too expensive.
- While searching for rotated patterns, use the **Angle Ranges** setting to limit the search to the smallest angle range for faster performance and lesser memory consumption. For example if the match is known to be only slightly rotated from the base position, an angle range of -10° to 10° might suffice.
- The algorithm automatically handles the coarse-to-fine matching based on the **Number of Matches Requested** and **Minimum Match Score**. Configure them to obtain the best mix of speed and accuracy for your application.
- The **Minimum Contrast** setting specifies a minimum contrast value a region must exhibit to be considered as a candidate. Use this for getting a speed boost in cases where there are significant areas of low or zero contrast (uniform regions) in the image background.
- If you wish to find potential matches which may lie partially outside the image or the Region of Interest, switch the **Process Border Pixels** setting to on. For larger templates with a well-defined Region of Interest, you may get a slight speed boost by turning this setting off.
- Use the **Min Match Separation Distance**, **Min Match Separation Angle** and **Max Match Overlap** settings to completely control the distance, angular resolution, and overlap between found matches.

Image Understanding (Low Discrepancy Sampling)

A pattern matching feature is a salient pattern of pixels that describe a template. Because most images contain redundant information, using all the information in the image to match patterns is time-insensitive and inaccurate.

Vision uses a non-uniform sampling technique that incorporates image understanding to thoroughly and efficiently describe the template. This intelligent sampling technique specifically includes a combination of edge pixels and region pixels as shown in figure B. NI Vision uses a similar technique when the user indicates that the pattern might be rotated in the image. This technique uses specially chosen template pixels whose values—or relative changes in values—reflect the rotation of the pattern.

Intelligent sampling of the template both reduces the redundant information and emphasizes the feature to allow for an efficient, yet robust, cross-correlation implementation. NI Vision pattern matching is able to accurately locate objects that vary in size ($\pm 5\%$) and orientation (between 0° and 360°) and that have a degraded appearance.



Pyramidal Matching

Similar to pyramidal matching, sampling based matching also employs a coarse-to-fine approach to eliminate excessive computation. First, coarse features are extracted based on region pixel samples. Then only a small number of probable candidate locations are chosen where finer features (primarily using edge pixels) are computed. The coarse match candidates are then refined using these fine features and a revised list of matches is obtained.

Matching under rotation also follows a similar paradigm to pyramidal matching. Intelligent sampling allows us to compactly represent template samples corresponding to different angles. Initially, we find matches using coarse sampling and at a coarse angle step size. The best coarse angle match locations are then refined at a finer angle step size and using finer features.

Limitations

Low discrepancy sampling extracts the most significant information to represent an

image. While this leads to a very sparse and efficient representation in most cases, certain types of images are known to cause problems:

- Templates containing large regions of similar grayscale information, with very little information, can sometimes exhibit inconsistent behavior due to a low number of sample points.
- Templates with skewed or long aspect ratios (1:6) may suffer from inconsistent results when searching for rotated matches.
- Very small templates are sometimes found to contain an insufficient number of samples for reliable training.

If these limitations negatively impact the performance of your application, use a pyramidal matching method.

Pyramid Pre-Processing

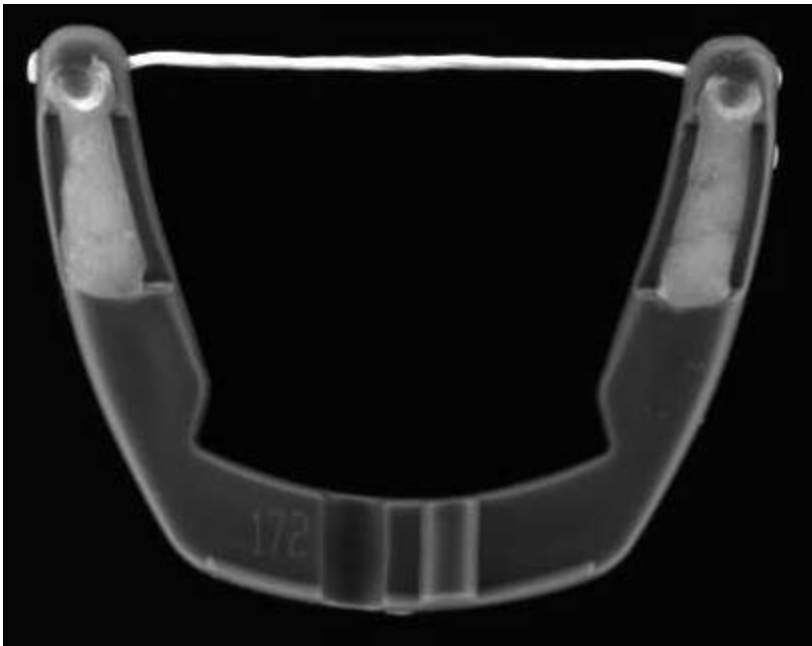
Pyramid Pattern Matching provides three types of Pre-Processing options:

- Sobel and Log,
- Sobel,
- Non-Linear Diffusion Filter.

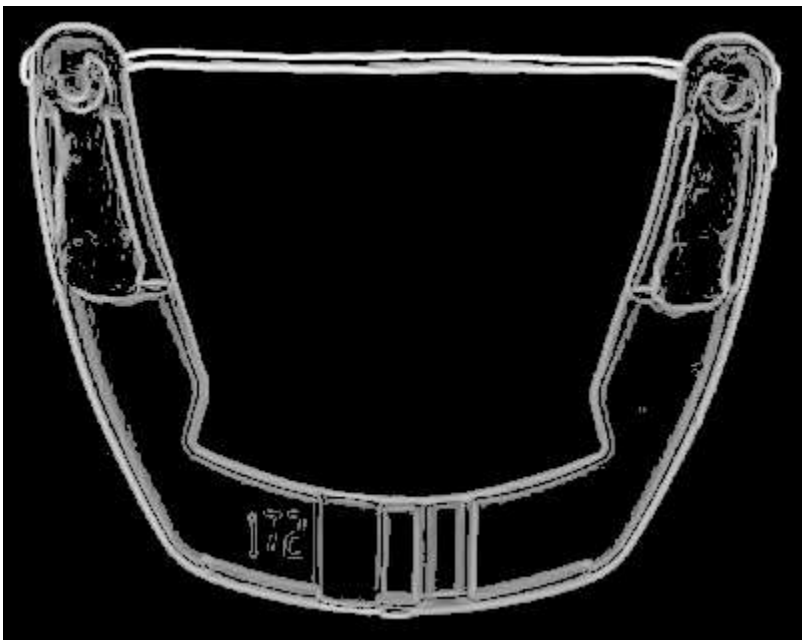
Sobel and Log

This filter applies Sobel kernel convolution on template and match image pyramid. Use this option to enhance the low contrast region and to consider only the edge features from the template.

The illustration shows the Original Image:



The illustration shows the After Sobel and Log image:



Sobel

This filter applies Sobel kernel convolution on template and match image pyramid. Use this filter to use only the edge feature from the template.

The illustration shows the Original Image:



The illustration shows the After Sobel Image:



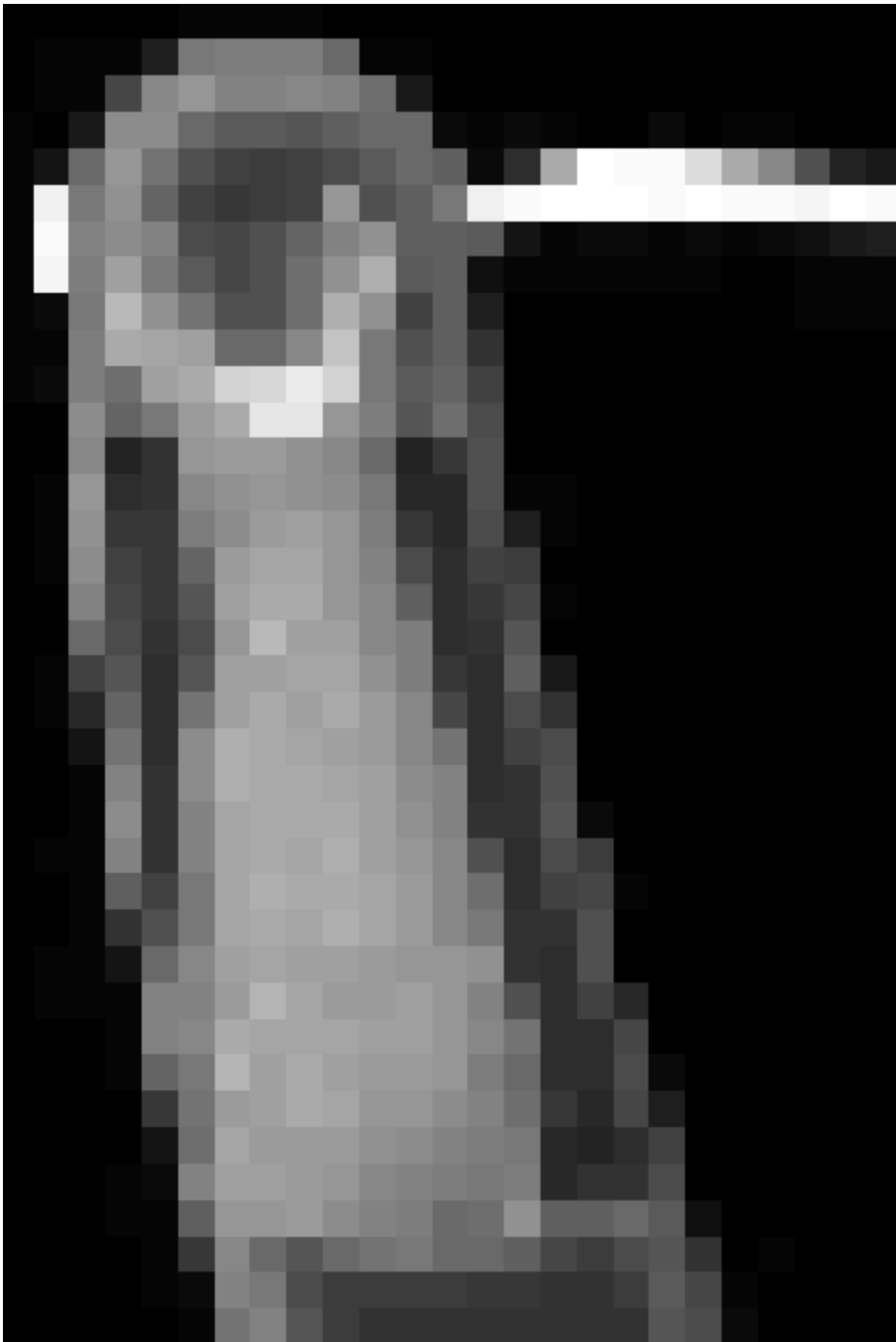
Non-Linear Diffusion Filter

This filter applies an anisotropic diffusion filter on the template and match image pyramid. Use this filter to reduce noise and to enhance the edge contrast. The following images illustrate that the Non-Linear Diffusion filter can reduce the noise without sacrificing edge contrast.

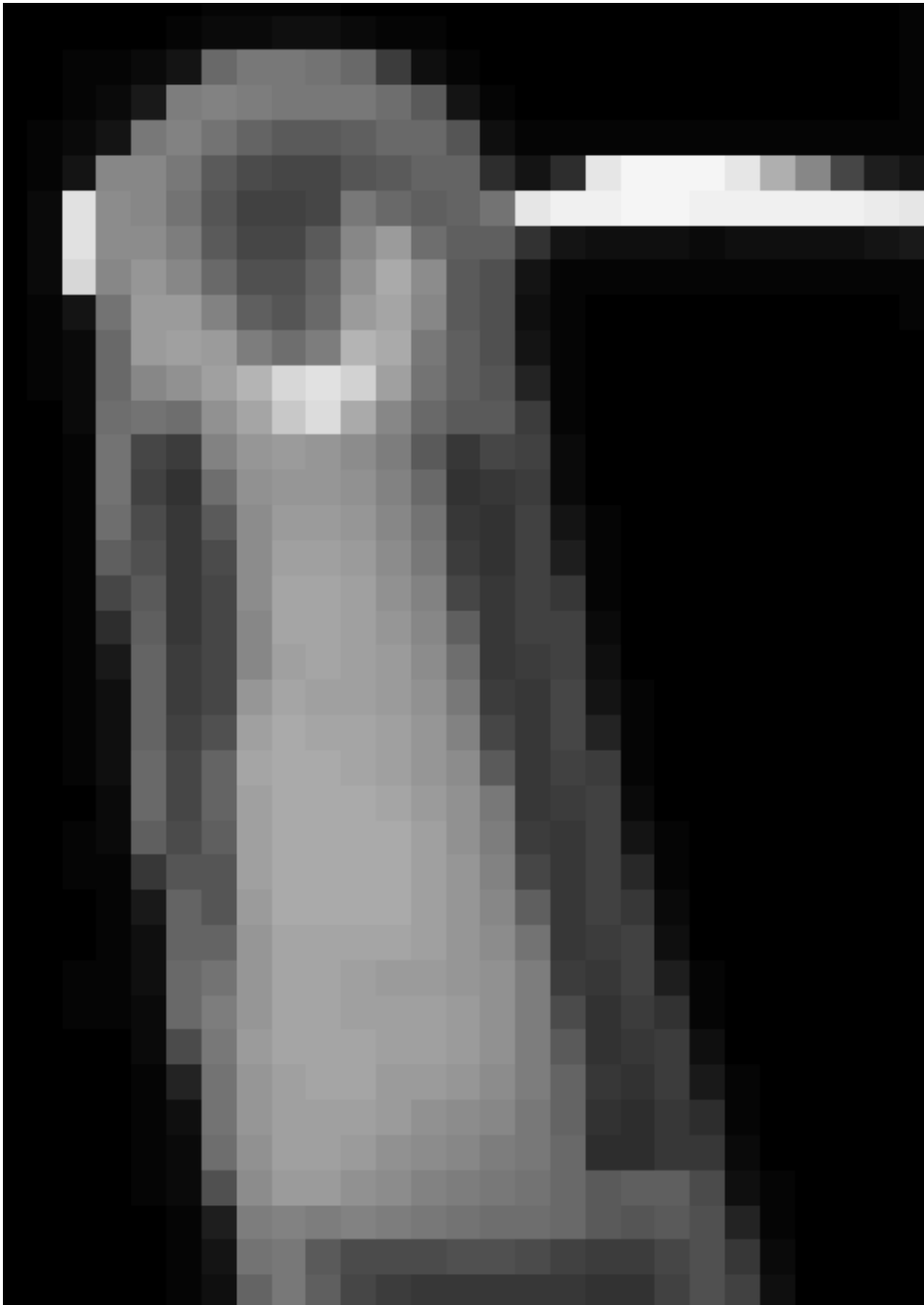
The illustration shows the Original Image:



The illustration shows the Gaussian Pyramid at Level 2:



The illustration shows the Non-Linear Diffusion Pyramid at Level 2:



Visit the ***Anisotropic*** wiki page for more reference.

Presets

Presets provide an easy and descriptive method to set advanced parameters for different Pattern Matching algorithms. Pattern Matching algorithms have advanced parameter settings which fine tune the algorithm to perform improved matching with different match requirements. Presets are the set of advanced parameter values which

have been tested to work with a broad range of match requirements.

A particular Preset (set of advanced parameter values) will be stored in a template image based on the chosen requirement. The values will be used automatically during matching. Hence, the matching results would be improved over the default results without having to understand the advanced parameters and their implications for matching. The Presets stored in the template is based on the selection of **Use-Case** and **Priority**. These inputs should be selected based on the match requirement.

The following options are provided in **Use-Case**:

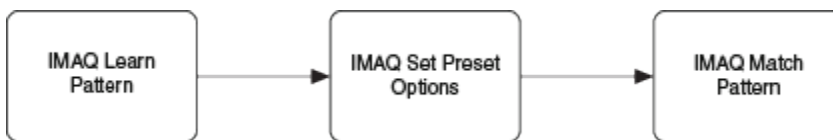
Preset Option	Description
Overlapping	Uses Advanced Options appropriate for match image that contain overlapping objects.
Overlapping	Uses Advanced Options appropriate for match image that contain overlapping objects.
Low Contrast	Uses Advanced Options appropriate for templates with large dimensions.
Screenshot	Uses advanced options appropriate for screenshots or screen captures that are pixel accurate in resolution (they are not captured via a camera) and usually have templates with small dimensions. The match algorithm used should be Grayscale Value Pyramid or Low Discrepancy Sampling when the Screenshot Preset is selected.

The following options are provided in **Priority**:

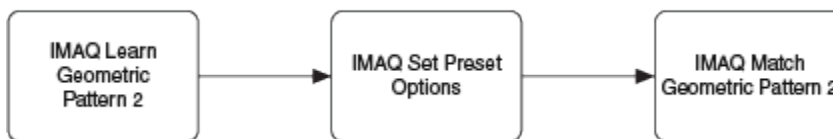
Preset Option	Description
---------------	-------------

Accurate	Uses Advanced Options that give a higher priority to Match Accuracy than Match Speed.
Fast	Uses Advanced Options that gives equal priority to Match Accuracy and Match Speed.
Very Fast	Uses Advanced Options that gives a higher priority to Match Speed than Match Accuracy.

The illustration shows the Workflow for Low Discrepancy, Grayscale, and Gradient Pyramid Algorithms:



The illustration shows the Workflow for Geometric Pattern Matching:



Sub-pixel Refinement

In both pyramidal as well as low-discrepancy sampling-based pattern matching, the user can choose to subject the refined match candidates to one last stage of refinement to find sub-pixel accurate locations and sub-degree accurate angles. This stage relies on specially-extracted edge and pixel information from the template and employs interpolation techniques to get a highly accurate match location and angle.

Once the refined locations (with or without sub-pixel refinement) are obtained, both pattern matching methods do a final and accurate score computation using most of the significant information present in the template.

Related information:

- [Anisotropic](#)

In-Depth Discussion 20

This section provides additional information you may need for building successful searching applications.

Normalized Cross-Correlation

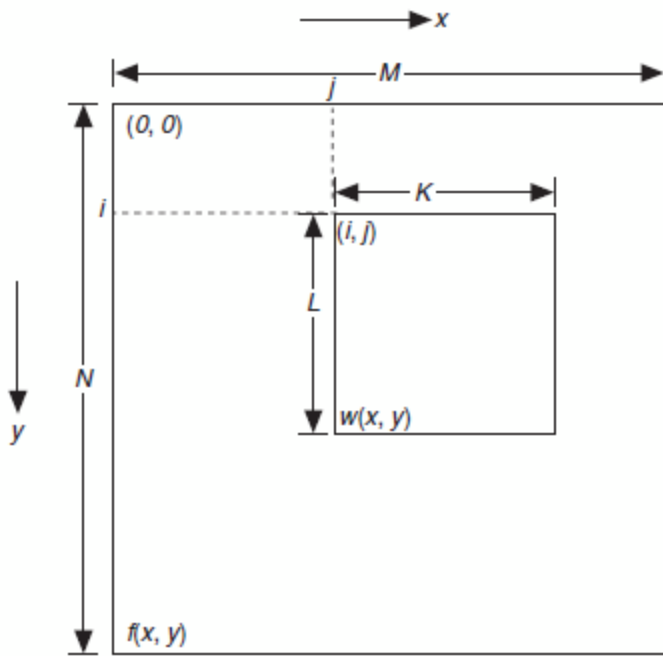
The following is the basic concept of correlation: Consider a subimage $w(x, y)$ of size $K \times L$ within an image $f(x, y)$ of size $M \times N$, where $K \leq M$ and $L \leq N$. The correlation between $w(x, y)$ and $f(x, y)$ at a point (i, j) is given by .

$$C_{(i, j)} = \sum_{x=0}^{L-1} \sum_{y=0}^{K-1} w(x, y) f(x+i, y+j)$$

where,

- $i = 0, 1, \dots, M - 1, j = 0, 1, \dots, N - 1,$
- the summation is taken over the region in the image where w and f overlap.

The following figure illustrates the correlation procedure. Assume that the origin of the image f is at the top left corner. Correlation is the process of moving the template or subimage w around the image area and computing the value C in that area. This involves multiplying each pixel in the template by the image pixel that it overlaps and then summing the results over all the pixels of the template. The maximum value of C indicates the position where w best matches f . Correlation values are not accurate at the borders of the image.



Basic correlation is very sensitive to amplitude changes in the image, such as intensity, and in the template. For example, if the intensity of the image f is doubled, so are the values of c . You can overcome sensitivity by computing the normalized correlation coefficient, which is defined as:

$$R(i, j) = \frac{\sum_{x=0}^{L-1} \sum_{y=0}^{K-1} (w(x, y) - \bar{w}) (f(x+i, y+j) - \bar{f}(i, j))}{\left(\sum_{x=0}^{L-1} \sum_{y=0}^{K-1} (w(x, y) - \bar{w})^2 \right)^{\frac{1}{2}} \left(\sum_{x=0}^{L-1} \sum_{y=0}^{K-1} (f(x+i, y+j) - \bar{f}(i, j))^2 \right)^{\frac{1}{2}}}$$

where w (calculated only once) is the average intensity value of the pixels in the template w . The variable f is the average value of f in the region coincident with the current location of w . The value of R lies in the range -1 to 1 and is independent of scale changes in the intensity values of f and w .

Advanced Pattern Matching Concepts

Introduction

Advanced pattern matching contains low-level learning and matching options that enable you to customize the pattern matching algorithm to your specific machine

vision application. In order to be useful to a broad audience, a pattern matching algorithm needs to process numerous diverse template images and search images. Determining a set of default advanced pattern matching options that is optimal for all possible applications is improbable. However, you can customize the IMAQ Vision pattern matching algorithm for your application by configuring several advanced options that affect the speed and accuracy of pattern matching.

When To Use

If the pattern matching portion of your application is not working as expected, make sure you have defined a template with the following qualities:

- Good feature detail,
- Adequate positional data,
- Sufficient background information,
- Appropriate level of asymmetry (for rotation-invariant matching).

You can change the template slightly to remove excessive background information or include additional feature information. You may also consider restricting the angle ranges, if possible, or defining a search region. In addition, make sure your application is running in optimal lighting conditions and that you are acquiring quality images. Nonuniform lighting and poor image quality adversely affect other image processing algorithms as well as pattern matching. If you have suitable template images and search images but want to improve the speed and/or accuracy of the pattern matching process, you can configure the advanced pattern matching options.

Pattern Matching Phases

Pattern matching consists of two stages: a learning stage, which is usually performed offline, and a matching stage. For the purpose of discussing advanced pattern matching options and the consequences of changing the default values, this document divides the matching stage into the following four phases.

- **Initial phase**—The first phase of shift- and rotation-invariant pattern matching. The algorithm makes steps larger than a pixel to locate potential matches.
- **Intermediate phase**—The second phase of rotation-invariant pattern matching. The algorithm makes coarse refinements to the location of matches found during the initial phase. Normally, this phase is skipped during shift-invariant matching.

- **Final phase**—The third, and often final, phase of shift- and rotation-invariant pattern matching. The algorithm makes small refinements to the matches.
- **Subpixel refinement phase**—The last phase of pattern matching if the **Enable Subpixel Accuracy** parameter is set to 1 (TRUE). The algorithm refines the matches to achieve subpixel and subangle accuracy.

Most advanced options influence only one phase of the pattern matching process. In most cases, you control one phase of the pattern matching process using multiple advanced options.



Note Because of the additional power and complexity of advanced pattern matching, suboptimal option values or sets of values can yield unpredictable results.

Guidelines for Using Advanced Options

Since the process of learning the template is handled offline, the learning stage has more time to optimize default options for a given template. Therefore, National Instruments highly recommends that you configure the advanced match options first to alter the speed and/or accuracy of pattern matching in your application.



Note Most advanced match options are available for both shift-invariant matching and rotation-invariant matching. However, the impact of an option on the matching process may differ considerably from shift-invariant matching to rotation-invariant matching.

If you still require better results after configuring the advanced match options, try changing the advanced learn options. The advanced learn options allow you to manage the amount of data used in different segments of the matching process. You have control of the number of data points and/or the angular accuracy of the data points while the algorithm selects the points.

The results you get from changing the advanced learn options vary significantly from one template to the next. At times, changing the options might produce undesired results. For example, requesting more data points for a template could result in less accurate matches because too many data points cause the algorithm to sample image noise. This scenario is not unusual or unexpected.

Advanced Match Options

The following list describes the advanced options you can configure for the match process.

- **Minimum Contrast**—Specifies the minimum contrast value a potential match region must have to contain a match. When used with templates having high contrast, this option improves speed by excluding regions of an image.
- **Enable Subpixel Accuracy**—Enables the subpixel refinement phase applied to matches at the end of the final phase. When the correct matches are found but their location is less accurate than expected, set this option to 1 (TRUE).
- **Subpixel Iterations**—Defines the number of refinements performed by the match process using the subpixel information stored in the template. These refinement iterations are applied to the number of matches requested.
- **Subpixel Tolerance**—Specifies the control tolerance used during subpixel refinement to stop processing when a match location has been improved to the given accuracy. This option can improve the speed of the algorithm when you have multiple matches because the amount of refinement for each match varies with the accuracy of its location. For shift-invariant matching, this option represents the tolerance, in pixels, for position. For rotation-invariant matching, this option represents the tolerance, in degrees, for angular accuracy and indirectly sets a lower tolerance using radian distances, in pixels, for position.
- **Initial Match List Length**—Specifies the number of match regions cached from the initial phase of matching. The match algorithm focuses on these regions in later processing. If the template is very distinct in the search images, reducing this length improves speed significantly. If the application is missing matches, increasing this value may solve the problem but processing becomes slower.
- **Match List Reduction Factor**—Controls how quickly the match list is shortened from one matching phase to the next. The value is a divisor of the list length. For example, a value of 2 cuts the list in half. If you increase the list length with Initial Match List Length, you can recover some speed by increasing the Match List Reduction Factor. In most instances, reduce the Match List Reduction Factor to keep more match regions for later processing, which increases accuracy but decreases speed.
- **Initial Step Size**—Specifies the number of pixels to shift the template across the inspection image during the initial phase of matching. The optimal step size is computed during the learning phase, stored with the template, and used by default. You can reduce the step size to improve match accuracy, but doing so

greatly reduces speed. Do not increase this value to make larger steps.

- **Intermediate Angular Accuracy**—Establishes the angular accuracy, in degrees, used when refining rotated matches from the initial phase. Decreasing this value causes higher accuracy in resulting matches but slows processing. If angular accuracy is not important to the application or the template is symmetrical, increase this value to get faster matching with less accurate angles. The angular accuracy value is rounded down to a value that evenly divides 360 and provides at least the accuracy requested.
- **Search Strategy**—Specifies the matching strategy used to find the matches. The default search strategy is Balanced (2), which initially searches the image using a medium step size to improve speed and then uses the match refinement of the Conservative strategy (1) to maintain accuracy. The Conservative strategy initially searches the image with a small step size, which reduces speed but provides very accurate matches. In general, avoid the Aggressive strategy (3) unless the center of the template does not contain distinguishing characteristics or the template is highly rectangular (3:1 or more in dimensions).

Advanced Learn Options

The following list describes the advanced options you can configure for the learning process.

- **Initial Sample Size**—Specifies the size of the sample, in pixels, used during the initial phase of matching. When distinguishing template characteristics are not defined by the edges in the template, increase this value to promote nonedge characteristics during matching. A larger sample size reduces the match speed. Increase the size by a multiple of 12 for the best possible match speed; a size of 60 is optimal. If you have a large template (for example, a template whose dimensions exceed 200 pixels), try increasing the sample size to improve accuracy. Increasing the sample size has less of an impact on match speed with large templates than with small templates.
- **Initial Sample Size Factor**—Determines the size of the sample, in percent of total pixels, used during the initial phase of matching. Defining a sample size as a percent of total pixels facilitates consistent sampling across templates of different dimensions.
- **Final Sample Size**—Specifies the size of the sample, in pixels, used during the final phase of matching. When edges define the most unique template characteristics, increase this value to promote edge characteristics during matching. Increase the

size by a multiple of 12 for best match speed; a size of 60 is optimal.



Note A large Final Sample Size does not reduce match speed as much as a large Initial Sample Size.

If you have a large template (for example, a template whose dimensions exceed 200 pixels), increasing this sample size improves match accuracy with only small speed degradation.

- **Final Sample Size Factor**—Determines the size of the sample, in percent of total pixels, used during the final phase of matching. Defining a sample size in percent of total pixels facilitates consistent sampling across templates of different dimensions.
- **Subpixel Sample Size**—Specifies the size of the sample, in pixels, used during the subpixel refinement phase of matching. The default sample size that the algorithm determines during the learning process is conservative—slightly larger than the initial or final sample sizes. Increasing this size may deteriorate match accuracy. Although this sample is used primarily during subpixel refinement, it is used also during final match processing. Therefore, the Subpixel Sample Size affects matching even when Enable Subpixel Accuracy is set to 0 (FALSE).
- **Subpixel Sample Size Factor**—Determines the size of the sample, in percent of total pixels, used during the subpixel refinement phase of matching. Defining a sample size in percent of total pixels facilitates consistent sampling across templates of different dimensions.
- **Initial Angular Accuracy**—Establishes the angular accuracy supported during the initial phase of rotation-invariant matching. Use the advanced match option Intermediate Angular Accuracy to limit the angular accuracy unless you do not need the template to produce matches at this angular accuracy. Reducing the initial accuracy reduces the template size by less than a third.
- **Final Angular Accuracy**—Establishes the angular accuracy supported during the intermediate and final phases of rotation-invariant matching. As with Initial Angular Accuracy, increasing this value reduces the final accuracy by restricting the template from generating more accurate match results regardless of the match options used. However, reducing the accuracy significantly shrinks the template size by as much as half.
- **Initial Step Size**—Defines the initial jump size to support when searching for the template. Because this value depends entirely on the pixel content and other special characteristics of the template, avoid changing this value. If you set a lower

Initial Step Size, the initial match phase rigorously searches more match regions located closer together. A better approach to achieving a rigorous initial search is to set Search Strategy Support to 1 (Conservative). This lowers the step size to an optimal value based on the template and automatically adjusts other options according to that value.

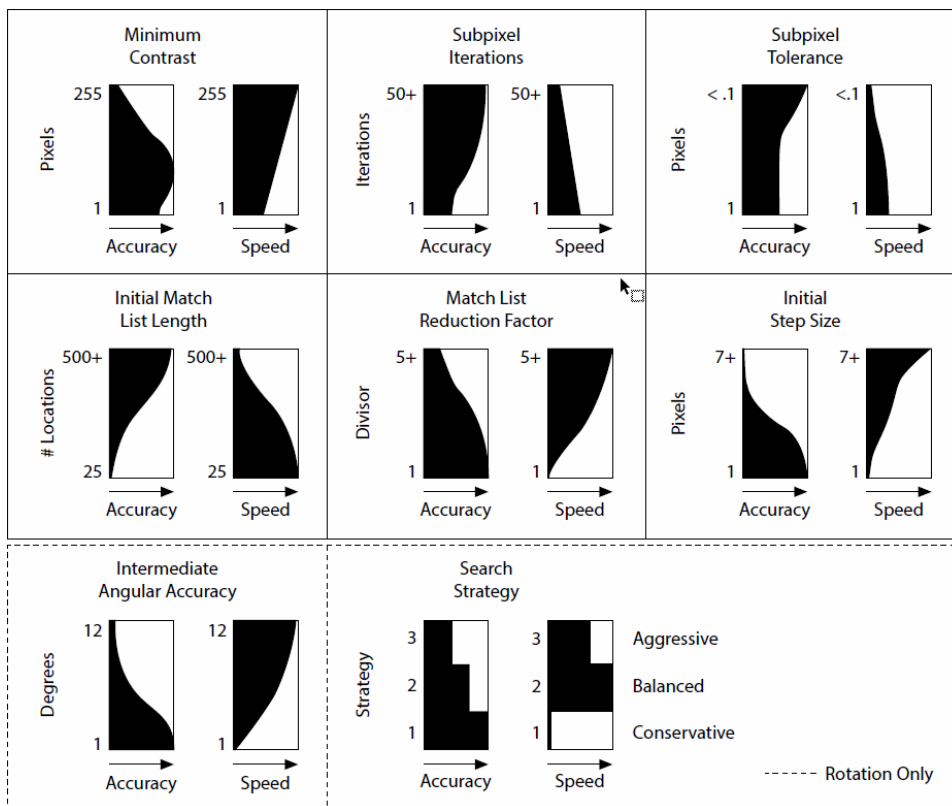


Note The value of Initial Step Size is a recommendation for the learn process. The actual step size may be smaller based on the content of the template image.

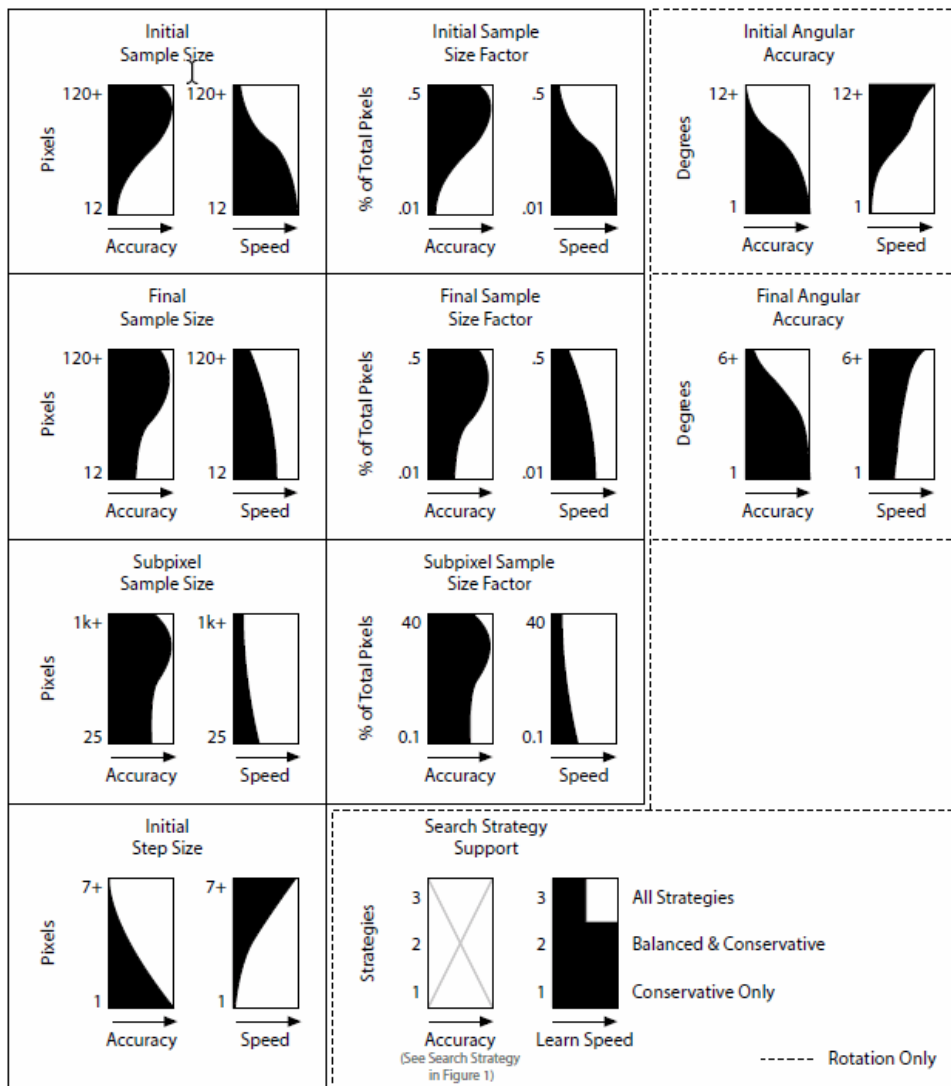
- **Search Strategy Support**—Defines the set of search strategies supported by the template during matching. The Balanced and Conservative search strategies are supported by default. If you intend to use this template with the advanced match option Search Strategy set to 3 (Aggressive), set Search Strategy Support to 3 (Aggressive, Balanced & Conservative). Set this option to 1 (Conservative) if you are performing matching only with the Conservative search strategy and want to minimize the template size.

Options at a Glance

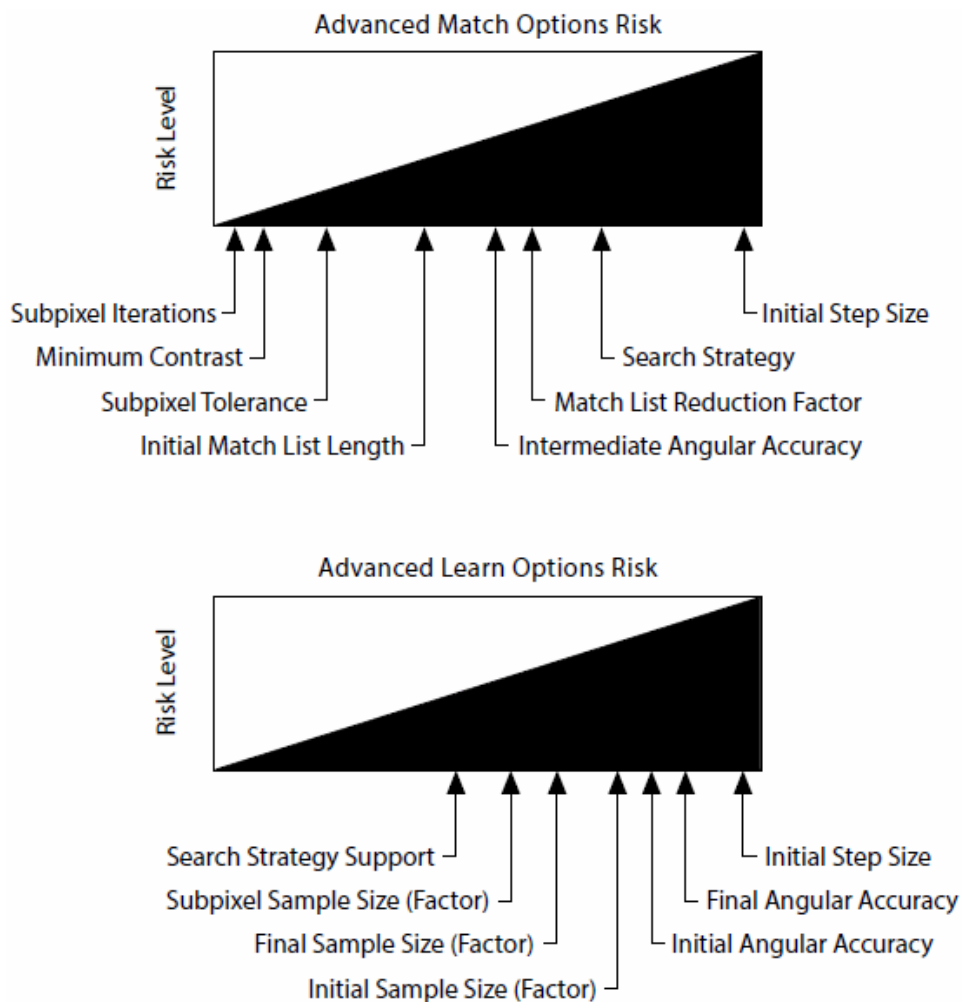
This section illustrates how the advanced pattern matching options effect the accuracy, speed, and risk of pattern matching in your application. The following illustration shows the impact that each advanced match option has on the accuracy and speed of the matching process.



The following figure shows the impact that each learn option has on the accuracy and speed of the matching process.



The following figure illustrates the risk of getting unexpected results when configuring the advanced match and learn options. The risk levels shown for each option correspond to setting more aggressive values for the option.



Optimizing for Accuracy

Many advanced options impact the accuracy of match results. Accuracy may refer to the following:

- Order in which multiple matches are returned,
- Number of matches returned,
- Match location(s) returned.

Each of these conditions requires a different usage of the advanced options. In general, if you find a match with a lower score than expected or an inaccurate position, use subpixel refinement to improve the results. First, set Enable Subpixel Accuracy to 1 (TRUE) to increase accuracy. Then, increase the number of subpixel iterations with Subpixel Iterations if you need additional improvement.

If only one of several matches is inaccurate, you can set Subpixel Tolerance to the pixel

accuracy desired, which is normally between 0.1 and 0.5. By setting this value, the pattern matching algorithm spends time improving only those matches outside the specified tolerance instead of refining all matches for the given number of iterations. When providing a tolerance, set Subpixel Iterations to 0 (default) to allow the refinement process choose when to stop, or set Subpixel Iterations to a maximum iteration to cap the amount of refinement performed when the tolerance is unreachable.

Altering subpixel refinement options is the last phase of the matching process you can manipulate to improve accuracy. If altering subpixel refinement options does not result in the expected match accuracy, you can adjust one or more of the following options. Determining which options to adjust varies among applications, but the following categorization of options provides a general order in which to proceed.

- Basic match options—Shift/rotation invariance, **Minimum Contrast**, **Rotation Angle Ranges**:
 - Low risk relative to other advanced options.
 - Can significantly impact match results by impacting the entire match process.
- Advanced match options used in the intermediate and final phases of the match process—**Intermediate Angular Accuracy**, **Match List Reduction Factor**:
 - Usually low risk because they occur during pixel-level refinement,
 - Impact on match speed is reduced because these options affect a later match phase.
- Advanced match options used in the initial phase of the match process— **Initial Match List Length**, **Initial Step Size**:
 - Higher risk because these options alter match selection from the start,
 - Slower match speed when more conservative values are selected because of additional processing required.



Note Changing **Initial Step Size** is not recommended because the template contains the optimal initial step size to use. If you want to perform an exhaustive search for the template, you can set this option to 1, but doing so significantly reduces matching speed.

- Advanced match option defining the match process—**Search Strategy**: Aggressive, Balanced (default), and Conservative:
 - High risk because changing this option alters most advanced match options and sometimes alters the initial search approach.

- Conservative works like Balanced but is more rigorous during the initial phase, causing match speeds to double in many cases.
- Aggressive uses a different initial searching technique, so resulting matches may differ significantly from the other strategies. Use the Aggressive option when searching on very rectangular templates or when many of the distinguishing characteristics are not in the center of the template.
- Advanced learn options used during the subpixel refinement phase—**Subpixel Sample Size, Subpixel Sample Size Factor:**
 - High risk relative to other advanced match options; low risk relative to other advanced learn options.
 - By default, Subpixel Sample Size is based on the size of the template, in pixels. The following Tables show the relationship between template size and Subpixel Sample Size. The first table depicts the sizes for Shift-Invariant Matching while the second table depicts the sizes for Rotation-Invariant Matching.

Template Size (in pixels)	Subpixel Sample Size
greater than or equal to 600	60
greater than or equal to 2400	120
greater than or equal to 9600	240
greater than or equal to 42000	420
greater than or equal to 160000	600

Template Size (in pixels)	Subpixel Sample Size
greater than or equal to 200	60
greater than or equal to 1200	120
greater than or equal to 4800	240
greater than or equal to 16800	420
greater than or equal to 60000	600
greater than or equal to 160000	840



Note The subpixel refinement phase is very sensitive to the sample size; therefore, do not change the default unless your application requires the change. For example, increase the sample size if you have large templates containing a lot of detailed information.

- Advanced learn options used during the final phase of matching—**Final Sample Size, Final Sample Size Factor, Final Angular Accuracy**:
 - High risk relative to other advanced match options; medium risk relative to other advanced learn options.
 - Final sample is used in the final phase of the match process and contains data for refining match locations to within a pixel or degree of accuracy. This sample includes, but is not limited to, locations around edges in the template.
 - By default, the final sample produces an angular accuracy of 1°. If you do not require this level of accuracy, you can increase the **Final Angular Accuracy** value, which reduces accuracy, reduces the template size, and increases the match speed. The accuracy value is always rounded down to a value that divides 360 evenly.
- Advanced learn options used during the initial phase of matching—**Initial Sample Size, Initial Sample Size Factor, Initial Angular Accuracy** :
 - Among the highest risk advanced options because they affect the entire match process from start to finish.
 - Initial sample contains representatives from regions having roughly the same pixel values.
 - By default, the initial sample produces an angular accuracy of 6°. This initial accuracy default is necessary to find matches in the final phase of matching with an accuracy of 1°. If you set the final accuracy to be less accurate, you can increase the **Initial Angular Accuracy** value to reduce accuracy, as well. The accuracy value is always rounded down to a value that divides 360 evenly.

Object Tracking

This section contains information about object tracking.

Introduction

Object tracking tracks the location of an object over a sequence of images. It is a

method of following the object through successive frames to determine how it is moving relative to other objects in the image.

When using object tracking, the user must locate and specify the object to be tracked. The object tracking methods then track the object in each acquired frame the object is present in.

When to Use

Object tracking is an essential machine vision function, and has many uses in the following application areas:

- Security and surveillance—In the surveillance industry, objects of interest such as people and vehicles can be tracked. Object tracking can be used for detecting trespassing or observing anomalies like unattended baggage.
- Traffic management—The flow of traffic can be analyzed, and collisions detected.
- Medicine—Cells can be tracked in medical images.
- Industry—Defective items can be detected and tracked.
- Robotics and navigation—Robots can follow the trajectory of an object. Robotic assistance can maneuver in a factory (de-palletizing objects).
- Human-computer interaction (HCI)—Users can be tracked in a gaming environment.
- Object modeling—An object tracked from multiple perspectives can be used to create a partial 3D model of the object.
- Bio-mechanics—Tracking body parts to interpret gestures or movements.

What to Expect from Object Tracking

A well-configured object tracking application tracks objects regardless of blur, noise, or partial occlusion of the object. Object tracking in Vision is tolerant of gradual changes in the tracked object, including geometric transformations such as shifting, rotation, or scaling. Object tracking in Vision can be used in grayscale (U8, U16, and I16) and color (RGB32).

Object Tracking Techniques

NI Vision implements two object tracking algorithms:

- **Mean shift**—A simple algorithm that tracks the user-defined objects by iteratively updating the location of the object.
- **EM-based mean shift (shape adapted mean shift)**—An extended version of the mean shift algorithm in which not only the location but also the shape (including scale) of the object is adapted frame after frame.

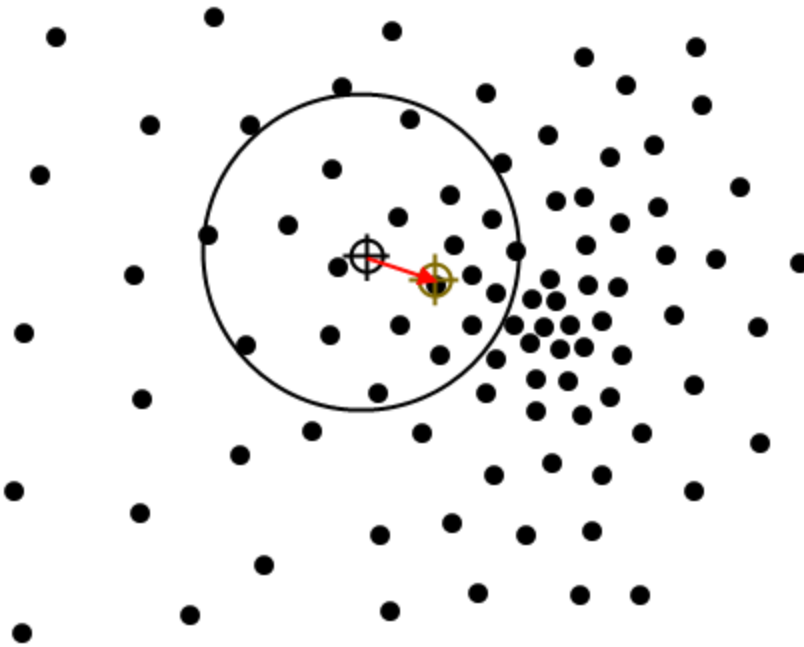
To track an object, the target object must first be characterized over a feature space. The color histogram is a very robust representation of the object appearance, and is chosen as the feature space. Moving objects are characterized by their histograms. The feature-histogram-based target representations are regularized by spatial masking with an isotropic kernel.

Understanding Mean Shift

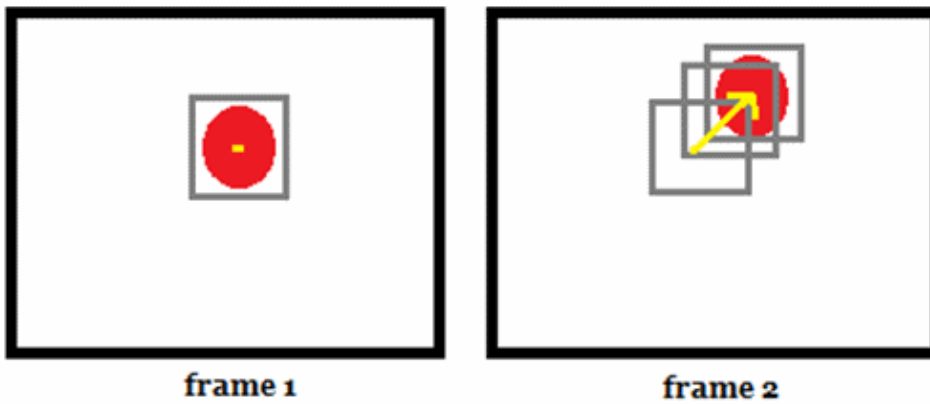
The mean shift algorithm is a simple method for finding the position of a local mode (local maximum) of a kernel-based estimate of a probability density function. Object tracking for an image frame is performed by a combination of histogram extraction, weight computation and derivation of new location.

There are three stages to the mean shift algorithm:

- **Target model**—Choose the target object in the given frame. Represent the target model in the given feature space (color histogram) with a kernel.
- **Mean shift convergence**—In the next frame, search with the current histogram and spatial data for the best target match candidate by maximizing the similarity function. In the mean shift algorithm, the object center moves from current location to a new location as shown in the figure below. The kernel is moved until the convergence of the similarity function, then the location of the object is updated.



- **Update location and model**—Update the target model, and the location of the target, based on the blending parameter.



Understanding EM-Based Mean Shift

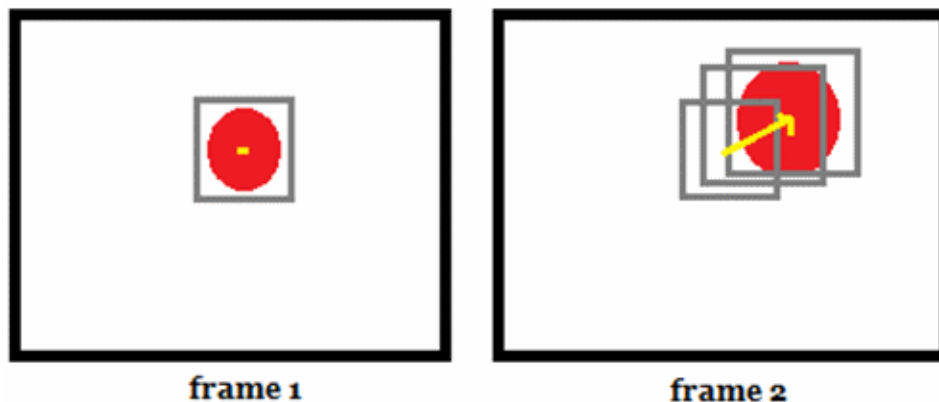
The mean shift algorithm is not scale or geometric-shift invariant. To track an object that may appear to change in size or shape, the EM-based mean shift algorithm is required.

The EM-based mean shift, or shape adapted mean shift, algorithm is an extension of the standard algorithm already described. The EM-based mean shift algorithm simultaneously estimates the position of the local mode and the covariance matrix that describes the approximate shape of the local mode. The covariance matrix that defines the shape and scale of the region (that defines the object) is updated every

frame to adapt to the shape and scale of the object in that frame.

There are three stages to the mean shift algorithm:

- **Target model**—Choose the target object in the given frame. Represent the target model in the given feature space (color histogram) with a kernel.
- **Mean shift convergence**—In the next frame, search with the current histogram and spatial data for the best target match candidate by maximizing the similarity function. In the mean shift algorithm, the object center moves from current location to a new location, essentially the center of mass, as shown in the figure below. The magnitude and direction of the move is represented by the mean shift vector. The kernel is moved until the convergence of the similarity function, then the location of the object is updated along with the covariance of the kernel.
- **Update location and model**—Update the target model (including the scale and shape), and the location of the target, based on the blending parameter and maximum acceptable scale and shape changes.



Kalman Prediction

EM-based mean shift also features a Kalman Filter implementation. A Kalman filter uses the history of measurements of the target to build a model of the state of the system. The history of measurements is used to accurately predict the location of the target.

Histogram Back Projection

Back projection is one method used to improve the convergence of the target candidate's size and location with the actual size and location of an object. Back projection is a way of recording how well the pixels of a target candidate fit the

distribution of pixels that the target models. This allows the user to gauge how well the model of the object matches its appearance.

A histogram of an image known to contain the object of interest is created, and is then back projected over the image. Proper thresholding of the resulting image should isolate the object from the background.

Each pixel value in the resulting image represents the likelihood that the pixel is part of the object. The minimum pixel value of 0 indicates the pixel does not belong to the object, while the maximum value of 255 verifies that the pixel belongs to the object. This back projected image is a good indication of how well the tracking algorithm has been able to identify the pixels that belong to the object to be tracked.

Background Subtraction

A second method used to improve the convergence of the target model is background subtraction. This method is a process that extracts foreground objects in a particular scene. This helps reduce false positives and creates a better match between the target model and the target candidates.

Choosing the Right Parameters

The following parameters can be set by the user to create an object tracking applications suited to their needs:

- **Histogram bins**—Defines the number of bins needed to represent the histogram that characterizes the object. As the number of bins decreases, the number of colors that fall into a given range expands, thus subtle color differentiation will not be possible. Increasing the number of bins allows greater differentiation between very similar colors. Generally, using more bins results in faster matching. By default, 16 bins are used for grayscale images, while RGB images use 8 bins.
- **Blending parameter**—Defines the degree to which the target model is based on the previous frame. This parameter falls between 1 and 100. For very high values, the model relies heavily on the current frame. As a result, if the target object is occluded or out of frame, it will be unable to locate the object in the next frame. For very low values, the model relies heavily on the previous frame. As a result, the model will not adapt to new changes in the appearance of the object. This may be desired in surveillance applications where the target may be frequently occluded.

The default value is 10%.

- **Max iterations**—Specifies the maximum number of iterations until a match is found. Matching iterates until the similarity of the target object and target model converges, or the maximum number of iterations is reached. The default value is 15.

The following additional parameters can be used to configure the EM-based mean shift algorithm.

- **Max scale change**—The maximum percentage that the size of the region defining the object can change between frames.
- **Max rotation change**—The maximum number of degrees that the region defining the object can rotate between frames.
- **Max shape change**—The maximum percentage that the shape of the region defining the object can change between frames.

In-Depth Discussion

This section provides additional information you may need for building successful object tracking applications.

Target Model

Let $\{x_i^*\}_{i=1\dots n}$ be the pixel locations of the target model, centered at 0. We define a function $b: \mathbb{R}^2 \rightarrow \{1\dots m\}$ which associates the pixel at location x_i^* to the histogram bin that corresponds to the color of that pixel. The probability of the occurrence of the color u in the target model is derived by employing a convex and monotonically decreasing kernel profile k which assigns a smaller weight to the locations that are farther from the center of the target.

$$\hat{q}_u = C \sum_{i=1}^n k\left(\left(x_i^*\right)^2\right) \delta\left(b\left(x_i^*\right) - u\right)$$

where δ is the Kronecker delta function.

The normalization constant C is derived by imposing the condition:

$$\sum_u^m \hat{q}_u = 1$$

on the equation:

$$C = \frac{1}{\sum_{i=1}^n k\left(\left(x_i^*\right)^2\right)}$$

Target Candidate

Let $\{x_i\}_{i=1 \dots n_h}$ be the pixel locations of the target model, centered at y . Using the same kernel profile, the probability of the color u in the target candidate is given by,

$$\hat{q}_u = C \sum_{i=1}^n k\left(\left(x_i^*\right)^2\right) \delta\left(b\left(x_i^*\right) - u\right)$$

where C_h is the normalization constant.

Mean-Shift Convergence

The Bhattacharyya coefficient is a similarity function that is used to calculate the similarity between the target model and target candidate.

$$w_i = \sum_{u=1}^M \sqrt{\frac{\hat{q}_u}{p_u(y_0)}} \delta\left(b\left(x_i - u\right)\right)$$

The weights are recalculated every iteration using the above formula, followed by the update to the target model and candidates.

Updating the Model

- **Mean-Shift**—In the mean-shift tracking algorithm, the object center moves from the current location, y , to a new location, y_1 according to the mean-shift iteration equation:

$$\hat{y}_1 = \frac{\sum_{i=1}^{nh} x_i w_i g_i}{\sum_{i=1}^{nh} w_i g_i}$$

where,

- $g_i = g(\|y - x_i\|)$,
- $g(x) = k(x)$,
- $k(x)$ is the kernel function.
- **EM-Based Mean-Shift**—In this method, the position is calculated as described above. Additionally, the covariance is calculated with the following equation:

$$V_1 = \frac{\sum_{i=1}^{nh} \left(x_i - \hat{y}_1 \right) \left(x_i - \hat{y}_1 \right)^T w_i g_i}{\sum_{i=1}^{nh} w_i g_i}$$

Geometric Matching

This section contains information about geometric matching.

Introduction

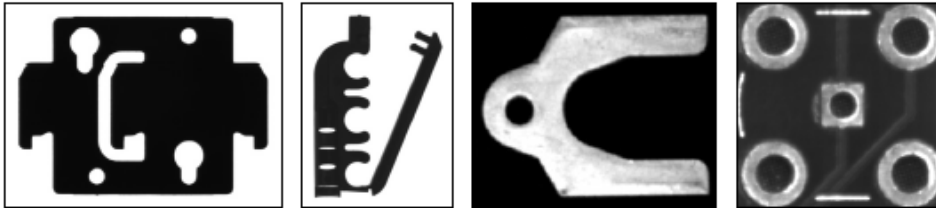
Geometric matching locates regions in a grayscale image that match a model, or template, of a reference pattern. Geometric matching is specialized to locate templates that are characterized by distinct geometric or shape information.

When using geometric matching, you create a template that represents the object for which you are searching. Your machine vision application then searches for instances of the template in each inspection image and calculates a score for each match. The score relates how closely the template resembles the located matches.

Geometric matching finds template matches regardless of lighting variation, blur, noise, occlusion, and geometric transformations such as shifting, rotation, or scaling of the template.

When to Use

Geometric matching helps you quickly locate objects with good geometric information in an inspection image. The following figure shows examples of objects with good geometric or shape information.



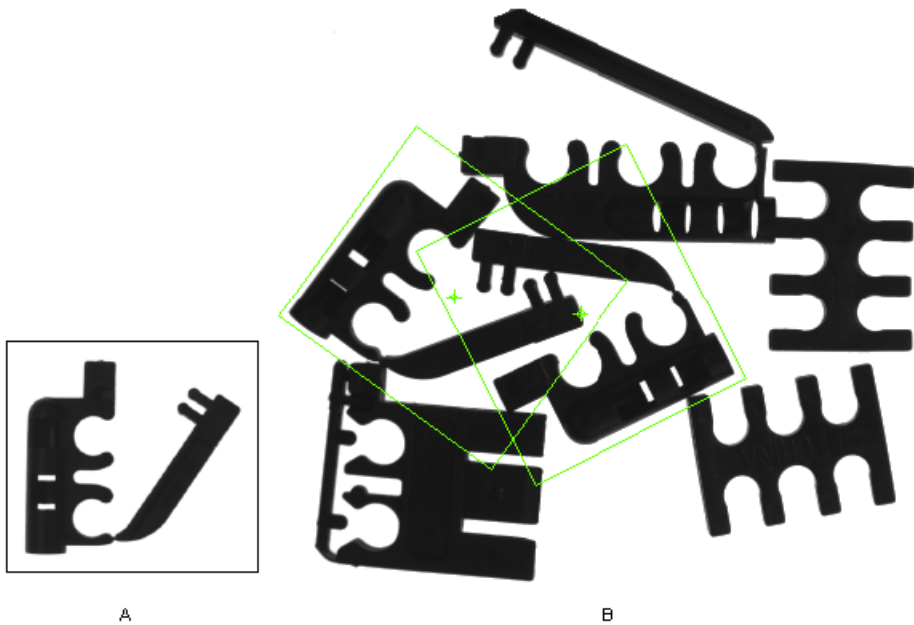
You can use geometric matching in the following application areas:

- **Gauging**—Measures lengths, diameters, angles, and other critical dimensions. If the measurements fall outside set tolerance levels, the object is rejected. Use geometric matching to locate the object, or areas of the object, you want to gauge. Use information about the size of the object to preclude geometric matching from locating objects whose sizes are too big or small.
- **Inspection**—Detects simple flaws, such as scratches on objects, missing objects, or unreadable print on objects. Use the occlusion score returned by geometric matching to determine if an area of the object under inspection is missing. Use the curve matching scores returned by geometric matching to compare the boundary (or edges) of a reference object to the object under inspection.
- **Alignment**—Determines the position and orientation of a known object by locating points of reference on the object or characteristic features of the object.
- **Sorting**—Sorts objects based on shape and/or size. Geometric matching returns the location, orientation, and size of each object. You can use the location of the object to pick up the object and place it into the correct bin. Use geometric matching to locate different types of objects, even when objects may partially occlude each other.

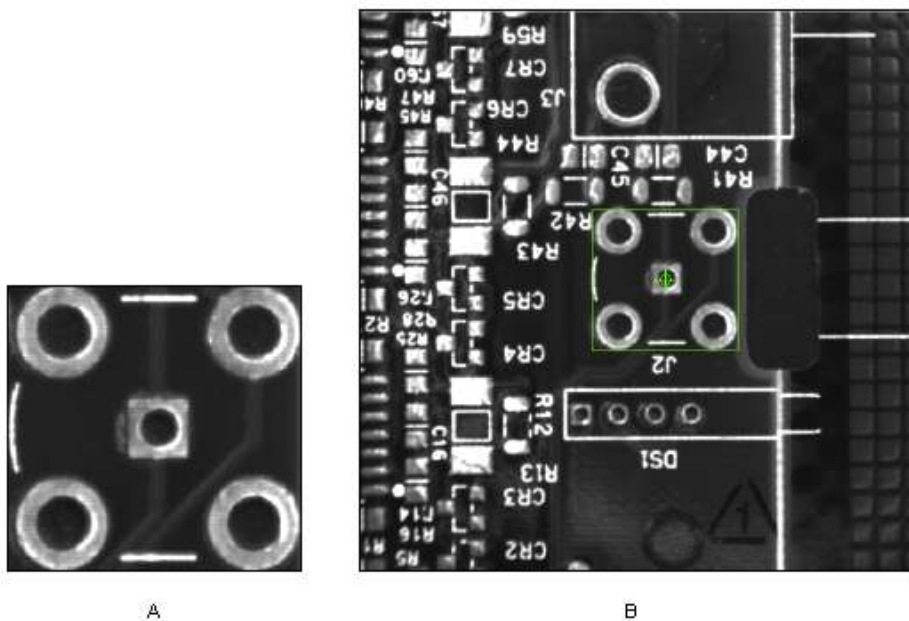
The objects that geometric matching locates in the inspection image may be rotated, scaled, and occluded in the image. Geometric matching provides your application with the number of object matches and their locations within the inspection image. Geometric matching also provides information about the percentage change in size (scale) of each match and the amount by which each object in the match is occluded.

For example, you can search an image containing multiple automotive parts for a particular type of part in a sorting application.

- Figure A shows an image of the part that you need to locate.
- Figure B shows an inspection image containing multiple parts and the located part that corresponds to the template.



The following figure shows the use of geometric matching in an alignment application.



When Not to Use Geometric Matching

The geometric matching algorithm is designed to find objects that have distinct geometric information. The fundamental characteristics of some objects may make other searching algorithms more optimal than geometric matching. For example, the template image in some applications may be defined primarily by the texture of an object, or the template image may contain numerous edges and no distinct geometric information. In these applications, the template image does not have a good set of features for the geometric matching algorithm to model the template. Instead, the pattern matching algorithm described in ***pattern matching***, would be a better choice.

In some applications, the template image may contain sufficient geometric information, but the inspection image may contain too many edges. The presence of numerous edges in an inspection image can slow the performance of the geometric matching algorithm because the algorithm tries to extract features using all the edge information in the inspection image. In such cases, if you do not expect template matches to be scaled or occluded, use pattern matching to solve the application.

Related concepts:

- [Pattern Matching](#)

What to Expect from a Geometric Matching Tool

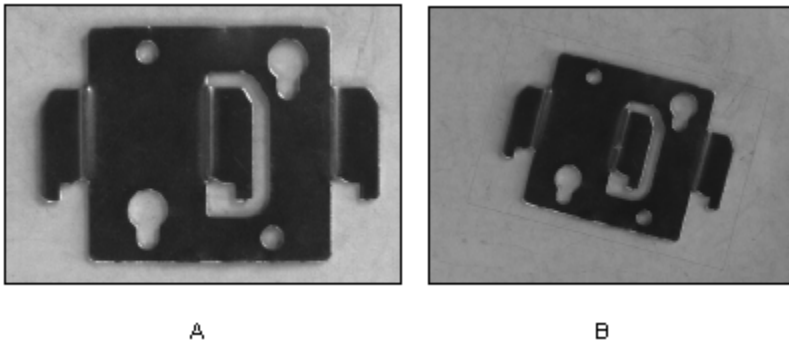
Because geometric matching is an important tool for machine vision applications, it must work reliably under various, sometimes harsh, conditions. In automated machine vision applications—especially those incorporated into manufacturing processes—the visual appearance of materials or components under inspection can change because of factors such as varying part orientation, scale, and lighting. The geometric matching tool must maintain its ability to locate the template patterns despite these changes. The following sections describe common situations in which the geometric matching tool needs to return accurate results.

Part Quantity, Orientation, and Size

The geometric matching algorithm can detect the following items in an inspection image:

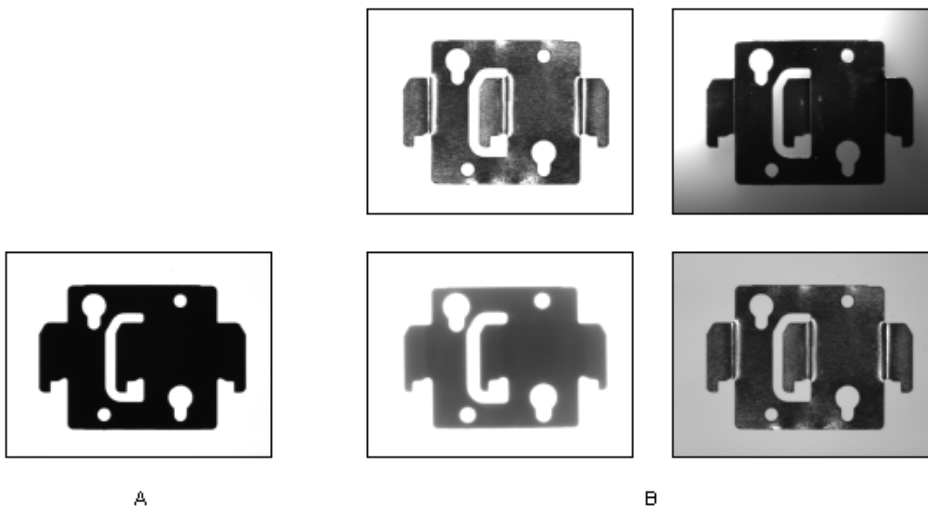
- One or more template matches,
- Position of the template match,
- Orientation of the template match,
- Change in size of the template match compared to the template image.

You can use the geometric matching algorithm to locate template matches that are rotated or scaled by certain amounts. Figure A shows a template image. Figure B shows the template match rotated and scaled in the image.



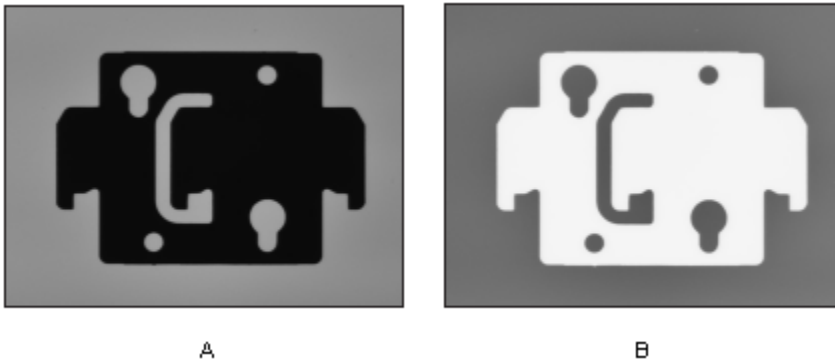
Non-Linear or Non-Uniform Lighting Conditions

The geometric matching algorithm can find a template match in an inspection image under conditions of non-linear and non-uniform lighting changes across the image. These lighting changes include light drifts, glares, and shadows. Figure A shows a template image. Figure B shows the typical conditions under which geometric matching correctly finds template matches.



Contrast Reversal

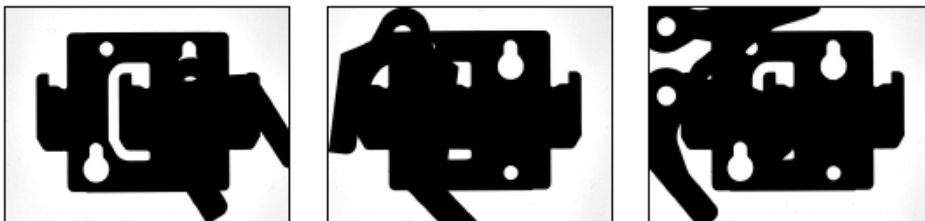
The geometric matching algorithm can find a template match in an inspection image even if the contrast of the match is reversed from the original template image. The following figure illustrates a typical contrast reversal. Figure A shows the original template image. Figure B shows an inspection image with the contrast reversed. The geometric matching algorithm can locate the part in figure B with the same accuracy as the part in figure A.



Partial Occlusion

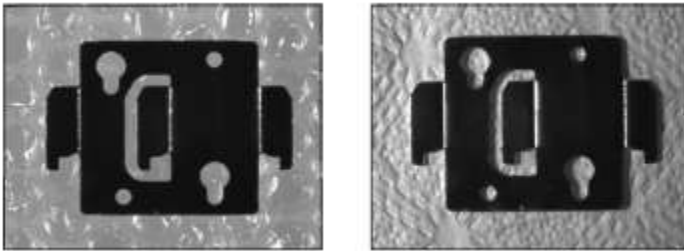
The geometric matching algorithm can find a template match in an inspection image even when the match is partially occluded because of overlapping parts or the part under inspection not fully being within the boundary of the image. In addition to locating occluded matches, the algorithm returns the percentage of occlusion for each match.

In many machine vision applications, the part under inspection may be partially occluded by other parts that touch or overlap it. Also, the part may seem partially occluded because of degradations in the manufacturing process. The following figure illustrates different scenarios of occlusion under which geometric matching can find a template match. The figure below represents the template image for this example.



Different Image Backgrounds

The geometric matching algorithm can find a template match even if the inspection image has a background that is different from the background in the template image. The following figure shows examples of geometric matching locating a template match in inspection images with different backgrounds. Figure A represents the template image for this example.



Geometric Matching Technique

Searching and matching algorithms, such as the pattern matching algorithm or geometric matching algorithm, find regions in the inspection image that contain information similar to the information in the template. This information, after being synthesized, becomes the set of features that describes the image. Pattern matching and geometric matching algorithms use these sets of features to find matches in inspection images.

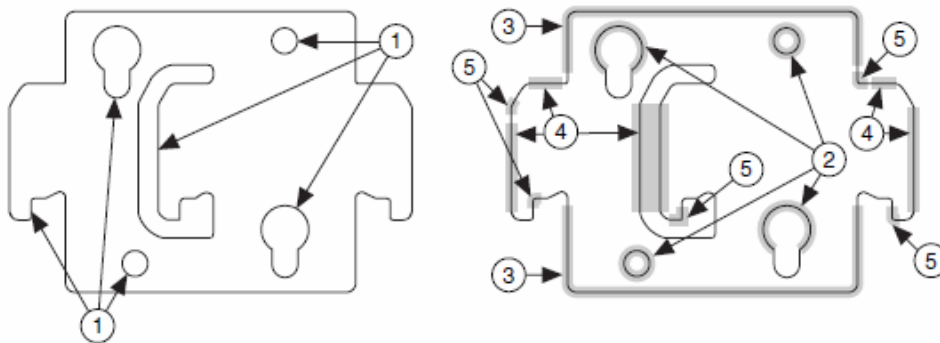
Pattern matching algorithms use the pixel intensity information present in the template image as the primary feature for matching. Geometric matching algorithms use geometric information present in the template image as the primary features for matching. Geometric features can range from low-level features, such as edges or curves, to higher-level features, such as the geometric shapes made by the curves in the image.

The geometric matching process consists of two stages: learning and matching. During the learning stage, the geometric matching algorithm extracts geometric information from the template image. The algorithm organizes and stores the information and the spatial relationships between these features in a manner that facilitates faster searching in the inspection image. In NI Vision, the information learned during this stage is stored as part of the template image.

During the matching stage, the geometric matching algorithm extracts geometric information from the inspection image that correspond to the information in the template image. Then, the algorithm finds matches by locating regions in the inspection image where features align themselves in spatial patterns similar to the spatial patterns of the features in the template.

Vision includes two geometric matching methods. Both geometric matching techniques rely on **curves extracted from image** to perform the matching. The two geometric matching techniques differ in how the curve information is used to perform the matching. The edge-based geometric matching method computes the gradient value of the edge at each point along the curves found in the image and uses the gradient value and the position of the point from the center of the template to perform the matching. The feature-based geometric matching method extracts geometric features from the curves and uses these geometric features to perform the matching.

The following figure shows the information from the template image that the geometric matching algorithm may use as matching features. Figure A shows the curves that correspond to edges in the template image. These curves form the underlying information that is used by the edge-based geometric matching technique. Figure B shows higher-level shape features that the feature-based geometric algorithm uses for matching. Refer to the Choosing The Right Geometric Matching Technique section to select the best geometric matching method for your application.



1. Curves
2. Circular Features
3. Rectangular Features
4. Linear Features
5. Corners

Curve Extraction

A curve is a set of edge points that are connected to form a continuous contour. Curves typically represent the boundary of the part in the image. In geometric matching, curves are the underlying information used to represent a template and to match the template in an inspection image. This section describes how curves are extracted from an image.

The curve extraction process consists of three steps: finding curve seed points, tracing the curve, and refining the curves.

Finding Curve Seed Points

A seed point is a point on a curve from which tracing begins. To qualify as a seed point, a pixel cannot be part of an already existing curve. Also, the pixel must have an edge contrast greater than the user-defined **Edge Threshold**. The edge contrast at a pixel is computed as a function of the intensity value at that pixel and the intensities of its neighboring pixels. If $P(i, j)$ represents the intensity of the pixel P with the coordinates (i, j) , the edge contrast at (i, j) is defined as:

$$\sqrt{\left(P_{(i-1, j)} - P_{(i+1, j)}\right)^2 + \left(P_{(i, j-1)} - P_{(i, j+1)}\right)^2}$$

For an 8-bit image, the edge contrast may vary from 0 to 360.

To increase the speed of the curve extraction process, the algorithm visits only a limited number of pixels in the image to determine if the pixel is a valid seed point. The number of pixels to visit is based on the values that the user provides for the **Row Step** and **Column Step** parameters. The higher these values are, the faster the algorithm searches for seed points. However, to make sure that the algorithm finds a seed point on all of the curves, **Row Step** must be smaller than the smallest curve in the y direction, and **Column Step** must be smaller than the smallest curve in the x direction.

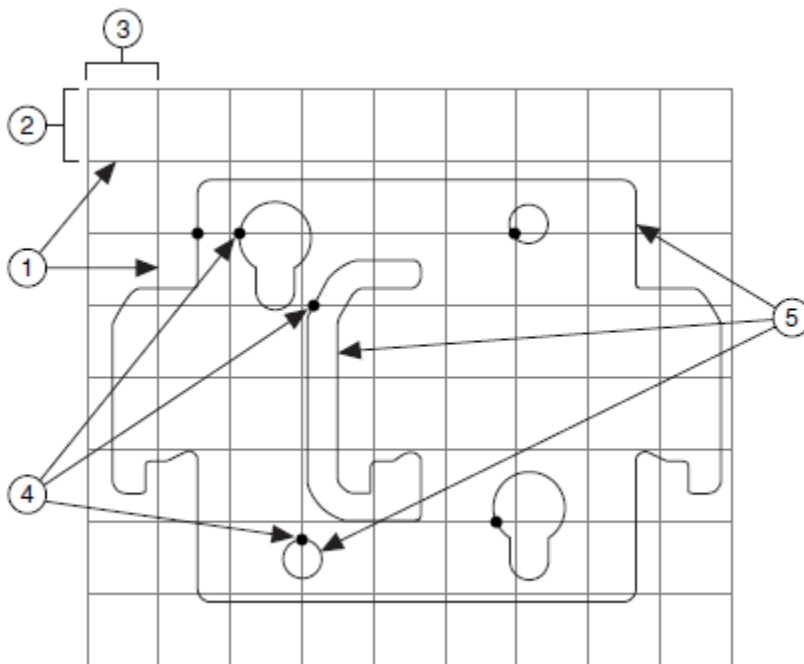
The algorithm starts by scanning the image rows from the top left corner. Starting at the first pixel, the edge contrast of the pixel is computed. If the edge contrast is greater than the given threshold, the curve is traced from this point. If the contrast is lower than the threshold, or if this pixel is already a member of an existing curve previously computed, the algorithm analyzes the next pixel in the row to determine if it qualifies

as a seed point. This process is repeated until the end of the current row is reached. The algorithm then skips **Row Step** rows and repeats the process.

After scanning all of the rows, the algorithm scans the image columns to locate seed points. The algorithm starts at the top left corner and analyzes each column that is **Column Step** apart.

Tracing the Curve

When it finds a seed point, the curve extraction algorithm traces the rest of the curve. Tracing is the process by which a pixel that neighbors the last pixel on the curve is added to the curve if it has the strongest edge contrast in the neighborhood and the edge contrast is greater than acceptable edge threshold for a curve point. This process is repeated until no more pixels can be added to the curve in the current direction. The algorithm then returns to the seed point and tries to trace the curve in the opposite direction. The following figure illustrates this process.



1. Scan Lines
2. Row Step
3. Column Step
4. Curve Seeds
5. Curves



Note To simplify the figure, **Row Step** and **Column Step** are not smaller than the smallest feature.

Refining the Curve

During the final stage of curve extraction, the algorithm performs the following tasks to refine the extracted curves:

- Combines curves into one large curve if their end points are close together.
- Closes a curve if the end points of the curve are within a user-defined distance of each other.
- Removes curves that fall below a certain size threshold defined by the user.

Edge-Based Geometric Matching

This section describes the learning and matching stages of the edge-based geometric matching technique. The edge-based technique utilizes the generalized Hough transform method for matching. The generalized Hough transform is an extension of the Hough transform to detect arbitrary shapes.⁵

Learning

The learning stage consists of two steps: edge point extraction and R-table generation.

Edge Point Extraction

During the edge point extraction stage, the algorithm detects curves in the image and computes the gradient value (θ) at each edge point along the contours. The gradient value specifies the orientation of the tangential line at each point along the contour.

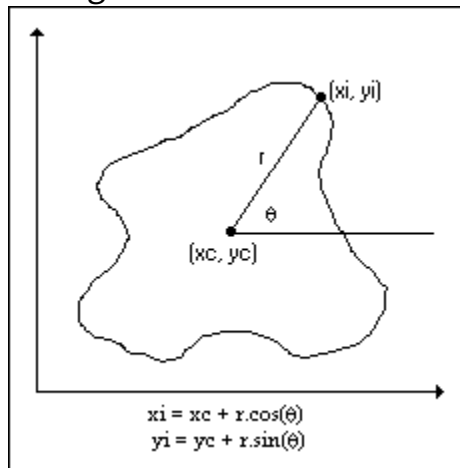
R-Table Generation

The generalized Hough transform uses a lookup table called an R-table to store the shape of the object. The R-table allows the generalized Hough transform to represent any arbitrary shape and does not require a parametric description of the object.

5. For more information about the Hough transform, see Ballard, D.H. "Generalizing the Hough Transform to Detect Arbitrary Shapes," Pattern Recognition 13, no. 2 (1981) 111-122.

The algorithm uses the following steps to compute the R-Table of a given shape (specified by the curves that are detected along the boundary of the shape).

1. The algorithm selects the center of the template image as the reference point (x_c, y_c) .
2. For each point (x_i, y_i) along the curves in the template image, the algorithm calculates the distance and orientation (r_i, θ_i) from the reference point as shown in the figure below:



3. The algorithm stores the (r_i, θ_i) value for each point in a R-table as a function of \emptyset , as shown in the following table:

Gradient Value (\emptyset)	r, θ Values
\emptyset_1	$(r_1, \theta_1), (r_4, \theta_4)$
\emptyset_2	$(r_2, \theta_2), (r_{10}, \theta_{10})$
\emptyset_n	$(r_n, \theta_n), (r_i, \theta_i)$

After the algorithm adds all points along the curves in the template image, the R-table represents the information that is learned from the template. The R-table can be used to regenerate the contour edge points and gradient angles at any point in the image during the matching phase.

An R-table stores the shift-invariant representation of the template object. Because each combination of scale and rotation requires a unique R-table, a template that allows variance in scale and rotation can occupy a large amount of memory. To reduce the size of the template and improve the speed of the matching process, NI Vision may sample the template image before computing the R-tables. By default, the software

automatically determines the sampling factor. Use the advanced learn options to manually specify a sampling factor.

Matching

The matching stage consists of three steps. The first step is edge point extraction, which is similar to the edge point extraction that occurs during the learning stage. The final two steps are generalized Hough matching and match refinement.

Edge Point Extraction

The edge points in the image are detected using the curve extraction process described in the learning section. If the size of the template image was reduced by sampling, then the inspection image is reduced by the same sampling factor before the curves are detected. The gradient value is computed and stored at each edge point along the detected curves.

Generalized Hough Matching

The matching process begins after the algorithm finds edge points and their gradient values in the inspection image. The matching process consists of the following steps:

1. The algorithm creates an accumulator, which stores candidate match locations in the inspection image.
2. The algorithm performs the following actions for each edge point (x, y) :
 - a. The algorithm uses the gradient value ϕ to index into the R-table and retrieve all the (r, θ) values.
 - b. The algorithm computes the candidate reference point for each (r, θ) value as follows:

$$x_c = x - r \cos(\theta)$$

$$y_c = y - r \sin(\theta)$$

- c. The algorithm increases the count in the accumulator for the location of the candidate reference point.
3. The algorithm finds the local peaks in the accumulator. These peaks represent possible match locations.

4. If matching for variation in rotation or scale, the algorithm builds an accumulator for each possible combination of rotation and scale, and performs steps 1–3 for each accumulator.
5. The algorithm processes the peaks in each accumulator to find the best matches.

Match Refinement

Match refinement is the final step in the matching stage. The algorithm uses curves extracted from both the template image and inspection image to ensure increased positional, scalar, and angular accuracy. Feature-Based Geometric Matching This section describes the learning and matching stages of the feature-based geometric matching technique.

Learning

Following curve extraction, the learning stage consists of two steps: feature extraction and representation of the spatial relationships between the features.

Feature Extraction

Feature extraction is the process of extracting high-level geometric features from the curves obtained from curve extraction. These features can be lines, rectangles, corners, or circles.

First, the algorithm approximates each curve using polygons. Then, the algorithm uses the line segments forming these polygons to create linear and corner features. These linear features are used to compose higher-level rectangular features. The curves or segments of curves that cannot be approximated well with polygons or lines are used to create circular features.

After the algorithm extracts high-level geometric features from the template image, the features are ordered based on the following criteria:

- Type—Lines, rectangles, corners, or circles
- Strength—How accurately the features portray a given geometric structure
- Saliency—How well the features describe the template

After the features have been ordered, the best are chosen to describe the template.

Representation of Spatial Relationships

Given two features, the algorithm learns the spatial relationship between the features, which consists of the vector from the first feature to the second feature. These spatial relationships describe how the features are arranged spatially in the template in relationship to one another. The algorithm uses these relationships to create a model of features that describes the template. The algorithm uses this template model during the matching stage to create match candidates and to verify that matches are properly found.

Matching

The matching stage consists of five main steps. The first two steps performed on the inspection image are curve extraction and feature extraction, which are similar to the curve extraction and feature extraction that occur during the learning stage. The final three steps are feature correspondence matching, template model matching, and match refinement.

Feature Correspondence Matching

Feature correspondence matching is the process of matching a given template feature to a similar type of feature in the inspection image, called a target feature. The algorithm uses feature correspondence matching to do the following:

- Create an initial set of potential matches in the inspection image.
- Update potential matches with additional information or refined parameters, such as position, angle, and scale.

Template Model Matching

Template model matching is the process of superimposing the template model from the learning step onto a potential match in the inspection image to confirm that the potential match exists or to improve the match. After superimposing the template model onto a potential match, the presence of additional target features found in accordance with the template model and its spatial relationships to existing features confirms the existence of the potential match and yields additional information that the algorithm uses to update and improve the accuracy of the match.

Match Refinement

Match refinement is the final step in the matching stage. Match refinement carefully refines known matches for increased positional, scalar, and angular accuracy. Match refinement uses curves extracted from both the template image and inspection image to ensure that the matches are accurately and precisely found.

Choosing The Right Geometric Matching Technique

The edge-based geometric matching technique works on any arbitrary shape and is guaranteed to find the object in the inspection image as long as a significant portion of the shape remains similar to the shape of the template object. There are no restrictions on the shape of the object in the template. As long as the curves detected around the object in the inspection image duplicate the curves that were extracted in the template image, the edge-based geometric matching technique will find the match.

The feature-based geometric matching technique works on the assumption that the shape of the pattern in the template can be reliably represented by a set of geometric features. This technique should be employed only when the pattern in the template and in the inspection images can be consistently and reliably represented by geometric shapes such as circles, rectangles and lines.

The memory and performance requirements of your application may influence which geometric matching technique you use. In general, an edge-based geometric template uses more memory than a feature-based geometric template. The size disparity between the template types increases with the permitted variance in scale. The more scale changes you want to match for, the larger the size of the edge-based template. The edge-based geometric matching technique is also slower than the feature-based geometric matching technique when matching at different scale ranges.

Follow these recommendations to choose the best geometric matching technique for your application:

- Always start with the edge-based geometric matching algorithm. The edge-based geometric matching algorithm provides the best recognition results.
- If the performance or memory requirements of the edge-based geometric matching algorithm and template do not meet the requirements of your

application, carefully adjust the match ranges for variance in scale or rotation. For example, if the match object in the inspection image is always the same size and rotates ± 10 degrees, then learn the template only for a scale range of 100% and a rotation range of -10 to 10 degrees. The performance of the edge-based method can also be improved by setting the factor by which the template and inspection are sampled at before the matching is done. Use the advanced learn options to specify a sampling factor.

- If you still cannot reach the performance or memory requirements of your application, and the object you need to match contains geometric features that can be reliably extracted, use the feature-based geometric matching algorithm.

Related concepts:

- [Pattern Matching](#)
- [Geometric Matching Technique](#)

Geometric Matching Using Calibrated Images

During matching, the geometric matching algorithm uses calibration information attached to the inspection image to return the position, angle, and bounding rectangle of a match in both pixel and real-world units. In addition, if the image is calibrated for perspective or nonlinear distortion errors, geometric matching uses the attached calibration information directly to find matches in the inspection image without using time-consuming image correction.

Simple Calibration or Previously Corrected Images

If an inspection image contains simple calibration information, or if the inspection image has been corrected prior to being used by geometric matching, the matching stage performs the same way that it does with uncalibrated images. However, each match result is returned in both pixel and real-world units. The pixel-unit results are identical to the results that would have been returned from matching the same, uncalibrated image. Geometric matching converts the pixel units to real-world units using the simple calibration information attached to the inspection image.

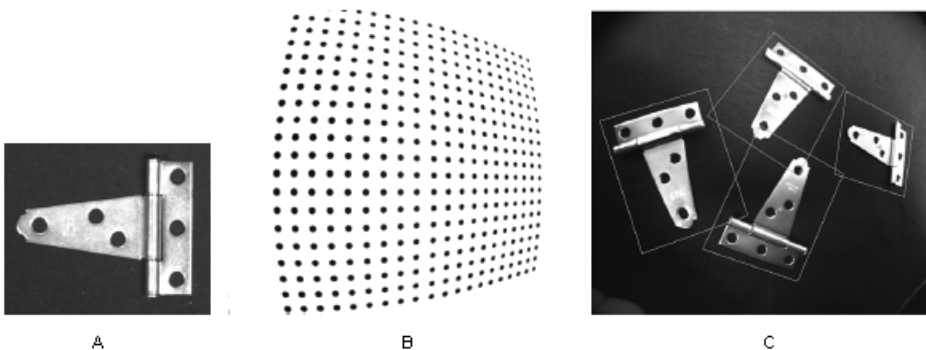
Perspective or Nonlinear Distortion Calibration

If an inspection image contains calibration information for perspective or nonlinear

distortions, the first step in the matching process is different than it would be with uncalibrated images. In the first step, curves extracted from the inspection image are corrected for distortion errors using calibration information. The remaining four steps in the matching process are performed on the corrected curves. Each match result is returned in pixel and real-world units.

Match results in pixel units are returned to be consistent with the inspection image. As a result, the bounding rectangle of a match in pixel units may not be rectangular, as shown in the following figure.

- Figure A shows the template image of a metallic part.
- Figure B shows an image of a calibration grid. The image exhibits nonlinear distortion.
- Figure C shows an image of metallic parts taken with the same camera setup used in Figure B. The gray lines depict the bounding rectangle of each match found by geometric matching.



In-Depth Discussion

This section provides additional information you may need for building successful geometric matching applications.

Geometric Matching Report

The geometric matching algorithm returns a report about the matches found in the inspection image. This report contains the location, angle, scale, occlusion percentage, and accuracy scores of the matches. The following sections explain the accuracy scores in greater detail.

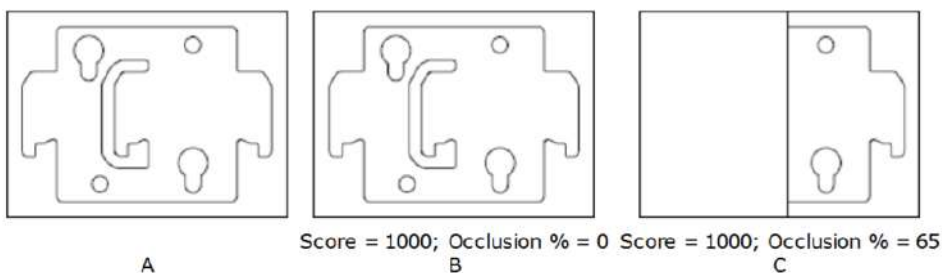
Score

The general score ranks the match results on a scale of 0 to 1000, where 0 indicates no match and 1000 indicates a perfect match. The general score takes the following factors into consideration:

- The number of geometric features in the template image that matched the target.
- The individual scores obtained from matching template features to their corresponding features in the inspection image.
- The score obtained by comparing the edge strengths of the curves in the template image to the edge strengths of the corresponding curves in the inspection image.

When geometric matching is used to find objects, the score is computed using only the curves and features in the template that were matched in the inspection image. Therefore, a partially occluded match could have a very high score if the features in the non-occluded regions of the part matched perfectly with the template features.

- Figure A shows the learned template curves of a part.
- Figure B shows the template match curves of a non-occluded part.
- Figure C shows the template match curves of an occluded part.



Note The general score is the score that the algorithm uses during matching to remove matches that fall below a user-defined **Minimum Match Score** value.

Template Target Curve Score

The template target curve score specifies how closely the curves in the template image match the curves in the match region of the inspection, or target, image. Score values can range from 0 to 1000, where a score of 1000 indicates that all template curves have a corresponding curve in the match region of the inspection image.

The template target curve score is computed by combining the match scores obtained by comparing each curve in the template to its corresponding curve in the target match region. Unlike the general score, the template target curve score is computed using all of the template curves. A low score implies one or both of the following:

- Some curves, or parts of curves, that are present in the template were not found in the inspection image, perhaps because of occlusion.
- The curves found in the inspection image were deformed and did not perfectly match the template curves.

You can use the template target curve score in inspection tasks to determine if the located part has flaws caused by anomalies such as process variations or printing errors. These flaws appear as deformed or missing curves in the inspection image. The following figure shows template target curve scores obtained for different scenarios.

Target Template Curve Score

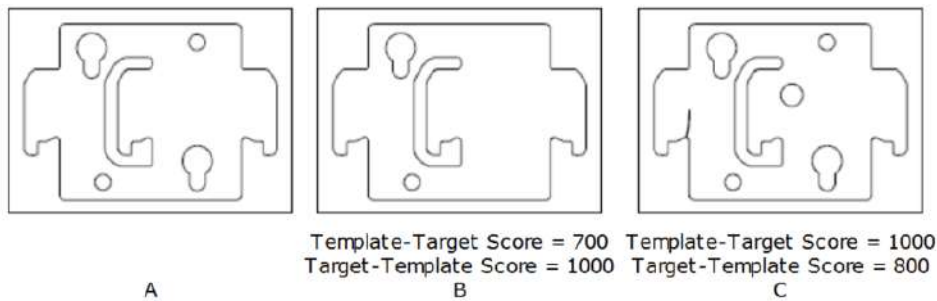
The target template curve score specifies how closely the curves in the match region of the inspection, or target, image match the curves in the template. Score values can range from 0 to 1000, where a score of 1000 indicates that all curves in the match region of the inspection image have a corresponding curve in the template image.

The target template curve score is computed by combining the match scores obtained by comparing each curve in the match region to the curves in the template image.

A low score implies one or both of the following:

- Some curves, or parts of curves, that are present in the match region of the inspection image were not found in the template image.
- The curves found in the inspection image were deformed and did not perfectly match the template curves.

You can use the target template curve score in inspection tasks to determine if there were additional curves in the inspection image because of flaws, such as scratches, or because of spurious objects in the match region that were not present in the template image. The following figure shows target template curve scores obtained for different scenarios.



Correlation Score

The correlation score is obtained by computing the correlation value between the pixel intensities of the template image to the pixel intensities of the target match. The correlation score is similar to the score returned by the pattern matching algorithm described in ***pattern matching***.

The correlation score ranges from 0 to 1000. A score of 1000 indicates a perfect match. The value of the correlation score is always positive. The algorithm returns the same correlation score for a match whose contrast is similar to that of the template and for a match whose contrast is a reversed version of the template.



Note The **Contrast Reversed** or inverse outputs of geometric matching indicate whether the contrast in the match region is the inverse of the contrast in the template.

You can specify regions in the template image that you do not want to use when computing the correlation score. Use the Vision Template Editor to specify regions in the template that you want to exclude from the computation of the correlation score.

Related concepts:

- [Pattern Matching](#)

Dimensional Measurements

This section contains information about coordinate systems, analytic tools, and clamps.

Introduction

You can use dimensional measurements or gauging tools in Vision to obtain quantifiable, critical distance measurements such as distances, angles, areas, line fits, circular fits, and quantities. These measurements can help you to determine if a product was manufactured correctly.

Components such as connectors, switches, and relays are small and manufactured in high quantity. Human inspection of these components is tedious, time consuming, and inconsistent. Vision can quickly and consistently measure certain features on these components and generate a report of the results. If the gauged distance or count does not fall within user-specified tolerance limits, the component or part fails to meet production specifications and should be rejected.

When to Use

Use gauging for applications in which inspection decisions are made on critical dimensional information obtained from image of the part. Gauging is often used in both inline and offline production. During inline processes, each component is inspected as it is manufactured. Inline gauging inspection is often used in mechanical assembly verification, electronic packaging inspection, container inspection, glass vial inspection, and electronic connector inspection.

You also can use gauging to measure the quality of products off-line. First, a sample of products is extracted from the production line. Then, using measured distances between features on the object, Vision determines if the sample falls within a tolerance range. Gauging techniques also allow you to measure the distance between particles and edges in binary images and easily quantify image measurements.

Concepts

The gauging process consists of the following four steps:

1. Locate the component or part in the image
2. Locate features in different areas of the part
3. Make measurements using these features
4. Compare the measurements to specifications to determine if the part passes

inspection

Locating the Part in the Image

A typical gauging application extracts measurements from ROIs rather than from an entire image. To use this technique, the necessary parts of the object must always appear inside the ROIs you define.

Usually, the object under inspection appears shifted or rotated within the images you want to process. When this occurs, the ROIs need to shift and rotate in the same way as the object. In order for the ROIs to move in relation to the object, you must locate the object in every image. Locating the object in the image involves determining the x, y position and the orientation of the object in the image using the reference coordinate system functions. You can build a coordinate reference using edge detection or pattern matching.

Locating Features

To gauge an object, you need to find landmarks or object features on which you can base your measurements. In most applications, you can make measurements based on points detected in the image or geometric fits to the detected points. Object features that are useful for measurements fall into two categories:

- Edge points along the boundary of an object located by the edge detection method
- Shapes or patterns within the object located by pattern matching

Making Measurements

You can make different types of measurements from the features found in the image. Typical measurements include the distance between points; the angle between two lines represented by three or four points; the best linear, circular, or elliptical fits; and the areas of geometric shapes, such as circles, ellipses, and polygons, that fit detected points. For more information about the types of measurements you can make, refer to your Vision user manual.

Qualifying Measurements

The last step of a gauging application involves determining the quality of the part based on the measurements obtained from the image. You can determine the quality of the part using either relative comparisons or absolute comparisons.

In many applications, the measurements obtained from the inspection image can be compared to the same measurements obtained from a standard specification or a reference image. Because all the measurements are made on images of the part, you can compare them directly.

In other applications, the dimensional measurements obtained from the image must be compared with values that are specified in real units. In this case, convert the measurements from the image into real-world units using the calibration tools described in ***system setup and calibration***.

Related concepts:

- [Spatial Calibration](#)

Coordinate System

In a typical machine vision application, measurements are extracted from an ROI rather than from the entire image. The object under inspection must always appear in the defined ROI in order to extract measurements from that ROI.

When the location and orientation of the object under inspection is always the same in the inspection images, you can make measurements directly without locating the object in every inspection image.

In most cases, the object under inspection is not positioned in the camera field of view consistently enough to use fixed search areas. If the object is shifted or rotated within an image, the search areas should shift and rotate with the object. The search areas are defined relative to a ***coordinate system***. A coordinate system is defined by a reference point (origin) and a reference angle in the image or by the lines that make up its two axes.

When to Use

Use coordinate systems in a gauging application when the object does not appear in the same position in every inspection image. You also can use a coordinate system to define search areas on the object relative to the location of the object in the image.

Concepts

All measurements are defined with respect to a coordinate system. A coordinate system is based on a characteristic feature of the object under inspection, which is used as a reference for the measurements. When you inspect an object, first locate the reference feature in the inspection image. Choose a feature on the object that the software can reliably detect in every image. Do not choose a feature that may be affected by manufacturing errors that would make the feature impossible to locate in images of defective parts.

You can restrict the region of the image in which the software searches for the feature by specifying an ROI that encloses the feature. Defining an ROI in which you expect to find the feature can prevent mismatches if the feature appears in multiple regions of the image. A small ROI may also improve the locating speed.

Complete the following general steps to define a coordinate system and make measurements based on the new coordinate system.

1. Define a reference coordinate system.
 - a. Define a search area that encompasses the reference feature or features on which you base your coordinate system. Make sure that the search area encompasses the features in all your inspection images.
 - b. Locate an easy-to-find reference feature of the object under inspection. That feature serves as the base for a reference coordinate system in a reference image. You can use two primary techniques to locate the feature: edge detection or pattern matching.

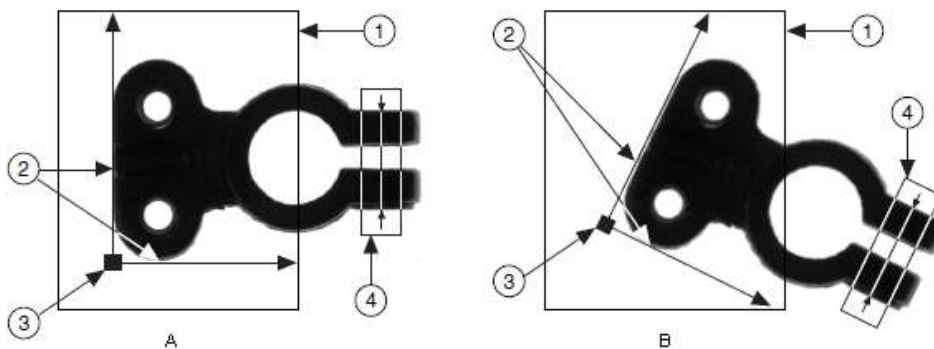
The software builds a coordinate system to keep track of the location and orientation of the object in the image.

2. Set up measurement areas within the reference image in which you want to make measurements.

3. Acquire an image of the object to inspect or measure.
4. Update the coordinate system. During this step, Vision locates the features in the search area and builds a new coordinate system based on the new location of the features.
5. Make measurements within the updated measurement area.

Vision computes the difference between the reference coordinate system and the new coordinate system. Based on this difference, the software moves the new measurement areas with respect to the new coordinate system.

Figure A illustrates a reference image with a defined reference coordinate system. Figure B illustrates an inspection image with an updated coordinate system.



1. Search Area for the Coordinate System
2. Object Edges
3. Origin of the Coordinate System
4. Measurement Area

In-Depth Discussion

You can use four different strategies to build a coordinate system. Two strategies are based on detecting the reference edges of the object under inspection. The other two strategies involve locating a specific pattern using a pattern matching algorithm.

Edge-Based Coordinate System Functions

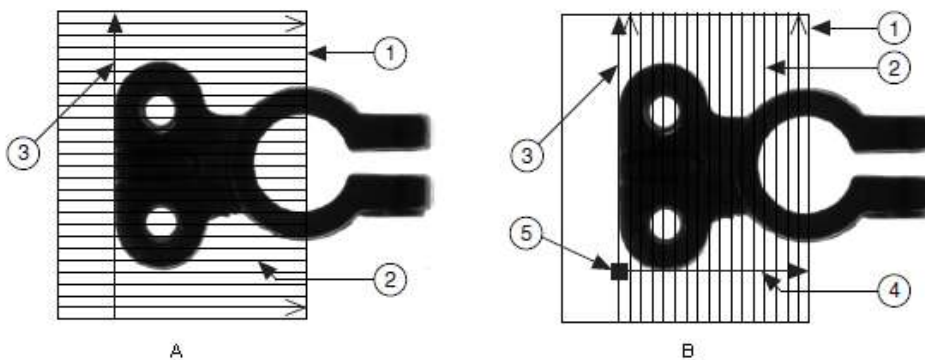
These functions determine the axis of the coordinate system by locating edges of the part under inspection. Use an edge-based method if you can identify two straight, distinct, non-parallel edges on the object you want to locate. Because the software

uses these edges as references for creating the coordinate system, choose edges that are unambiguous and always present in the object under inspection.

Single Search Area

This method involves locating the two axes of the coordinate system—the main axis and secondary axis—in a single search area based on an edge detection algorithm. First, the function determines the main, vertical, axis of the coordinate system, as illustrated in figure A. NI Vision uses the **straight edge detection algorithm** to locate the main axis in the image. The straight edge detected by the algorithm defines the main axis. The function then searches for a secondary, horizontal, axis using the straight edge detection algorithm on a search area perpendicular to the main axis. The detected straight edge defines the secondary axis of the coordinate system. Figure B shows the location of the secondary axis in a sample image. The secondary axis must not be parallel to the main axis. The intersection between the main axis and secondary axis defines the origin of the reference coordinate system.

Figure A illustrates a reference image with a defined reference coordinate system. Figure B illustrates an inspection image with an updated coordinate system.



1. Search Area for the Coordinate System
2. Search Lines
3. Main Axis
4. Secondary Axis
5. Origin of the Reference Coordinate System

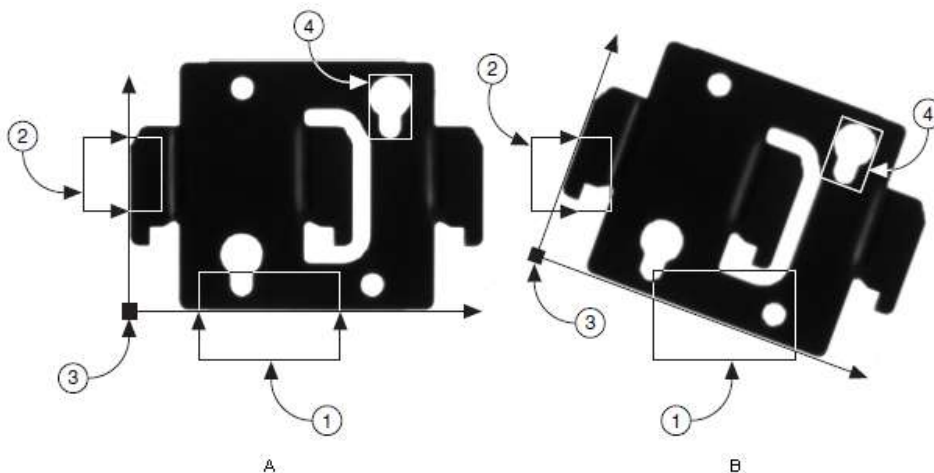
Two Search Areas

This method uses the same operating mode as the single search area method.

However, the two edges used to define the coordinate system axes are located in two distinct search areas.

The function first determines the position of the main axis of the coordinate system. It locates the main axis using the **straight edge detection algorithm** in the primary search area. The detected straight edge defines the primary axis. The process is repeated perpendicularly in the secondary search area to locate the secondary axis. The intersection between the primary axis and secondary axis is the origin of the coordinate system.

Figure A illustrates a reference image with a defined reference coordinate system. Figure B illustrates an inspection image with an updated coordinate system.



1. Primary Search Area
2. Secondary Search Area
3. Origin of the Coordinate System
4. Measurement Area

Pattern Matching-Based Coordinate System Functions

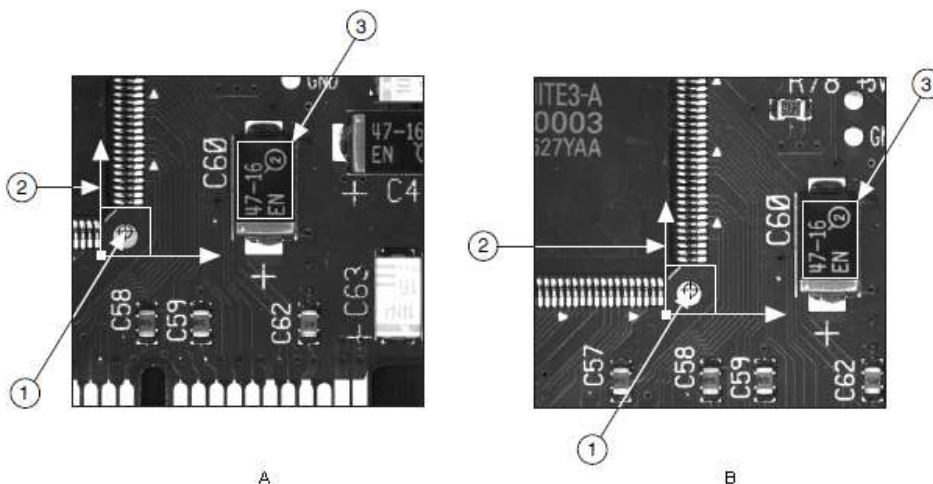
Using **pattern matching** techniques to locate a reference feature is a good alternative to edge detection when you cannot find straight, distinct edges in the image. The reference feature, or template, is the basis for the coordinate system.

The software searches for a template image in a rectangular search area of the reference image. The location and orientation of the located template is used to create

the reference position of a coordinate system or to update the current location and orientation of an existing coordinate system.

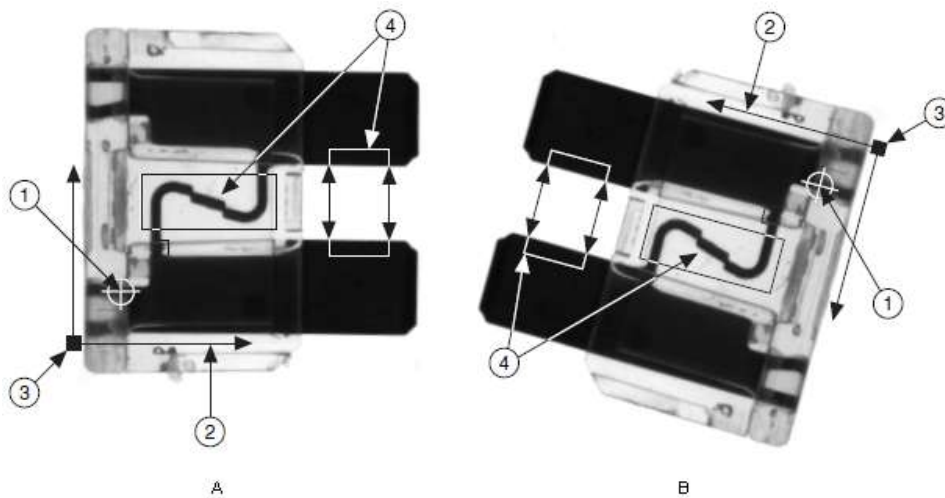
The same constraints on feature stability and robustness that apply to the edge-detection techniques also apply to pattern matching. Pattern matching uses one of two strategies: shift-invariant pattern matching and rotation-invariant pattern matching. Shift-invariant pattern matching locates a template in an ROI or in the entire image with a maximum tolerance in rotation of $\pm 5^\circ$. The rotation-invariant strategy locates a template in the image even when the template varies in orientation between 0° and 360° .

The following illustration shows how to locate a coordinate system using a shift-invariant pattern matching strategy. Figure A shows a reference image with a defined reference coordinate system. Figure B shows an inspection image with an updated coordinate system.



1. Located Feature
2. Coordinate System
3. Measurement Area

The following illustration shows how to locate a coordinate system using a rotation-invariant pattern matching strategy. Figure A shows a reference image with a defined reference coordinate system. Figure B shows an inspection image with an updated coordinate system.



1. Located Feature
2. Coordinate System
3. Origin of the Coordinate System
4. Measurement Area

Related concepts:

- [Concepts](#)
- [Concepts](#)
- [Pattern Matching](#)

Finding Features or Measurement Points

Before making measurements, you must locate features that you can use to make the measurements. There are many ways to find these features on an image. The most common features used to make measurements are points along the boundary of the part you want to gauge.

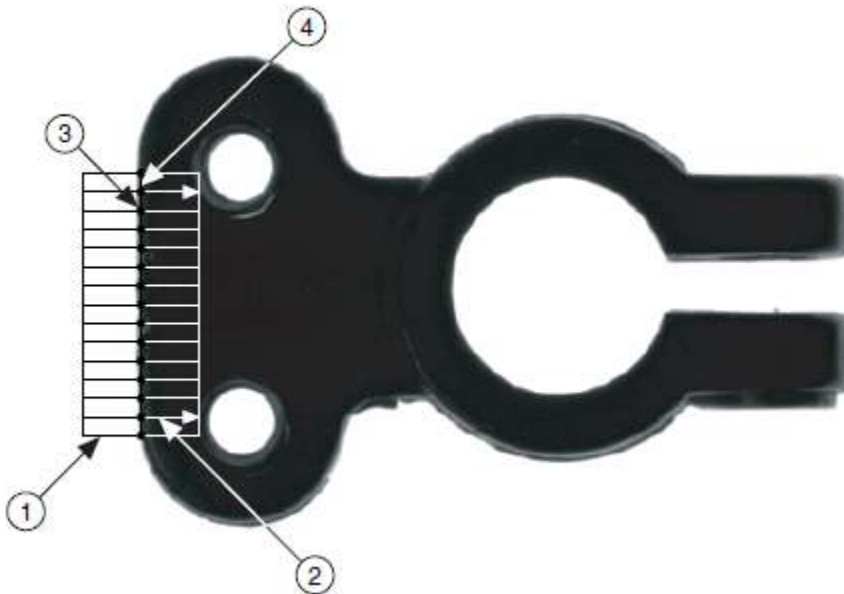
Edge-Based Features

Use **edge detection** techniques to find edge points along a single search contour or along multiple search contours defined inside a 2D search area.

Line and Circular Features

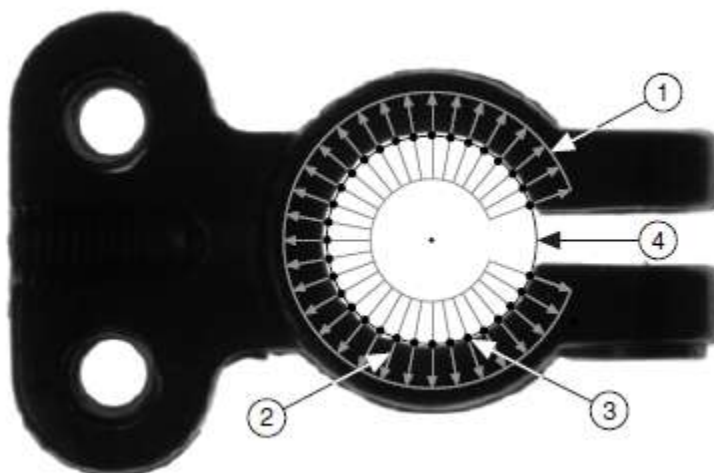
Use the line detection functions in Vision to find vertically or horizontally oriented

lines. These functions use the **rake** and **concentric rake** functions to find a set of points along the edge of an object and then **fit a line** through the edge. The following figure illustrates how a rake finds a straight edge.



1. Search Region
2. Search Lines
3. Detected Edge Points
4. Line Fit to Edge Points

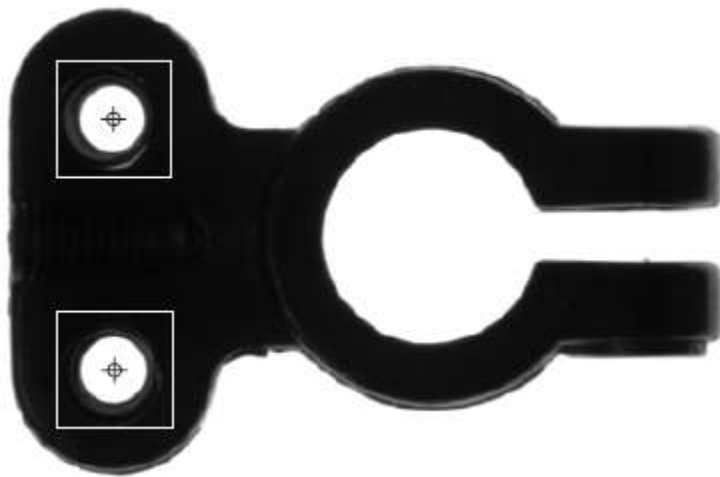
Use the circle detection function to locate circular edges. This function uses a spoke to find points on a circular edge, and then fits a circle on the detected points. The following figure illustrates how a spoke finds circular edges.



1. Annular Search Region
2. Search Lines
3. Detected Edge Points
4. Circle Line Fit to Edge Points

Shape-Based Features

Use pattern matching or color pattern matching to find features that are better described by the shape and grayscale or color content than the boundaries of the part.



Related concepts:

- [Edge Detection](#)
- [Concepts](#)
- [Making Measurements on the Image](#)

Making Measurements on the Image

After you have located points in the image, you can make distance or geometrical measurements based on those points.

Distance Measurements

Make distance measurements using one of the following methods:

- Measure the distance between points found by one of the feature detection methods

- Measure the distance between two edges of an object using the clamp functions available in Vision

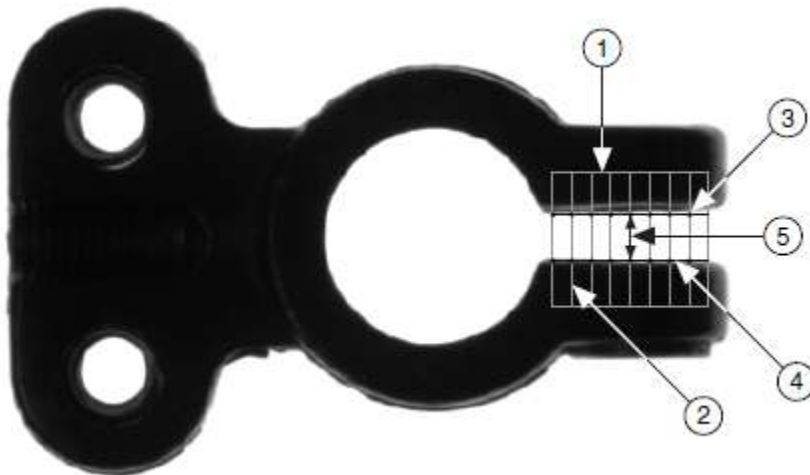
Clamp functions measure the separation between two edges in a region of interest. Use the clamp functions to find the smallest or largest separation between two edges at the same orientation.

Vision includes two clamp functions: one which uses rake-based edge detection, and one which uses contour detection.

Clamp (Rake-Based)

The rake-based clamp function supports both min and max distance calculations. The clamp function detects points along the two edges using the rake function, then computes the distance between the detected points and returns the largest or smallest distance.

The following figure illustrates how a rake-based clamp function finds the minimum distance between the edges of an object.



1. Rectangular Search Region
2. Search Lines for Edge Detection
3. Detected Edge Points
4. Line Fit to Edge Points
5. Measured Distance

Clamp (Contour Extraction Based)

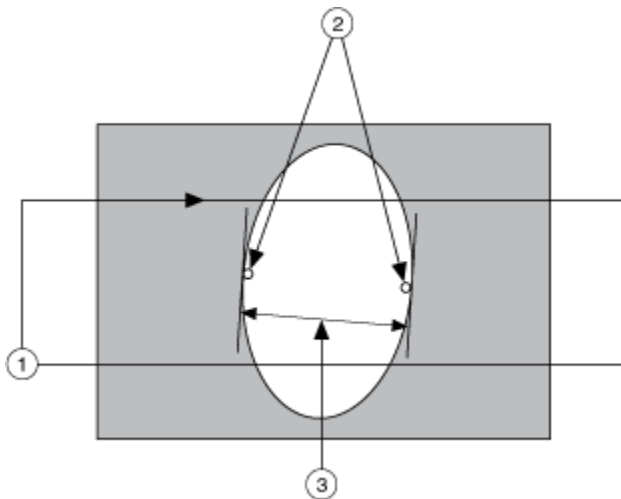
The contour extraction based clamp function supports only max distance calculations. The clamp searches for contours within a user-specified angle range relative to the search axis of a rectangular region of interest. You can also specify the desired edge polarity for the clamp boundaries. The edge polarity for the entire boundary is defined by the initial edge polarity of the boundary along the search direction. For more information about edge polarity, refer to ***Edge Detection Concepts***.



Note The clamp function treats each extracted contour as a single boundary. For example, if the object under inspection is a white disc, the function will identify a single rising polarity boundary. Create an ROI that does not include the entire object to force the function to identify multiple boundaries.

After extracting a contour, the clamp selects opposing points with parallel tangents and computes the distance between the points.

The following figure illustrates how a contour extraction based clamp function finds the maximum distance between edges of an object. Refer to ***Contour Analysis Concepts*** for more information about contour extraction.



1. Rotated rectangle search region
2. Found clamp points
3. Measured distance, within angle tolerance

Analytic Geometry

You can make the following geometrical measurements from the feature points detected in the image.

- The area of a polygon specified by its vertex points
- The line that fits to a set of points and the equation of that line
- The circle that fits to a set of points and its area, perimeter, and radius
- The ellipse that fits to a set of points and its area, perimeter, and the lengths of its major and minor axis
- The intersection point of two lines specified by their start and end points
- The line bisecting the angle formed by two lines
- The line midway between a point and a line that is parallel to the line
- The perpendicular line from a point to a line, which computes the perpendicular distance between the point and the line

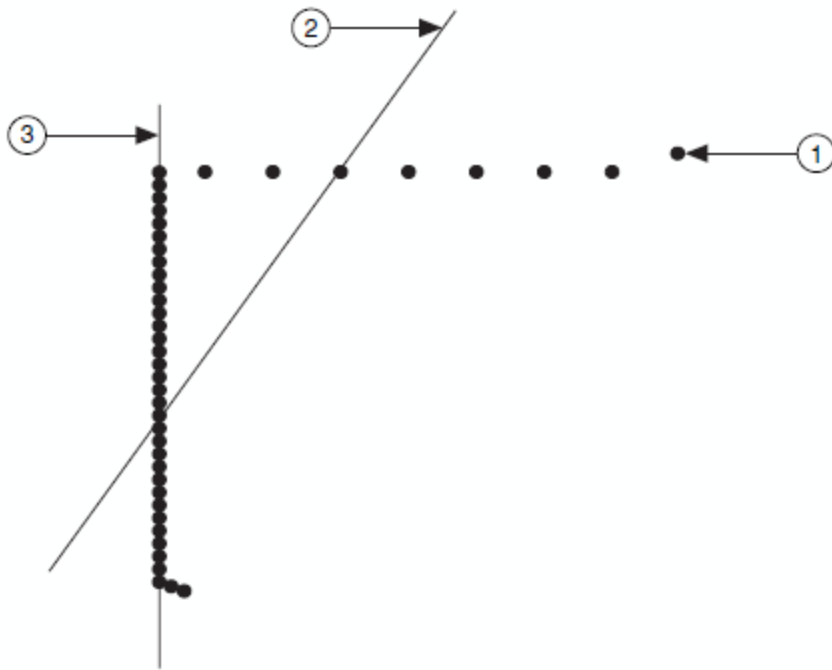
Line Fitting

The line fitting function in Vision uses a robust algorithm to find a line that best fits a set of points. The line fitting function works specifically with the feature points obtained during gauging applications.

In a typical gauging application, a rake or a concentric rake function finds a set of points that lie along a straight edge of the object. In an ideal case, all the detected points would make a straight line. However, points usually do not appear in a straight line for one of the following reasons:

- The edge of the object does not occupy the entire search region used by the rake
- The edge of the object is not a continuous straight line
- Noise in the image causes points along the edge to shift from their true positions

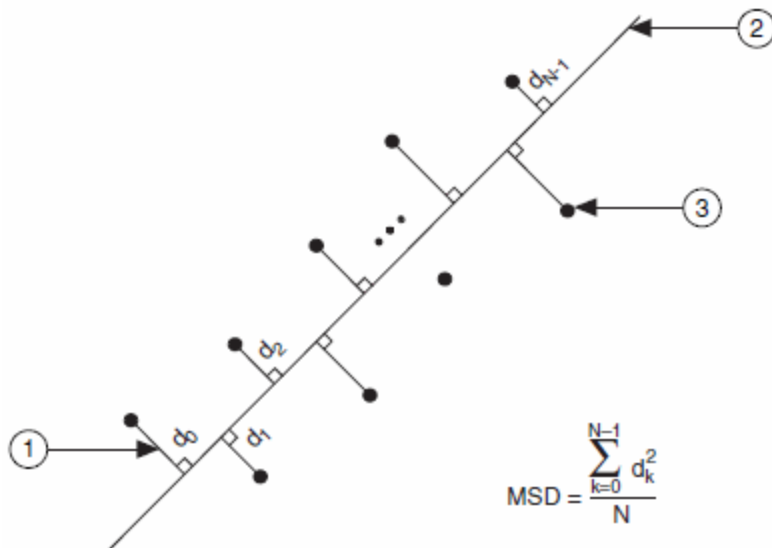
The following figure illustrates an example of a set of points located by the rake function. As shown in the figure, a typical line fitting algorithm that uses all of the points to fit a line returns inaccurate results. The line fitting function in NI Vision compensates for outlying points in the dataset and returns a more accurate result.



1. Edge Points
2. Standard Line Fit
3. Vision Fit Line

Vision uses the following process to fit a line. Vision assumes that a point is part of a line if the point lies within a user-defined distance—or pixel radius—from the fitted line. Then the line fitting algorithm fits a line to a subset of points that fall along an almost straight line. Vision determines the quality of the line fit by measuring its mean square distance (MSD), which is the average of the squared distances between each point and the estimated line.

The following figure illustrates how the MSD is calculated. Next, the line fitting function removes the subset of points from the original set. Vision repeats these steps until all points have been fit. Then, the line fitting algorithm finds the line with the lowest MSD, which corresponds to the line with the best quality. The function then improves the quality of the line by successively removing the furthest points from the line until a user-defined minimum score is obtained or a user-specified maximum number of iterations is exceeded.

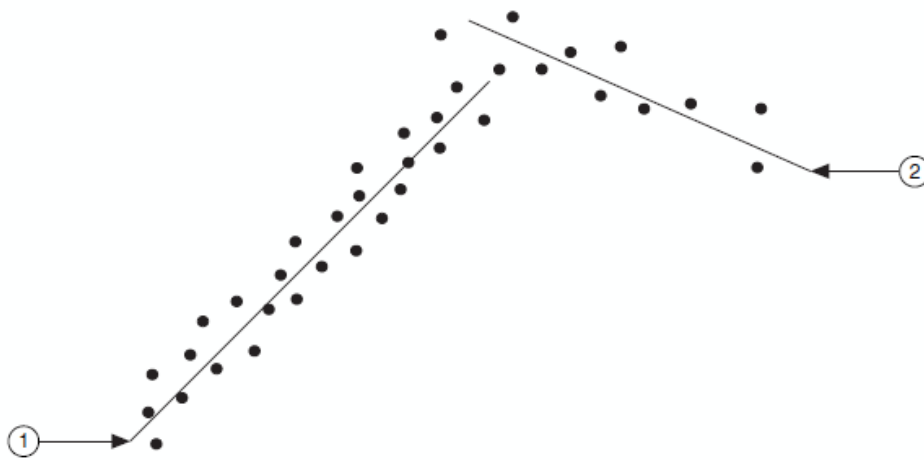


1. Perpendicular Distance from an Edge Point to the Line
2. Line Fit
3. Points Used to Fit the Line

The result of the line fitting function is a line that is fit to the strongest subset of the points after ignoring the outlying points, as shown in the following figure.

The pixel radius, minimum score, and maximum iteration parameters control the behavior of the line fit function.

The pixel radius defines the maximum distance allowed, in pixels, between a valid point and the estimated line. The algorithm estimates a line where at least half the points in the set are within the pixel radius. If a set of points does not have such a line, the function attempts to return the line that has the most number of valid points.



1. Strongest Line Returned by the Line Fit Function
2. Alternate Line Discarded by the Line Fit Function

Increasing the pixel radius increases the distance allowed between a point and the estimated line. Typically, you can use the imaging system resolution and the amount of noise in your system to gauge this parameter. If the resolution of the imaging system is very high, use a small pixel radius to minimize the use of outlying points in the line fit. Use a higher pixel radius if your image is noisy.

The minimum score allows you to improve the quality of the estimated line. The line fitting function removes the point furthest from the fit line, and then refits a line to the remaining points and computes the MSD of the line. Next, the function computes a line fit score (LFS) for the new fit using the following equation:

$$LFS = \left(\frac{1 - MSD}{PR^2} \right) \times 1000$$

where PR is the pixel radius.

Vision repeats the entire process until the score is greater than or equal to the minimum score or until the number of iterations exceeds the user-defined maximum number of iterations.

Use a high minimum score to obtain the most accurate line fit. For example, combining a large pixel radius and a high minimum score produces an accurate fit within a very noisy data set. A small pixel radius and a small minimum score produces a robust fit in a standard data set.

The maximum number of iterations defines a limit in the search for a line that satisfies the minimum score. If you reach the maximum number of iterations before the algorithm finds a line matching the desired minimum score, the algorithm stops and returns the current line. If you do not need to improve the quality of the line in order to obtain the desired results, set the maximum iterations value to 0 in the line fit function.

Related concepts:

- [Concepts](#)
- [Contour Analysis Concepts](#)

Contour Analysis

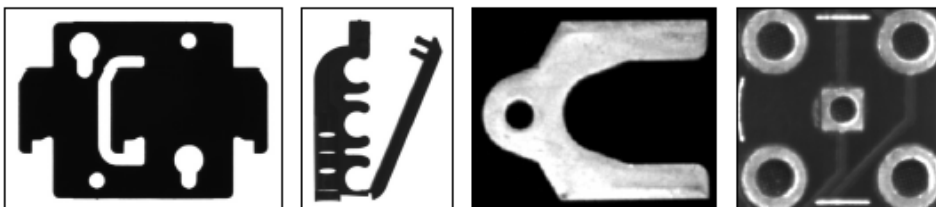
This section contains information about contour analysis.

Introduction

A contour represents a series of edge points that define the outline of an object in the image. Contour analysis locates and extracts contours in grayscale images. After extracting the contour the contour analysis algorithm can calculate the curvature along the contour, fit the contour with an equation of known type, or compare multiple contours.

When to Use

Contour analysis locates and extracts contours in grayscale images and allows you to compare extracted contours with a template contour or a fitted equation. The following figure shows examples of objects with good contour information.



You can use contour analysis in the following application areas:

- **Gauging**—Measures lengths, diameters, angles, and other critical dimensions. If the measurements fall outside set tolerance levels, the object is rejected. Use geometric matching to locate the object, or areas of the object, you want to gauge. Use information about the size of the object to preclude contour analysis from analyzing objects that are too big or small.
- **Inspection**—Detects flaws or missing elements. Compare an extracted contour to a fitted contour or template contour to detect flaws in edges or determine if a portion of the object under inspection is missing.

Contour Analysis Concepts

Contour extraction involves the following steps:

1. Curves are extracted from the ROI
2. Optionally, multiple curves are connected according to settings stored in connection parameters
3. A single connected curve is selected to represent the contour

Curve Extraction

A curve is a set of edge points that are connected to form a continuous contour. Curves typically represent the boundary of the part in the image. In curve extraction, curves are the underlying information used to represent a template and to match the template in an inspection image. This section describes how curves are extracted from an image.

The curve extraction process consists of finding curve seed points and tracing the curve.

Finding Curve Seed Points

A seed point is a point on a curve from which tracing begins. To qualify as a seed point, a pixel cannot be part of an already existing curve and must have an edge contrast greater than the user-defined edge threshold. The edge contrast at a pixel is computed as a function of the intensity value at that pixel and the intensities of its neighboring pixels. If $P_{(i,j)}$ represents the intensity of the pixel P with the coordinates (i, j) , the edge contrast at (i, j) is defined as

$$\sqrt{\left(P_{(i-1, j)} - P_{(i+1, j)}\right)^2 + \left(P_{(i, j-1)} - P_{(i, j+1)}\right)^2}$$

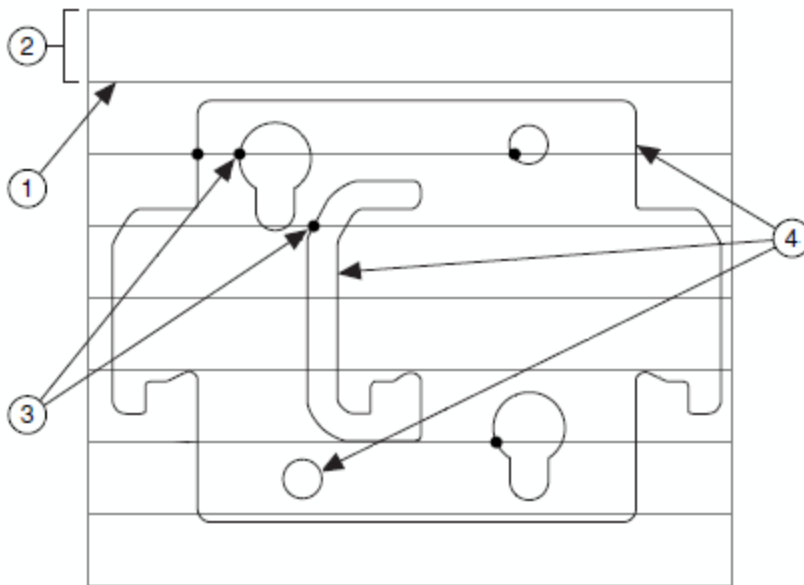
For an 8-bit image, the edge contrast may vary from 0 to 360.

To increase the speed of the curve extraction process, the algorithm visits only a limited number of pixels in the image to determine if the pixel is a valid seed point. The number of pixels to visit is based on the values that the user provides for the **Search Step** and ROI parameters. The larger the **Search Step** and the smaller the ROI, the faster the algorithm searches for seed points. However, to make sure that the algorithm finds a seed point on all of the curves, **Search Step** must be smaller than the smallest curve along the search direction.

The algorithm scans from the selected side of the ROI. Starting at the first pixel, the edge contrast of the pixel is computed. If the edge contrast is greater than the given threshold, the curve is traced from this point. If the contrast is lower than the threshold, or if this pixel is already a member of an existing curve previously computed, the algorithm analyzes the next pixel in the row to determine if it qualifies as a seed point. This process is repeated until the opposite side of the ROI is reached. The algorithm then skips Search Step pixels along the side of the ROI and repeats the process.

Tracing the Curve

When it finds a seed point, the curve extraction algorithm traces the rest of the curve. Tracing is the process by which a pixel that neighbors the last pixel on the curve is added to the curve if it has the strongest edge contrast in the neighborhood and the edge contrast is greater than acceptable edge threshold for a curve point. This process is repeated until no more pixels can be added to the curve in the current direction. The algorithm then returns to the seed point and tries to trace the curve in the opposite direction. The following figure illustrates this process.



1. Scan Lines
2. Search Step
3. Curve Seeds
4. Curves



Note To simplify the figure, **Search Step** is not smaller than the smallest curve.

Curve connection parameters specify how the contour extraction algorithm connects individual curves to produce contours. There are two methods for building curves into a contour. For both methods, all possible connections are limited by the input ranges for the connection metrics below. If a range is not entered, the connection metric is not evaluated when considering connections. For example, if only a distance and angle metric are provided, gradient angle and connectivity are not evaluated and will not affect connection choices. For both methods, once the curve that is being built is connected to itself, it is considered a closed curve and no more connections are made.

When selecting the closest contour, the extraction algorithm starts with the curve that is closest to the ROI side that curves were scanned from. The extraction algorithm builds onto each end of the closest curve by selecting the next curve that continues along the object boundary. This is defined as the candidate curve, whose end requires the smallest magnitude of rotation from the closest curve end, through the direction of

the ROI side.

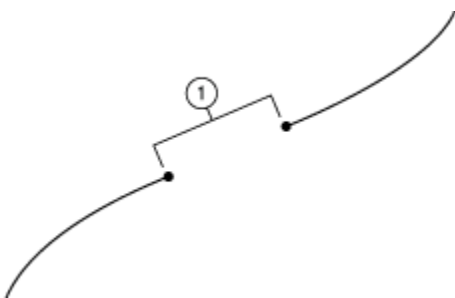
When selecting for the longest or strongest contour, the extraction algorithm connects curves with the minimum cost for all of the considered metrics. The cost is defined as the sum of each metric normalized to the range of that metric. For example, if the angle range is 0 to 30 degrees, and a possible connection has as 15 degree change between curves, the angle metric will contribute 0.5 to the cost of that connection. The extraction algorithm calculates the cost of all connections that fall within the metric ranges, then makes the connections in order from the least to most costly.

When selecting the closest contour, the closest contour is always built as described above. When selecting for the longest or strongest contour, and no connection parameters are entered, no connections that increase performance will be made. In any case where connections are made, the default metrics are distance with a range of 0 to 10 pixels, and angle distance with a range of 0 to 180 degrees. Input metrics will override these defaults.

Distance

Distance is the euclidean distance, in pixels, between the endpoints. Modify the **Distance** range parameters to only connect curves with end points separated by a distance within the specified range.

The following figure illustrates distance between end points:

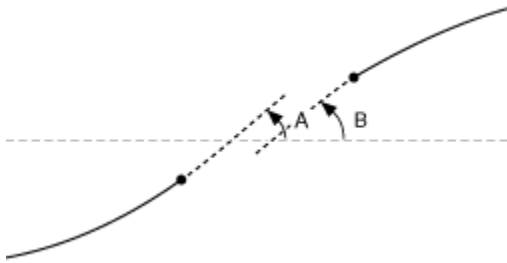


1. Distance

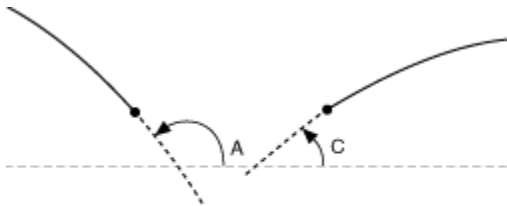
Angle Distance

Angle distance is the amount, in degrees, that one curve endpoint must rotate for two curves to be parallel at the endpoints. Modify the **Angle Distance** range parameters to

only connect curves when the difference between the angle of the curves, measured at the end points, is within the specified range. The following figures illustrate how the angle of a curve is calculated:



A

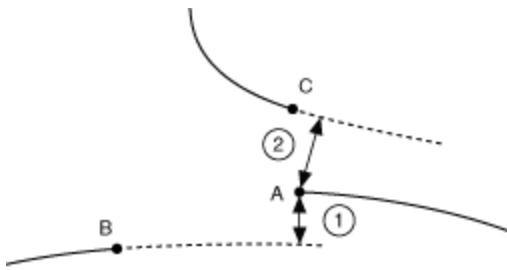


B

In Figure A, the difference between angle A and angle B is close to 0. In Figure B, the difference between angle A and angle C is close to 90 degrees.

Connectivity Distance

Connectivity distance is determined by projecting the endpoint of one curve as a line and finding the minimum distance from the projected line to the other endpoint. If the distance to the projection from either endpoint is within the range, the endpoints are connected. Modify the **Connectivity Distance** range parameters to only connect curves when a line extended from the end point of one curve passes the end point of another curve within the specified distance. The following figure illustrates how connectivity is calculated:

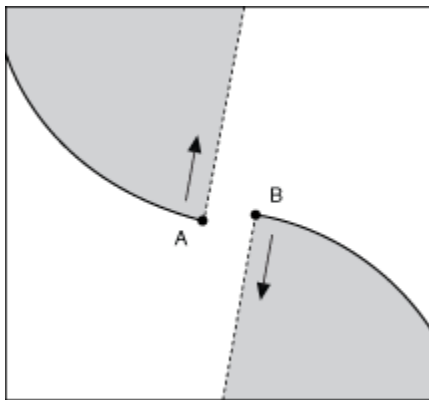


1. Distance between end point A and a line extended from end point B
2. Distance between end point A and endpoint C

End point A is closer to end point C than to end point B. Specify a connectivity range to connect end point A to end point B instead of end point C.

Gradient Difference

Gradient difference is calculated by determining the gradient angle at each endpoint, then taking the absolute difference of the two angles in degrees. Modify the **Gradient Difference** range parameters to only connect curves when the difference between the gradient angle of each curve is within the specified range. The following figure illustrates two curves with opposite gradient angles:



Contour Selection

After the curves have been extracted from the image and optionally connected, a single contour is selected from this set of curves based on contour selection parameters. You can select the first contour detected along the search direction, the longest contour, or the contour with the highest edge strength averaged from each point on the curve.

In-Depth Discussion

Curvature

The curvature of a contour is calculated from the edge points of the contour and by the input kernel size. For each point along the contour the algorithm selects two additional points at half the kernel width before and half the kernel width after the current point. The algorithm fits a cubic b-spline to the three points. Curvature represents the inverse radius of the circle inscribed by the cubic b-spline at the current point. Curvature can be negative. A negative curvature indicates a curve to the left along the search direction. If the current point is too close to either end of the array to choose the additional points, the curvature is calculated as 0.

Distances

When comparing two contours, the algorithm generates pairs of corresponding points between the contours. At each point along the template contour the algorithm examines the target contour for a matching point. Ideally, the matching point is normal to the curve of the template contour, as when the target contour is a fitted line, circle, or ellipse. However, when the target contour is defined by a discrete set of points, a point along the target contour may not be normal to the point on the template curve.

To compare a template contour and a target contour defined by a discrete set of points, the algorithm applies a Gaussian averaging kernel to smooth the target contour points. Then, for each point along the template contour, the algorithm examines a line segment formed by consecutive points on the smoothed target contour points. If a line segment includes a point that is normal to the template point, the algorithm pairs the points. Otherwise, the algorithm approximates a normal point by pairing the template point with the closest point on the target contour. Finally, for unpaired sections of the target contour, the algorithm pairs each unmatched target contour point with a point on the template contour between the matched points on either side of the unmatched point.

For calibrated contour images, the algorithm first performs the distance computation in the calibrated space of the target image, then transforms the points into pixel coordinates.

Color Inspection

This section contains information about color spaces, the color spectrum, color matching, color location, and color pattern matching.

Color Spaces

Color spaces allow you to represent a color. A color space is a subspace within a 3D coordinate system where each color is represented by a point. You can use color spaces to facilitate the description of colors between persons, machines, or software programs.

Various industries and applications use a number of different color spaces. Humans perceive color according to parameters such as brightness, hue, and intensity, while computers perceive color as a combination of red, green, and blue. The printing industry uses cyan, magenta, and yellow to specify color. The following is a list of common color spaces:

- **RGB**—Based on red, green, and blue. Used by computers to display images.
- **HSL**—Based on hue, saturation, and luminance. Used in image processing applications.
- **CIE**—Based on brightness, hue, and colorfulness. Defined by the Commission Internationale de l'Eclairage (International Commission on Illumination) as the different sensations of color that the human brain perceives.
- **CMY**—Based on cyan, magenta, and yellow. Used by the printing industry.
- **YIQ**—Separates the luminance information (Y) from the color information (I and Q). Used for TV broadcasting.

When to Use

You must define a color space every time you process color images. With NI Vision, you specify the color space associated with an image when you create the image. NI Vision supports the RGB and HSL color spaces.

If you expect the lighting conditions to vary considerably during your color machine vision application, use the HSL color space. The HSL color space provides more accurate color information than the RGB space when running color processing

functions, such as color matching, color location, and color pattern matching. NI Vision's advanced algorithms for color processing—which perform under various lighting and noise conditions—process images in the HSL color space.

If you do not expect the lighting conditions to vary considerably during your application, and you can easily define the colors you are looking for using red, green, and blue, use the RGB space. Also, use the RGB space if you want only to display color images, but not process them, in your application. The RGB space reproduces an image as you would expect to see it. NI Vision always displays color images in the RGB space. If you create an image in the HSL space, NI Vision automatically converts the image to the RGB space before displaying it.

Concepts

Because color is the brain's reaction to a specific visual stimulus, color is best described by the different sensations of color that the human brain perceives. The color-sensitive cells in the retina sample color using three bands that correspond to red, green, and blue light. The signals from these cells travel to the brain where they combine to produce different sensations of colors. The Commission Internationale de l'Eclairage has defined the following sensations:

- **Brightness**—The sensation of an area exhibiting more or less light.
- **Hue**—The sensation of an area appearing similar to a combination of red, green, and blue.
- **Colorfulness**—The sensation of an area appearing to exhibit more or less of its hue.
- **Lightness**—The sensation of an area's brightness relative to a reference white in the scene.
- **Chroma**—The colorfulness of an area with respect to a reference white in the scene.
- **Saturation**—The colorfulness of an area relative to its brightness.

The trichromatic theory describes how three separate lights—red, green, and blue—can be combined to match any visible color. This theory is based on the three color sensors that the eye uses. Printing and photography use the trichromatic theory as the basis for combining three different colored dyes to reproduce colors in a scene. Similarly, computer color spaces use three parameters to define a color.

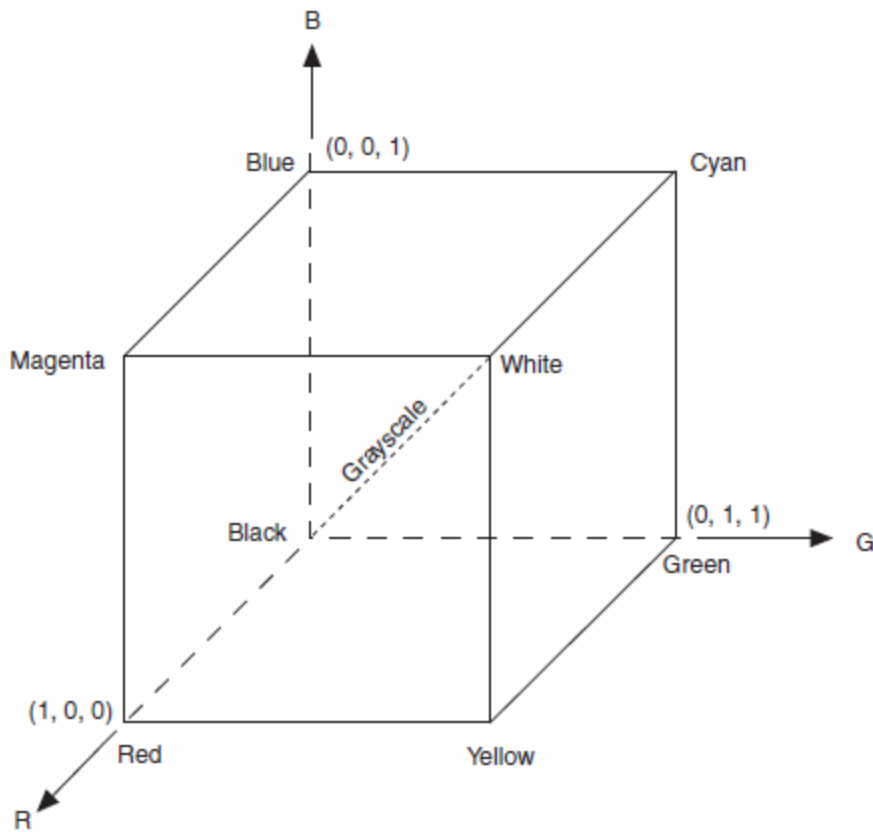
Most color spaces are geared toward displaying images with hardware, such as color

monitors and printers, or toward applications that manipulate color information, such as computer graphics and image processing. Color CRT monitors, the majority of color-video cameras, and most computer graphics systems use the RGB color space. The HSL space, combined with RGB and YIQ, is frequently used in applications that manipulate color, such as image processing. The color picture publishing industry uses the CMY color space, also known as CMYK. The YIQ space is the standard for color TV broadcast.

RGB Color Space

The RGB color space is the most commonly used color space. The human eye receives color information in separate red, green, and blue components through cones—the color receptors present in the human eye. These three colors are known as additive primary colors. In an additive color system, the human brain processes the three primary light sources and combines them to compose a single color image. The three primary color components can combine to reproduce most possible colors.

You can visualize the RGB space as a 3D cube with red, green, and blue at the corners of each axis, as shown in the following figure. Black is at the cube origin, while white is at the opposite corner of the cube. Each side of the cube has a value between 0 and 1. Along each axis of the RGB cube, the colors range from no contribution of that component to a fully saturated color. Any point, or color, within the cube is specified by three numbers: an R, G, B triple. The diagonal line of the cube from black (0, 0, 0) to white (1, 1, 1) represents all the grayscale values or where all of the red, green, and blue components are equal. Different computer hardware and software combinations use different color ranges. Common combinations are 0–255 and 0–65,535 for each component. To map color values within these ranges to values in the RGB cube, divide the color values by the maximum value that the range can take.



The RGB color space lies within the perceptual space of humans. In other words, the RGB cube represents fewer colors than we can see.

The RGB space simplifies the design of computer monitors, but it is not ideal for all applications. In the RGB color space, the red, green, and blue color components are all necessary to describe a color. Therefore, RGB is not as intuitive as other color spaces. The HSL color space describes color using only the hue component, which makes HSL the best choice for many image processing applications, such as color matching.

HSL Color Space

The HSL color space was developed to put color in terms that are easy for humans to quantify. Hue, saturation, and brightness are characteristics that distinguish one color from another in the HSL space. Hue corresponds to the dominant wavelength of the color. The hue component is a color, such as orange, green, or violet. You can visualize the range of hues as a rainbow. Saturation refers to the amount of white added to the hue and represents the relative purity of a color. A color without any white is fully saturated. The degree of saturation is inversely proportional to the amount of white

light added. Colors such as pink, composed of red and white, and lavender composed of purple and white, are less saturated than red and purple. Brightness embodies the chromatic notion of luminance, or the amplitude or power of light. Chromaticity is the combination of hue and saturation. The relationship between chromaticity and brightness characterizes a color. Systems that manipulate hue use the HSL color space.

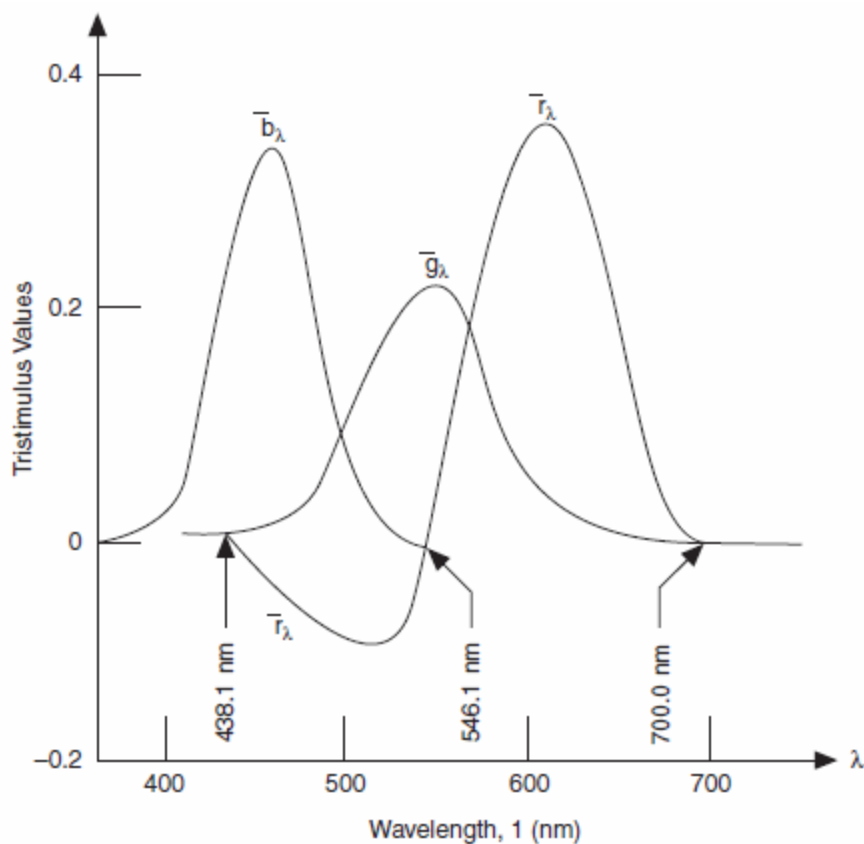
The coordinate system for the HSL color space is cylindrical. Colors are defined inside a hexcone, as shown in the ***color space used to generate the spectrum*** section. The hue value runs from 0 to 360°. The saturation ranges from 0 to 1, where 1 represents the purest color without any white. Luminance also ranges from 0 to 1, where 0 is black and 1 is white.

Overall, two principal factors—the de-coupling of the intensity component from the color information and the close relationship between chromaticity and human perception of color—make the HSL space ideal for developing machine vision applications.

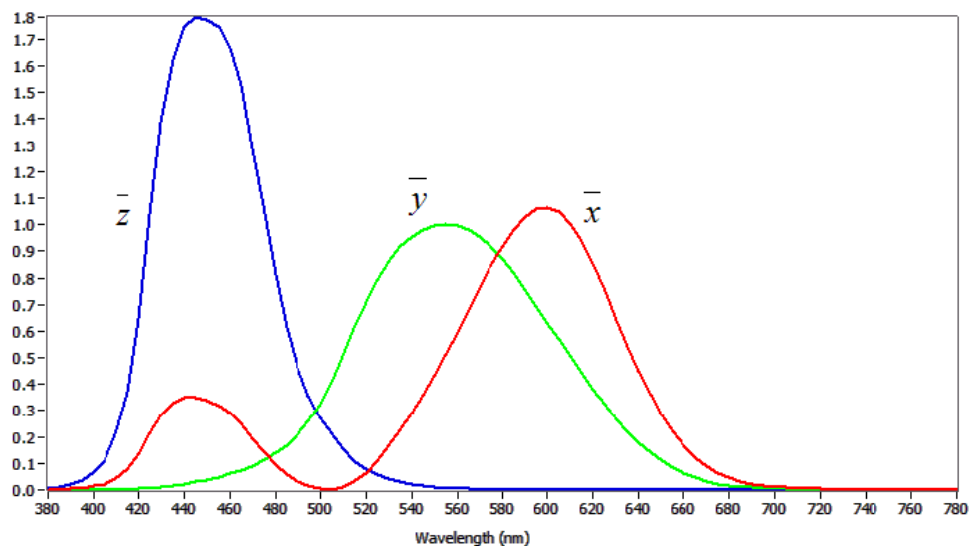
CIE XYZ Color Space

The CIE color space system classifies colors according to the human vision system. This system specifies colors in CIE coordinates and is a standard for comparing one color in the CIE coordinates with another.

Visible light is electromagnetic energy that occupies approximately the 400 nm to 700 nm wavelength part of the spectrum. Humans perceive these wavelengths as the colors violet through indigo, blue, green, yellow, orange, and red. The following figure shows the amounts of red, green, and blue light needed by an average observer to match a color of constant luminance for all values of dominant wavelengths in the visible spectrum. The dominant wavelength is the wavelength of the color humans see when viewing the light. The negative values between 438.1 nm and 546.1 nm indicate that all visible colors cannot be specified by adding together the three positive primaries R, G, and B in the RGB color space.



In 1931, the CIE developed a system of three primary colors (XYZ) in which all visible colors can be represented using a weighted sum of only positive values of X, Y, and Z. The following figure shows the functions used to define the weights of the X, Y, and Z components.



CIE $L^*a^*b^*$ Color Space

CIE 1976 $L^*a^*b^*$, one of the CIE-based color spaces, is a way to linearize the perceptibility of color differences. The nonlinear relations for L^* , a^* , and b^* mimic the logarithmic response of the eye.

CMY Color Space

CMY is another set of familiar primary colors: cyan, magenta, and yellow. CMY is a subtractive color space in which these primary colors are subtracted from white light to produce the desired color. The CMY color space is the basis of most color printing and photography processes. CMY is the complement of the RGB color space because cyan, magenta, and yellow are the complements of red, green, and blue.

YIQ Color Space

The YIQ space is the primary color space adopted by the National Television System Committee (NTSC) for color TV broadcasting. It is a linear transformation of the RGB cube for transmission efficiency and for maintaining compatibility with monochrome television standards. The Y component of the YIQ system provides all the video information that a monochrome television set requires. The main advantage of the YIQ space for image processing is that the luminance information (Y) is de-coupled from the color information (I and Q). Because luminance is proportional to the amount of light perceived by the eye, modifications to the grayscale appearance of the image do not affect the color information.

Related concepts:

- [Color Spectrum](#)

Color Spectrum

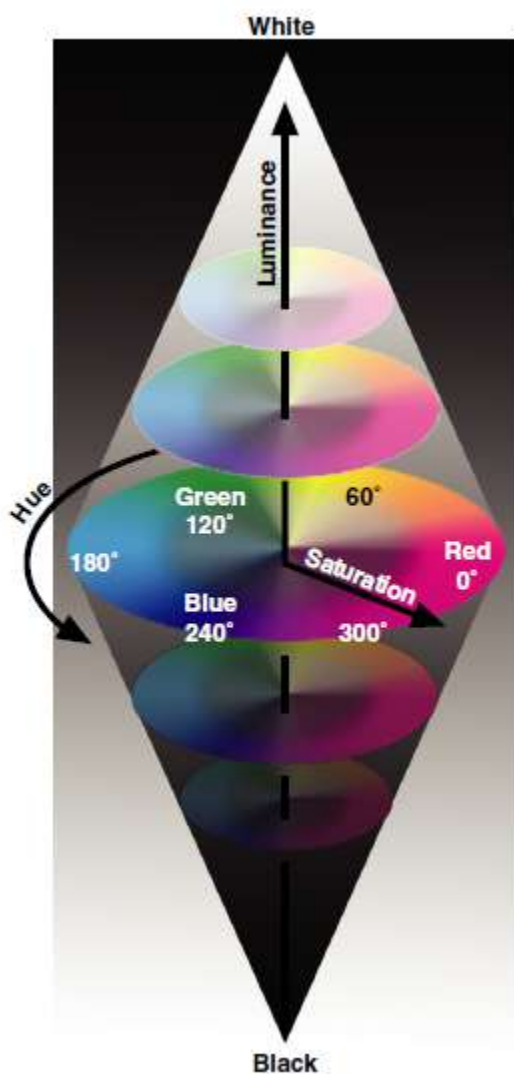
The color spectrum represents the 3D color information associated with an image or a region of an image in a concise 1D form that can be used by many of the Vision color processing functions. Use the color spectrum for color matching, color location, and color pattern matching applications with Vision.

The color spectrum is a 1D representation of the 3D color information in an image. The

spectrum represents all the color information associated with that image or a region of the image in the HSL space. The information is packaged in a form that can be used by the color processing functions in Vision.

Color Space Used to Generate the Spectrum

The color spectrum represents the color distribution of an image in the HSL space, as shown in the following figure. If the input image is in RGB format, the image is first converted to HSL format and the color spectrum is computed from the HSL space. Using HSL images directly—those acquired with an image acquisition device with an onboard RGB to HSL conversion for color matching—improves the operation speed.

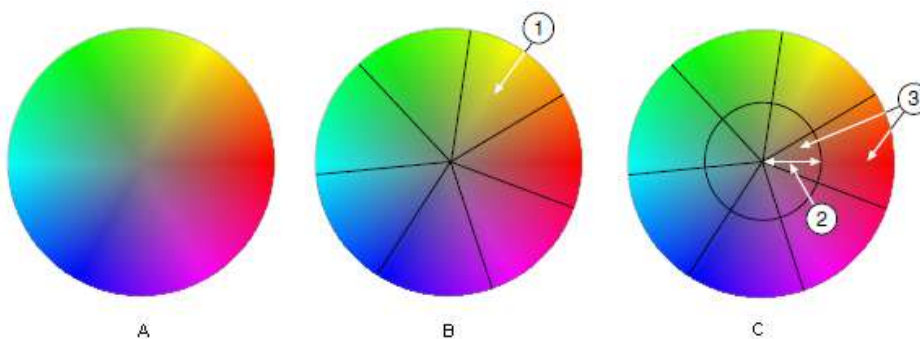


Colors represented in the HSL model space are easy for humans to quantify. The

luminance—or intensity—component in the HSL space is separated from the color information. This feature leads to a more robust color representation independent of light intensity variation. However, the chromaticity—or hue and saturation—plane cannot be used to represent the black and white colors that often comprise the background colors in many machine vision applications. Refer to the **color pattern matching** section for more information about color spaces.

Generating the Color Spectrum

Each element in the color spectrum array corresponds to a bin of colors in the HSL space. The last two elements of the array represent black and white colors, respectively. The following figure illustrates how the HSL color space is divided into bins. The hue space is divided into a number of equal sectors, and each sector is further divided into two parts: one part representing high saturation values and another part representing low saturation values. Each of these parts corresponds to a color bin—an element in the color spectrum array.

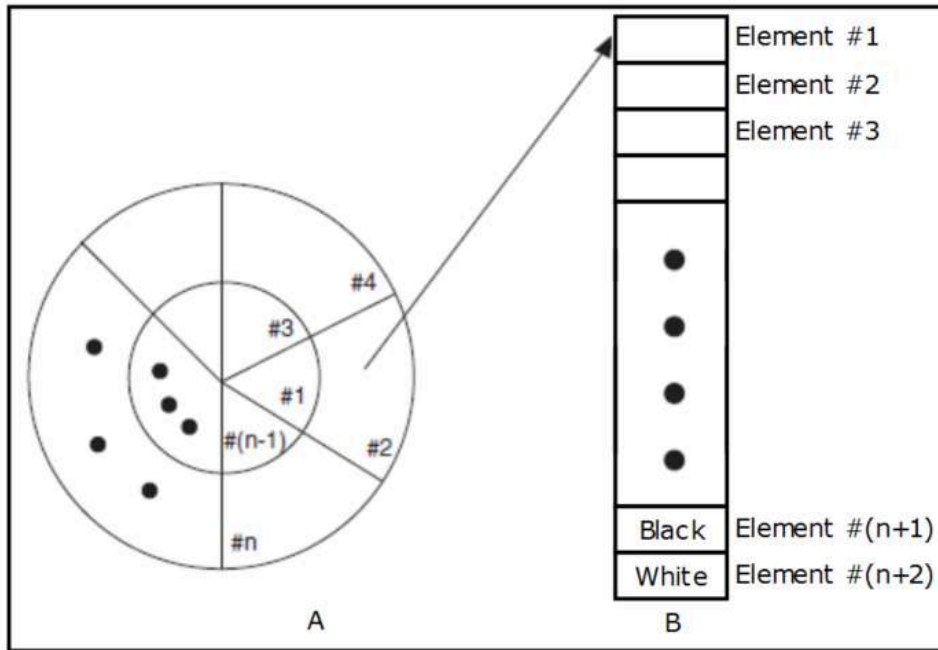


1. Sector
2. Saturation Threshold
3. Color Bins

The color sensitivity parameter determines the number of sectors the hue space is divided into. Figure A shows the hue color space when luminance is equal to 128. Figure B shows the hue space divided into a number of sectors, depending on the desired color sensitivity. Figure C shows each sector divided further into a high saturation bin and a low saturation bin. The saturation threshold determines the radius of the inner circle that separates each sector into bins.

The following figure illustrates the correspondence between the color spectrum

elements and the bins in the color space. The first element in the color spectrum array represents the high saturation part in the first sector; the second element represents the low saturation part; the third element represents the high saturation part of the second sector and so on. If there are n bins in the color space, the color spectrum array contains $n + 2$ elements. The last two components in the color spectrum represent the black and white color, respectively.



A color spectrum with a larger number of bins, or elements, represents the color information in an image with more detail, such as a higher color resolution, than a spectrum with fewer bins. In NI Vision, you can choose between three color sensitivity settings—low, medium, and high. Low divides the hue color space into seven sectors, giving a total of $2 \times 7 + 2 = 16$ bins. Medium divides the hue color space into 14 sectors, giving a total of $2 \times 14 + 2 = 30$ bins. High divides the hue color space into 28 sectors, giving a total of $2 \times 28 + 2 = 58$ bins.

The value of each element in the color spectrum indicates the percentage of image pixels in each color bin. When the number of bins is set according to the color sensitivity parameter, the machine vision software scans the image, counts the number of pixels that fall into each bin, and stores the ratio of the count and total number of pixels in the image in the appropriate element within the color spectrum array.

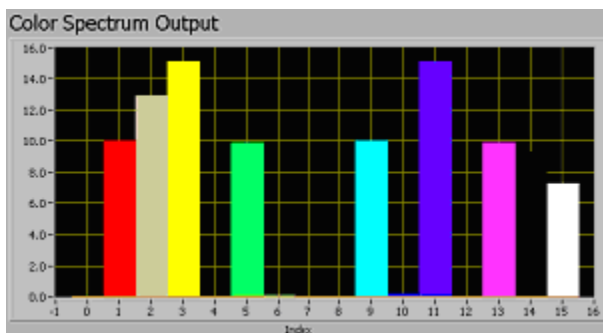
The software also applies a special adaptive learning algorithm to determine if pixels

are either black or white before assigning it to a color bin. Figure B represents the low sensitivity color spectrum of figure A. The height of each bar corresponds to the percentage of pixels in the image that fall into the corresponding bin.

The color spectrum contains useful information about the color distribution in the image. You can analyze the color spectrum to get information such as the most dominant color in the image, which is the element with the highest value in the color spectrum. You also can use the array of the color spectrum to directly analyze the color distribution and for color matching applications.



A



B

Related concepts:

- [Color Pattern Matching](#)

Color Matching

Color matching quantifies which colors and how much of each color exist in a region of an image and uses this information to check if another image contains the same colors

in the same ratio.

Use color matching to compare the color content of an image or regions within an image to a reference color information. With color matching, you create an image or select regions in an image that contain the color information you want to use as a reference. The color information in the image may consist of one or more colors. The machine vision software then learns the 3D color information in the image and represents this information as a 1D color spectrum. Your machine vision application compares the color information in the entire image or regions in the image to the learned color spectrum, calculating a score for each region. The score relates how closely the color information in the image region matches the information represented by the color spectrum.

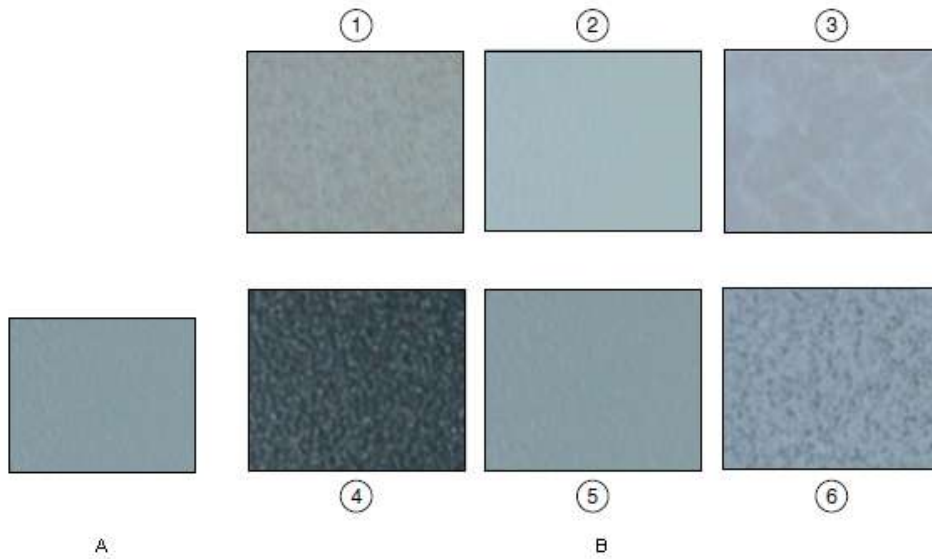
When to Use

Color matching can be used for applications such as color identification, color inspection, color object location and other applications that require the comparison of color information to make decisions.

Color Identification

Color identification identifies an object by comparing the color information in the image of the object to a database of reference colors that correspond to pre-defined object types. The object is assigned a label corresponding to the object type with closest reference color in the database. Use color matching to first learn the color information of all the pre-defined object types. The color spectrums associated with each of the pre-defined object types become the reference colors. Your machine vision application then uses color matching to compare the color information in the image of the object to the reference color spectrums. The object receives the label of the color spectrum with the highest match score.

The following figure shows an example of a tile identification application. Figure A shows the image of a tile that needs to be identified. Figure B shows the scores obtained using color matching with a set of the reference tiles.



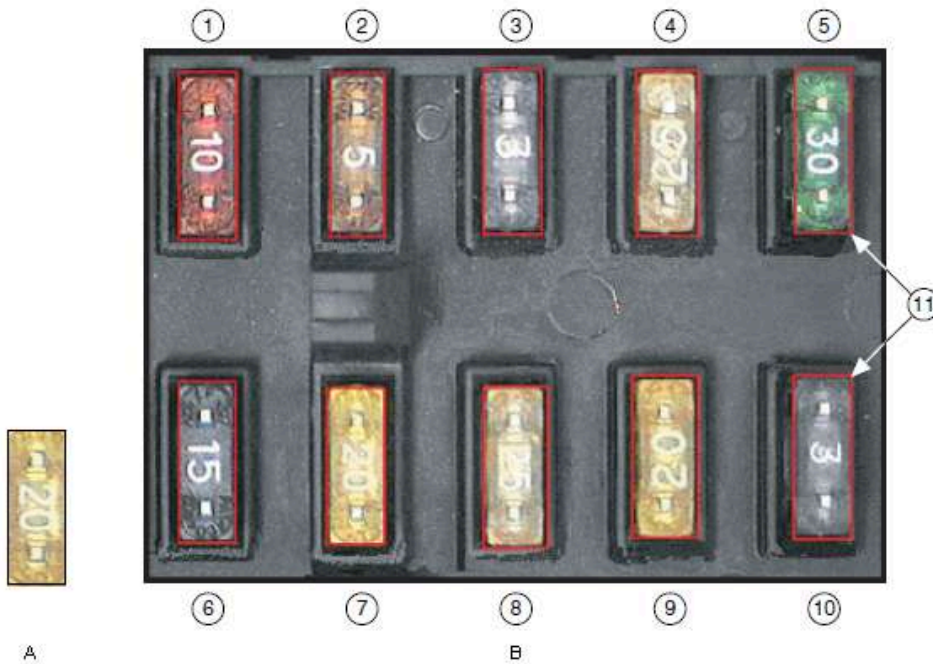
1. Score = 592
2. Score = 6
3. Score = 31
4. Score = 338
5. Score = 1000
6. Score = 405

Use color matching to verify the presence of correct components in automotive assemblies. An example of a color identification task is to ensure that the color of the fabric in the interior of a car adheres to specifications.

Color Inspection

Color inspection detects simple flaws such as missing or misplaced color components, defects on the surfaces of color objects, or printing errors on color labels. You can use color matching for these applications if known regions of interest predefine the object or areas to be inspected in the image. You can define these regions, or they can be the output of some other machine vision tool, such as pattern matching.

The layout of the fuses in junction boxes in automotive assemblies is easily defined by regions of interest. Color matching determines if all of the fuses are present and in the correct locations. The following figure shows an example of a fuse box inspection application in which the exact location of the fuses in the image can be specified by regions of interest. Color matching compares the color of the fuse in each region to the color that is expected to be in that region.



1. Score = 51
2. Score = 382
3. Score = 23
4. Score = 649
5. Score = 29
6. Score = 70
7. Score = 1000
8. Score = 667
9. Score = 990
10. Score = 8
11. Inspection Ranges

Color matching can be used to inspect printed circuit boards containing a variety of components including diodes, resistors, integrated circuits, and capacitors. In a manufacturing environment, color matching can find flaws in a manufactured product when the flaws are accompanied by a color change.

Concepts

Color matching is performed in two steps. In the first step, the machine vision software learns a reference color distribution. In the second step, the software compares color information from other images to the reference image and returns a score as an

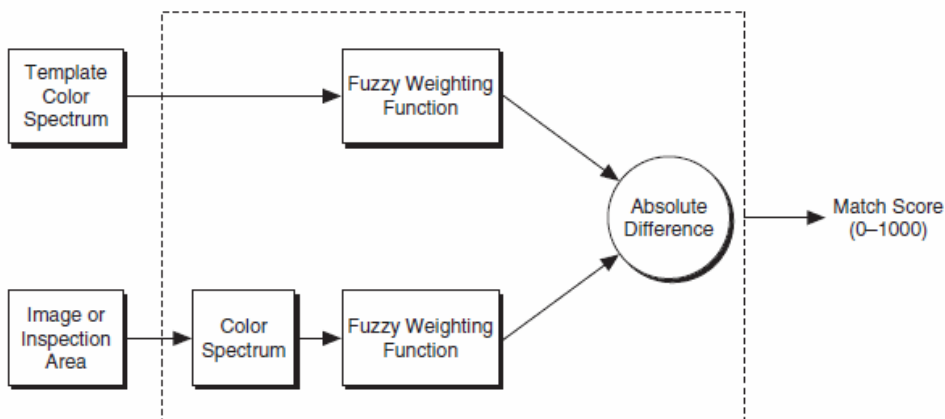
indicator of similarity.

Learning Color Distribution

The machine vision software learns a color distribution by generating a color spectrum. You provide the software with an image or regions in the image containing the color information that you want to use as a reference in your application. The machine vision software then generates a color spectrum based on the information you provide. The color spectrum becomes the basis of comparison during the matching phase.

Comparing Color Distributions

During the matching phase, the color spectrum obtained from the target image or region in the target image is compared to the reference color spectrum taken during the learning step. A match score is computed based on the similarity between these two color spectrums using the Manhattan distance between two vectors. A fuzzy membership weighting function is applied to both the color spectrums before computing the distance between them. The weighting function compensates for some errors that may occur during the binning process in the color space. The fuzzy color comparison approach provides a robust and accurate quantitative match score. The match score, ranging from 0 to 1000, defines the similarity between the color spectrums. A score of zero represents no similarity between the color spectrums, whereas a score of 1000 represents a perfect match. The following figure illustrates the comparison process.



Color Location

Use color location to quickly locate known color regions in an image. With color location, you create a model or template that represents the colors that you are searching. Your machine vision application then searches for the model in each acquired image, and calculates a score for each match. The score indicates how closely the color information in the model matches the color information in the found regions.

When to Use

Color can simplify a monochrome visual inspection problem by improving contrast or separating the object from the background. Color location algorithms provide a quick way to locate regions in an image with specific colors.

Use color location when your application has the following characteristics.

- Requires the location and the number of regions in an image with their specific color information.
- Relies on the cumulative color information in the region, instead of how the colors are arranged in the region.
- Does not require the orientation of the region.
- Does not require the location with subpixel accuracy.

The color location tools in Vision measure the similarity between an idealized representation of a feature, called a model, and a feature that may be present in an image. A feature for color location is defined as a region in an image with specific colors.

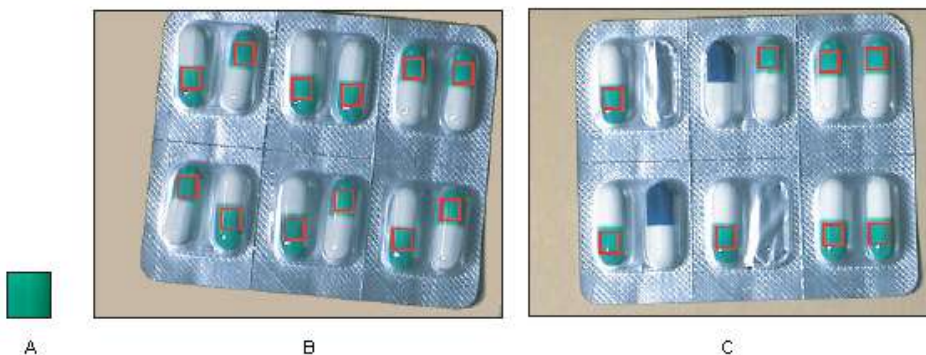
Color location is useful in many applications. Color location provides your application with information about the number of instances and locations of the template within an image. Use color location in the following general applications—inspection, identification, and sorting.

Inspection

Inspection detects flaws such as missing components, incorrect printing, and incorrect fibers on textiles. A common pharmaceutical inspection application is inspecting a blister pack for the correct pills. Blister pack inspection involves checking that all the

pills are of the correct type, which is easily performed by checking that all the pills have the same color information. Because your task is to determine if there are a fixed number of the correct pills in the pack, color location is a very effective tool.

Figure A shows the template image of the part of the pill that contains the color information that you want to locate. Figure B shows the pills located in a good blister pack. Figure C shows the pills located when a blister pack contains the wrong type of pills or missing pills. Because the exact locations of the pills is not necessary for the inspection, the number of matches returned by color location indicates whether a blister pack passes inspection.



Identification

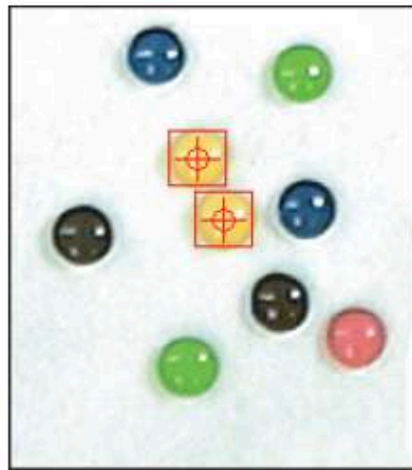
Identification assigns a label to an object based on its features. In many applications, the color-coded identification marks are placed on the objects. In these applications, color matching locates the color code and identifies the object. In a spring identification application, different types of springs are identified by a collection of color marks painted on the coil. If you know the different types of color patches that are used to mark the springs, color location can find which color marks appear in the image. You then can use this information to identify the type of spring.

Sorting

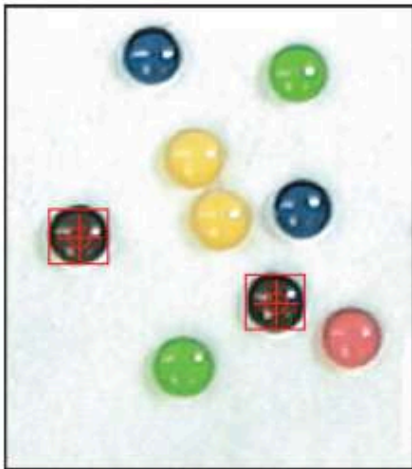
Sorting separates objects based on attributes such as color, size, and shape. In many applications, especially in the pharmaceutical and plastic industries, objects are sorted according to color, such as pills and plastic pellets. The following figure shows an example of how to sort different colored candies. Using color templates of the different candies in the image, color location quickly locates the positions of the different candies.



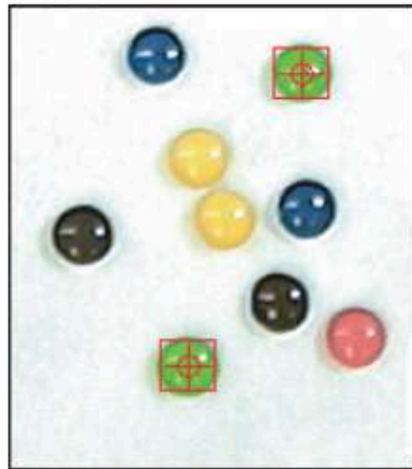
A



B



C



D

What to Expect from a Color Location Tool

In automated machine vision applications, the visual appearance of inspected materials or components changes because of factors such as orientation of the part, scale changes, and lighting changes. The color location tool maintains its ability to locate the reference patterns despite these changes. The color location tool provides accurate results during the following common situations: pattern orientation and multiple instances, ambient lighting conditions, and blur and noise conditions.

Pattern Orientation and Multiple Instances

A color location tool locates the reference pattern in an image even if the pattern in the image is rotated or scaled. When a pattern is rotated or slightly scaled in the image, the color location tool can detect the following:

- The pattern in the image
- The position of the pattern in the image
- Multiple instances of the pattern in the image, if applicable

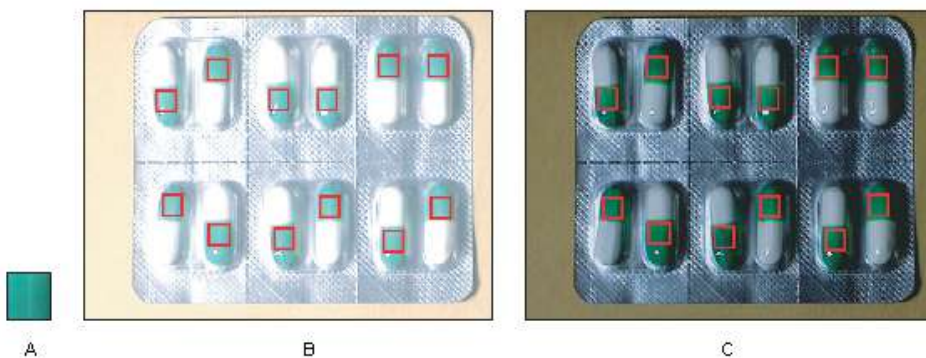
Because color location only works on the color information of a region and does not use any kind of shape information from the template, it does not find the angle of the rotation of the match. It only locates the position of a region in the image whose size matches a template containing similar color information.

Refer to the inspection section for an example illustrating pattern orientation and multiple instances.

Ambient Lighting Conditions

The color location tool finds the reference pattern in an image under conditions of uniform changes in the lighting across the image. Color location also finds patterns under conditions of non-uniform light changes, such as shadows.

The following figure shows typical conditions under which the color location tool works correctly. Figure A shows the original template image. Figure B shows the same pattern under bright light. Figure C shows the pattern under poor lighting.



Blur and Noise Conditions

Color location finds patterns that have undergone some transformation because of blurring or noise. Blurring usually occurs because of incorrect focus or depth of field changes.

Concepts

Color location is built upon the **color matching** functions to quickly locate regions with specific color information in an image.

The color location functions extend the capabilities of color matching to applications in which the location of the objects in the image is unknown. Color location uses the color information in a template image to look for occurrences of the template in the search image. The basic operation moves the template across the image pixel by pixel and comparing the color information at the current location in the image to the color information in the template using the color matching algorithm.

The color location process consists of two main steps—learning template information and searching for the template in an image. The following figure illustrates the general flow of the color location process. During the learning phase, the software extracts the color spectrum from the template image. This color spectrum is used to compare the color information of the template with the color information in the image.

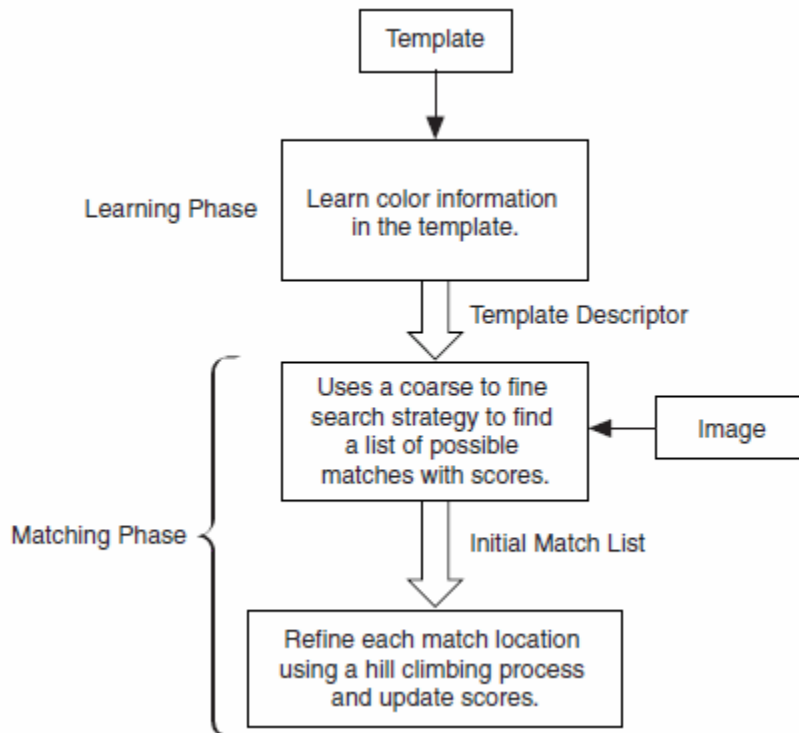
During the search step, a region the size of the template is moved across the image pixel by pixel from the top of the image to the bottom. At each pixel, the function computes the color spectrum of the region under consideration. This color spectrum is then compared with the template's color spectrum to compute a match score.

The search step is divided into two phases. First, the software performs a coarse-to-fine search phase that identifies all possible locations, even those with very low match scores. The objective of this phase is to quickly find possible locations in the image that may be potential matches to the template information. Because stepping through the image pixel by pixel and computing match scores is time consuming, the following techniques are used to speed up the search process.

- **Subsampling**—When stepping through the image, the color information is taken from only a few sample points in the image to use for comparison with the template. This reduces the amount of data used to compute the color spectrum in the image, which speeds up the search process.
- **Step size**—Instead of moving the template across the image pixel by pixel, the search process skips a few pixels between the each color comparison, thus speeding up the search process. The step size indicates the number of pixels to skip. For color location, the initial step size can be as large as half the size of the

template.

The initial search phase generates a list of possible match locations in the image. In the second step, that list is searched for the location of the best match using a hill-climbing algorithm.



Related concepts:

- [Color Matching](#)

Color Pattern Matching

Use color pattern matching to quickly locate known reference patterns, or fiducials, in a color image. With color pattern matching, you create a model or template that represents the object you are searching for. Then your machine vision application searches for the model in each acquired image, calculating a score for each match. The score indicates how closely the model matches the color pattern found. Use color pattern matching to locate reference patterns that are fully described by the color and spatial information in the pattern.

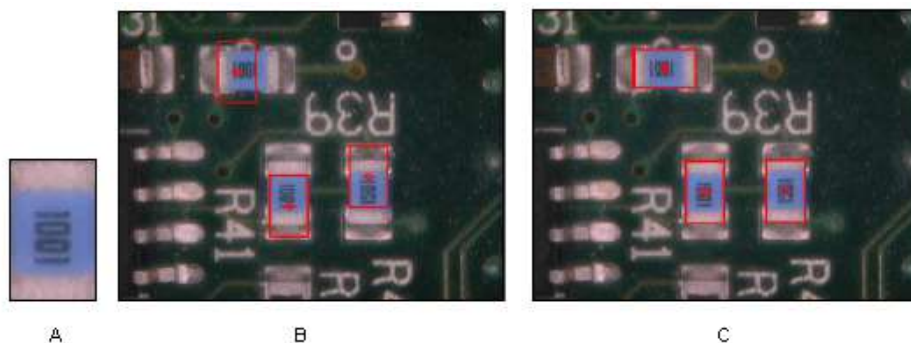
When to Use

Grayscale, or monochrome, **pattern matching** is a well-established tool for alignment, gauging, and inspection applications. In all of these application areas, color simplifies a monochrome problem by improving contrast or separation of the object from the background. Color pattern matching algorithms provide a quick way to locate objects when color is present.

Use color pattern matching when the object under inspection has the following qualities:

- The object contains color information that is very different from the background, and you want to find the location of the object in the image very precisely. For these applications, color pattern matching provides a more accurate solution than color location—because color location does not use shape information during the search phase, finding the locations of the matches with pixel accuracy is difficult.
- The object has grayscale properties that are difficult to characterize or that are very similar to other objects in the search image. In such cases, grayscale pattern matching may not give accurate results. If the object has some color information that differentiates it from the other objects in the scene, color provides the machine vision software with the additional information to locate the object.

The following figure illustrates the advantage of using color pattern matching over color location to locate the resistors in an image. Although color location finds the resistors in the image, the matches are not very accurate because they are limited to color information. Color pattern matching uses color matching first to locate the objects, and then pattern matching to refine the locations, providing more accurate results.



The following figure shows the advantage of using color information when locating color-coded fuses on a fuse box. Figure A shows a grayscale image of the fuse box. In the image of the fuse box in figure A, the grayscale pattern matching tool has difficulty clearly differentiating between fuse 20 and fuse 25 and will return close match scores because of similar grayscale intensities and the translucent nature of the fuses. In the color image, figure B, the addition of color helps to improve the accuracy and reliability of the pattern matching tool.



The color pattern matching tools in Vision measure the similarity between an idealized representation of a feature, called a model, and the feature that may be present in an image. A feature is defined as a specific pattern of color pixels in an image.

Color pattern matching is the key to many applications. Color pattern matching provides your application with information about the number of instances and location of the template within an image. Use color pattern matching in the following three general applications: gauging, inspection, and alignment.

Gauging

Many gauging applications locate and then measure or gauge the distance between objects. Searching and finding a feature is the key processing task that determines the success of many gauging applications. If the components you want to gauge are uniquely identified by their color, color pattern matching provides a fast way to locate the components.

Inspection

Inspection detects simple flaws, such as missing parts or unreadable printing. A common application is inspecting the labels on consumer product bottles for printing

defects. Because most of the labels are in color, color pattern matching is used to locate the labels in the image before a detailed inspection of the label is performed. The score returned by the color pattern matching tool also can be used to decide whether a label is acceptable.

Alignment

Alignment determines the position and orientation of a known object by locating fiducials. Use the fiducials as points of reference on the object. Grayscale pattern matching is sufficient for most applications, but some alignment applications require color pattern matching for more reliable results.

What to Expect from a Color Pattern Matching Tool

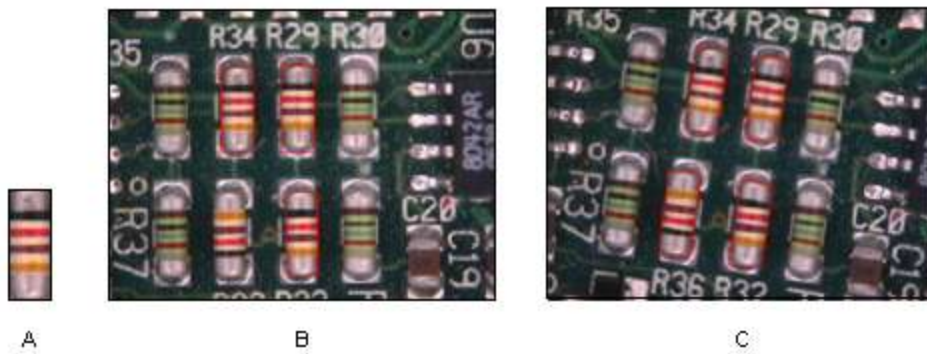
In automated machine vision applications, the visual appearance of materials or components under inspection can change due to factors such as orientation of the part, scale changes, and lighting changes. The color pattern matching tool maintains its ability to locate the reference patterns and gives accurate results despite these changes.

Pattern Orientation and Multiple Instances

A color pattern matching tool locates the reference pattern in an image even when the pattern in the image is rotated and slightly scaled. When a pattern is rotated or scaled in the image, the color pattern matching tool detects the following features of an image:

- The pattern in the image
- The position of the pattern in the image
- The orientation of the pattern
- Multiple instances of the pattern in the image, if applicable

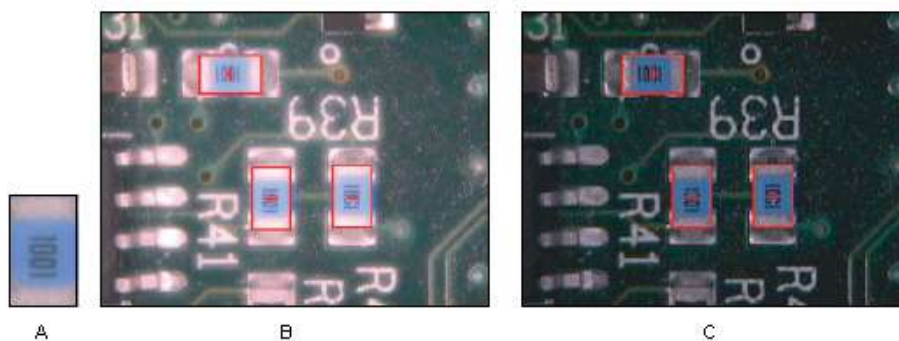
Figure A shows a template image, or pattern. Figures B and C illustrate multiple occurrences of the template. Figure B shows the template shifted in the image. Figure C shows the template rotated in the image.



Ambient Lighting Conditions

The color pattern matching tool finds the reference pattern in an image under conditions of uniform changes in the lighting across the image. Because color analysis is more robust when dealing with variations in lighting than grayscale processing, color pattern matching performs better under conditions of non-uniform light changes, such as in the presence of shadows, than grayscale pattern matching.

Figure A shows the original template image. Figure B shows the same pattern under bright light. Figure C shows the pattern under poor lighting.



Blur and Noise Conditions

Color pattern matching finds patterns that have undergone some transformation because of blurring or noise. Blurring usually occurs because of incorrect focus or depth of field changes.

Concepts

Color pattern matching is a unique approach that combines color and spatial information to quickly find color patterns in an image. It uses the technologies behind

color matching and grayscale pattern matching in a synergistic way to locate color patterns in color images.

Color Matching and Color Location

Color matching compares the color content of an image or regions in an image to existing color information. The color information in the image may consist of one or more colors. To use color matching, define regions in an image that contain the color information you want to use as a reference. The machine vision functions then learn the 3D color information in the image and represents it as a 1D color spectrum. Your machine vision application compares the color information in the entire image or regions in the image to the learned color spectrum, calculating a score for each region. This score relates how closely the color information in the image region matches the information represented by the color spectrum. To use color matching, you need to know the location of the objects in the image before performing the match.

Color location functions extend the capabilities of **color matching** to applications where you do not know the location of the objects in the image. Color location uses the color information from a template image to look for occurrences of the template in the search image. The basic operation moves the template across the image pixel by pixel and compares the color information at the current location in the image to the color information in the template, using the color matching algorithm. Because searching an entire image for color matches is time consuming, the color location software uses some techniques to speed up the location process. A coarse-to-fine search strategy finds the rough locations of the matches in the image. A more refined search, using a hill climbing algorithm, is then performed around each match to get the accurate location of the match. Color location is an efficient way to look for occurrences of regions in an image with specific color attributes.

Grayscale Pattern Matching

Vision grayscale **pattern matching** methods incorporate image understanding techniques to interpret the template information and use that information to find the template in the image. Image understanding refers to image processing techniques that generate information about the features of a template image. These methods include the following:

- Geometric modeling of images

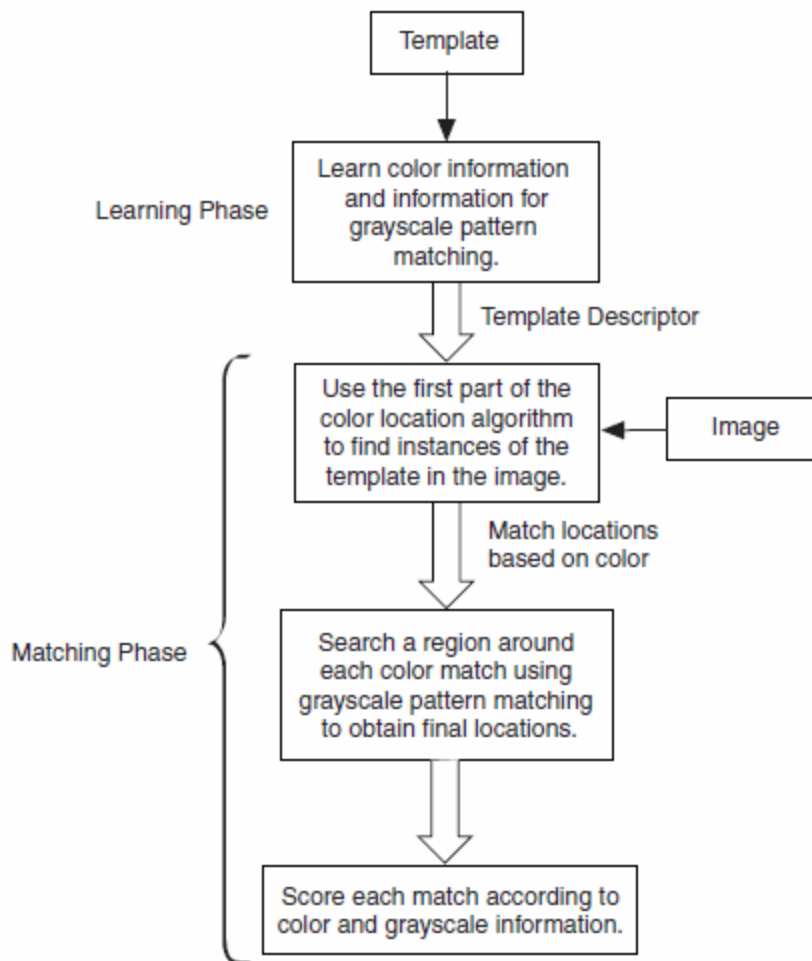
- Efficient non-uniform sampling of images
- Extraction of rotation-independent template information

Vision uses a combination of the edge information in the image and an intelligent image sampling technique to match patterns. The image edge content provides information about the structure of the image in a compact form. The intelligent sampling technique extracts points from the template that represent the overall content of the image. The edge information and intelligent sampling technique reduce the inherently redundant information in an image and improve the speed and accuracy of the pattern matching tool. In cases where the pattern can be rotated in the image, a similar technique is used, but with specially chosen template pixels whose values, or relative change in values, reflect the rotation of the pattern. The result is fast and accurate grayscale pattern matching.

Vision pattern matching accurately locates objects in conditions where they vary in size ($\pm 5\%$) and orientation (between 0° and 360°) and when their appearance is degraded.

Combining Color Location and Grayscale Pattern Matching

Color pattern matching uses a combination of color location and grayscale pattern matching to search for the template. When you use color pattern matching to search for a template, the software uses the color information in the template to look for occurrences of the template in the image. The software then applies grayscale pattern matching in a region around each of these occurrences to find the exact position of the template in the image. The following figure illustrates the general flow of the color pattern matching algorithm. The size of the searchable region is determined by the software, based on the inputs you provide, such as search strategy and color sensitivity.



In-Depth Discussion

There are standard ways to convert RGB to grayscale and to convert one color space to another. The transformation from RGB to grayscale is linear. However, some transformations from one color space to another are nonlinear because some color spaces represent colors that cannot be represented in other spaces.

RGB to Grayscale

The following equations convert an RGB image into a grayscale image on a pixel-by-pixel basis:

$$\text{grayscale value} = 0.299R + 0.587G + 0.114B$$

This equation is part of the NTSC standard for luminance. An alternative conversion from RGB to grayscale is a simple average:

$$\text{grayscale value} = (R + G + B) / 3$$

RGB and HSL

There is no matrix operation that allows you to convert from the RGB color space to the HSL color space. The following equations describe the nonlinear transformation that maps the RGB color space to the HSL color space.

$$V2 = \sqrt{3}(G - B)$$

$$V1 = 2R - G - B$$

$$\varphi = \begin{cases} \tan^{-1}(V2/V1) & \text{if } V1 \neq 0 \\ \frac{\pi}{2} & \text{else if } V1 = 0, V2 > 0 \\ -\frac{\pi}{2} & \text{else if } V1 = 0, V2 < 0 \\ 0 & \text{otherwise} \end{cases}$$

$$H = \begin{cases} 256 \times \frac{\varphi}{2\pi} & \text{if } V1 \geq 0, V2 \geq 0 \\ 256 \times \frac{(2\pi + \varphi)}{2\pi} & \text{else if } V1 < 0 \\ 256 \times \frac{(2\pi + \varphi)}{\pi} & \text{otherwise} \end{cases}$$

$$L = 0.299R + 0.587G + 0.114B$$

$$S = 255(1 - 3\min(R, G, B) / (R + G + B))$$

The following equations map the HSL color space to the RGB color space:

$$h = H \frac{2\pi}{256}$$

$$s = S/255$$

$$s' = (1 - s)/3$$

$$f(h) = (1 - s \cdot \cos(h)/\cos(\pi/3 - h))/3$$

$$\left. \begin{array}{l} b = s' \\ r = f(h) \\ g = 1 - r - b \end{array} \right\} [0 < h \leq 2\pi/3]$$

$$\left. \begin{array}{l} h' = h - 2\pi/3 \\ r = s' \\ g = f(h') \\ b = 1 - r - g \end{array} \right\} [2\pi/3 < h \leq 4\pi/3]$$

$$\left. \begin{array}{l} h' = h - 4\pi/3 \\ g = s' \\ b = f(h') \\ r = 1 - g - b \end{array} \right\} [4\pi/3 < h \leq 2\pi]$$

$$1 = 0.299r + 0.587g + 0.114b$$

$$1' = L/l$$

$$R = r'$$

$$G = g'$$

$$B = b'$$

RGB and CIE XYZ

The following 3×3 matrix converts RGB to CIE XYZ without applying gamma correction.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

By projecting the tristimulus values on to the unit plane $X + Y + Z = 1$, color can be expressed in a 2D plane. The chromaticity coordinates are defined as follows:

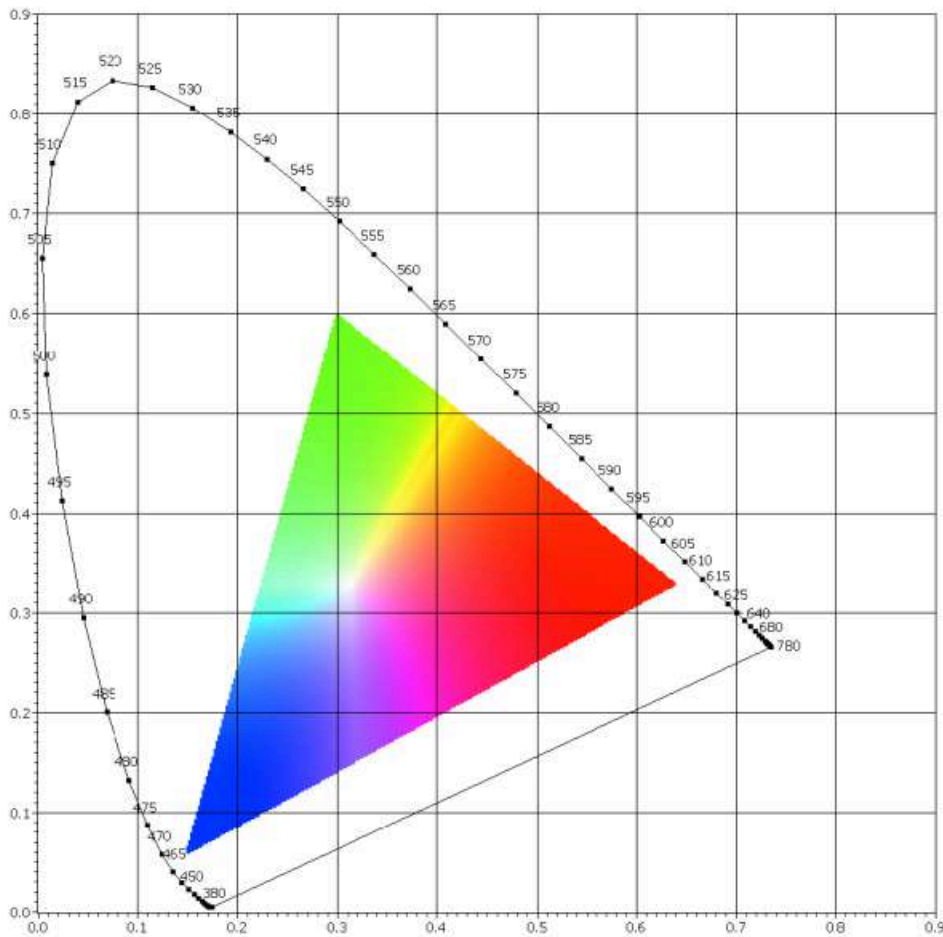
$$x = X / (X + Y + Z)$$

$$y = Y / (X + Y + Z)$$

$$z = Z / (X + Y + Z)$$

You can obtain z from x and y by $z = 1 - x - y$. Hence, chromaticity coordinates are usually given as (x, y) only. The chromaticity values depend on the hue or dominant wavelength and the saturation. Chromaticity values are independent of luminance.

The diagram from (x, y) is referred to as the CIE 1931 chromaticity diagram, or the CIE (x, y) chromaticity diagram, as illustrated in the bell curve of the following figure.



The three color components R, G, and B define a triangle inside the CIE diagram of the previous figure. Any color within the triangle can be formed by mixing R, G, and B. The triangle is called a gamut. Because the gamut is only a subset of the CIE color space, combinations of R, G, and B cannot generate all visible colors.

To transform values back to the RGB space from the CIE XYZ space, use the following matrix operation:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 3.240479 & -1.537150 & -0.498535 \\ -0.969256 & 1.875992 & 0.041556 \\ 0.055648 & -0.204043 & 1.057311 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Notice that the transform matrix has negative coefficients. Therefore, some XYZ color may transform into R, G, B values that are negative or greater than one. This means

that not all visible colors can be produced using the RGB color space.

RGB and CIE $L^*a^*b^*$

To transform RGB to CIE $L^*a^*b^*$, you first must transform the RGB values into the CIE XYZ space. Use the following equations to convert the CIE XYZ values into the CIE $L^*a^*b^*$ values.

$$L^* = 116 \times (Y/Y_n)^{1/3} - 16 \text{ for } Y/Y_n > 0.008856$$

$$L^* = 903.3 \times Y / Y_n \text{ otherwise}$$

$$a^* = 500(f(X / X_n) - f(Y / Y_n))$$

$$b^* = 200(f(Y / Y_n) - f(Z / Z_n))$$

where,

$$f(t) = t^{1/3} \text{ for } t > 0.008856$$

$$f(t) = 7.787t + 16/116 \text{ otherwise}$$

Here X_n , Y_n , and Z_n are the tri-stimulus values of the reference white.

L^* represent the light intensity. Vision normalizes the result of the L^* transformation to range from 0 to 255. The hue and chroma can be calculated as follows:

$$\text{Hue} = \tan^{-1}(b^* / a^*)$$

$$\Delta E^*_{ab} = \sqrt{(a^*)^2 + (b^*)^2}$$

Based on the fact that the color space is now approximately uniform, a color difference formula can be given as the Euclidean distance between the coordinates of two colors in the CIE $L^*a^*b^*$.

$$\text{Chroma} = \sqrt{(\Delta L^*)^2 + (\Delta a^*)^2 + (\Delta b^*)^2}$$

To transform CIE $L^*a^*b^*$ values to RGB, first convert the CIE $L^*a^*b^*$ values to CIE XYZ using the following equations:

$$X = X_n \left(P + a^* / 500 \right)^3$$

$$Y = Y_n P^3$$

$$Z = Z_n \left(P - b^* / 200 \right)^3$$

where

$$P = (L^* + 16) / 116$$

Then, use the conversion matrix given in the RGB and CIE XYZ section to convert CIE XYZ to RGB.

RGB and CMY

The following matrix operation converts the RGB color space to the CMY color space.

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Normalize all color values to lie between 0 and 1 before using this conversion equation. To obtain RGB values from a set of CMY values, subtract the individual CMY values from 1.

RGB and YIQ

The following matrix operation converts the RGB color space to the YIQ color space.

$$\begin{bmatrix} Y \\ U \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 10.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

The following matrix operation converts the YIQ color space to the RGB color space.

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.0 & 0.956 & 0.621 \\ 1.0 & -0.272 & -0.647 \\ 1.0 & -1.105 & 1.702 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$$

Related concepts:

- [Pattern Matching](#)
- [Color Location](#)
- [Color Matching](#)

Color Segmentation

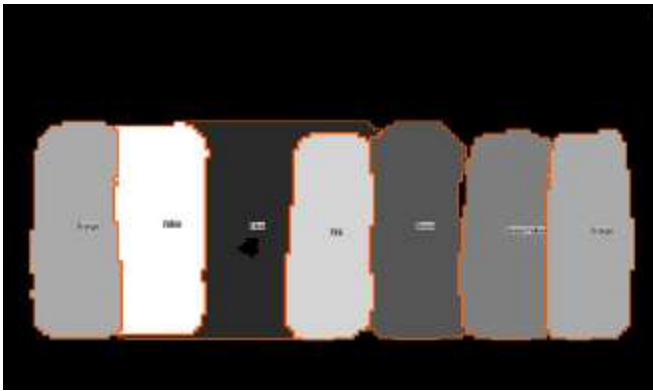
Color segmentation compares the color feature of each pixel with the color features of surrounding pixels or a trained **color classifier** to segment an image into color regions. Use color segmentation to separate color objects of interest from background clutter.

You can use color segmentation in a wide variety of machine vision applications, such as the following:

- **Inspection**—Partition an image into different regions based on the color of the part in each region.
- **Counting**—Segment an image to quickly count the number of objects with a particular color composition. The following figures illustrate a color segmentation that can be used to count the number of bottles that contain each color of liquid.



A

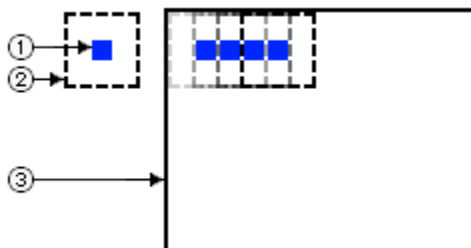


B

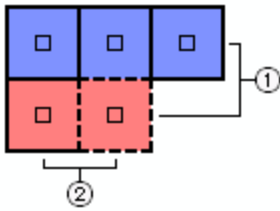
Concepts

Color segmentation involves three stages.

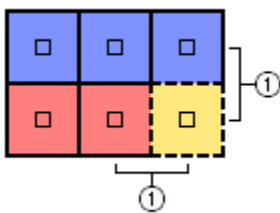
1. Train a **color classifier** with color samples for your application.
2. Segment an image into different color regions. Color segmentation consists of the following steps.
 - a. Move an inspection window across the image to calculate the color feature of each pixel.



1. Pivot Pixel
 2. Inspection Window
 3. Image
- b. Compare the color feature for each inspection window with the color feature of neighboring windows.
 - c. If the closest distance between the inspection window and a neighboring window is less than maximum distance, apply the color label from the pivot pixel in the neighboring window to the pivot pixel in the inspection window.



- i. Distance Between Neighboring Color Features Exceeds Maximum Distance
- ii. Distance Between Neighboring Color Features Does Not Exceed MaximumDistance
- d. If the closest distance between the inspection window and a neighboring window is greater than maximum distance, use the color classifier to label the pivot pixel in the inspection window.



1. Distance Between Neighboring Color Features Exceeds Maximum Distance

If the identification score for the inspection window is less than the minimum identification score, the color classification algorithm does not label the pivot pixel.

3. Filter segmented regions to eliminate regions that do not meet the specified size requirements.

In-Depth Discussion

Maximum distance refers to the maximum distance allowed between the color features of pivot pixels with the same color label. Maximum distance is calculated from the trained color classifier as:

$$\text{Maximum Distance} = \text{Distance Between Two Closest Trained Classes} \times \frac{\text{Step Size}}{\text{Window Size}}$$

An aggressive maximum distance defines the distance between the two closest trained classes as the median distance between samples in each class. A conservative maximum distance defines the distance between the two closest trained classes as the smallest distance between samples in each class. A high maximum distance typically allows more pixels to use the color label of neighboring pixels, which avoids using the

color classifier and decreases the time required to perform color segmentation. A high maximum distance reduces the accuracy of color segmentation.

Color segmentation can be time-consuming if it operates on each pixel. To increase the speed of color segmentation increase the step size, which increases the offset between each inspection window, or train color samples at a lower color resolution to reduce the size of the color feature for each color class.

Related concepts:

- [Color Classification](#)

Deep Learning

Introduction

Vision Development Module supports loading and executing third party Deep Learning framework models. The models from the following Deep Learning frameworks are supported.

- TensorFlow Inference Engine
- OpenVINO™ Inference Engine

The Deep Learning Inference Engines enable user to:

- Load pre-trained Deep Learning Models into Software and Hardware ecosystem
- Run loaded models in Windows and Real Time targets
- Supply Vision Image and LabVIEW data to learned models

Supported Platforms

The following platforms are supported:

Development Environments

- LabVIEW 64-bit

TensorFlow Runtime and Real Time Targets Support:

- Windows 64-bit
- NI Linux RT 64-bit

OpenVINO™ Runtime and Real Time Targets Support:

- Windows 64-bit (Windows 7 Embedded Standard is not supported)
- NI Linux RT 64-bit

When to Use

The Deep Learning Inference Engines provide the ability to load and execute third party framework models. These functions can be used when there are pre-trained models present and there is a need to use them along with other Vision functions. They can also be deployed in Real Time targets. The prerequisites for using the Deep Learning Inference Engines are:

- Pre-trained models from supported libraries
- The model must be a Frozen Model or a Saved Model

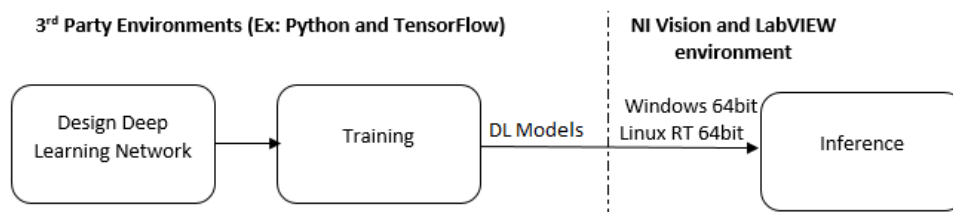


Note Model Training is currently not supported with Vision Development Module.

Deep Learning Inference Engines

Vision Development Module supports TensorFlow and OpenVINO™ Inference Engines.

User Workflow

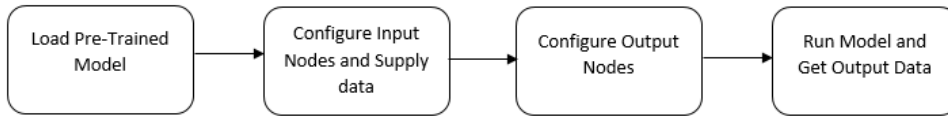


A new Deep Learning Model or topology is created by the developers. This model is then trained using the data acquired from the dataset. The trained model so obtained maybe further deployed on Targets using the LabVIEW APIs provided by the Vision Development Module. For more details about the inference engines workflow, see the

Development Workflow section.

Development Workflow

The following workflow is applicable in LabVIEW using Vision functions.



Supplying Input Data

Deep Learning libraries usually accept data as a Tensor (a representation of multi-dimensional arrays). Once a model is loaded using Vision functions, it understands input and output tensor configurations for the loaded models. Supplied data to the Vision function is converted to input node tensors and fed to the model while running inference. The following table depicts input data compatibility:

NI Data Type	Tensor Data Type	Default Expected Tensor Dimension	Comment
NI Vision Image (U8, U16, I16, SGL)	Unsigned Integer 8 / 16 / Float	4 [1*X*Y*1]	Error displayed for dimension mismatch. Data is converted if tensor expects Float data type. If tensor is not Float, it must be same as Image Type.
NI Vision Image (RGB32)	Unsigned Integer 8 / 16 / Float	4 [1*X*Y*3]	Error displayed for dimension mismatch. Data is converted if tensor expects Float data type. If tensor is not Float, it must be same as Image Type.
Array (U8, I8, I16, U16, I32, I64, Float, Double)	U8, I8, I16, U16, I32, I64, Float, Double	Same as supplied	Error is displayed if there is a mismatch in dimensions or datatype.

NI Vision Image (RGB64, Complex, HSL)	-	-	Unsupported
Array (Complex, U32, U64)	-	-	Unsupported

Interpreting Output Data

The output data from the model is converted into single dimensional Float Array in LabVIEW. The dimensional information of the original data from the graph is also given out. The user needs to construct back the data from this LabVIEW output.

Tensor Data Type	NI Data Type	Comment
Array (U8, I8, I16, U16, U32, I32, U64, I64, Float, Double)	Float	Data is converted. Data loss may result for Double to Float conversion.

Frozen Model(*.pb)

The supported model file format for Frozen Model is Protocol Buffer (.pb). This format is created and maintained by Google™. If Saved Models are supplied, a folder must be provided with Protocol Buffer files and other intermediate files. This is supported only for **TensorFlow**.

Saved Models

These are primarily folders which must be provided with Protocol Buffer files and other intermediate files. This is supported only for **TensorFlow**.

Intermediate Representation (*.xml)

These are primarily xml files with graph information about the model. They are compulsorily accompanied with same named “.bin” file that contains weights and biases relevant for the defined model. This format is supported only for OpenVINO™ Deep Learning and Deployment toolkit and is maintained by Intel.

Reference Link: [**https://en.wikipedia.org/wiki/Protocol_Buffers**](https://en.wikipedia.org/wiki/Protocol_Buffers)

The supported LabVIEW datatypes are:

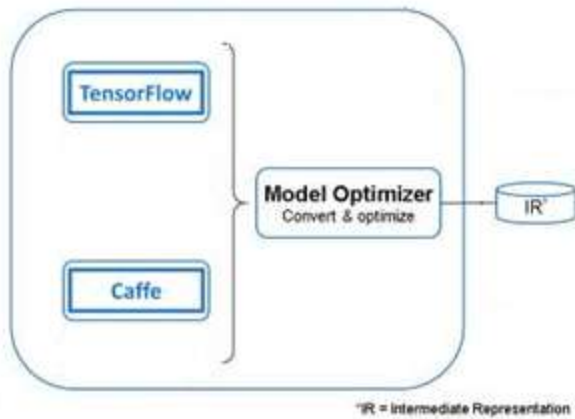
1. NI Vision Image
 - U8, U16, RGB32, SGL
2. LabVIEW Arrays

Model Optimizer (OpenVINO™ only)

Model Optimizer, as a part of OpenVINO™ toolkit is a cross-platform python based command line tool that facilitates the transition between the training and deployment environment, performs static model analysis, and adjusts deep learning models for optimal execution on end-point target devices. The following diagram summarizes the workflow.



Note For more details about converting a model to OpenVINO model using Model Optimizer and accessing the Readme, use the command `cd %NI_MO_INSTALL_PATH%` or go to `C:\Users\Public\Documents\National Instruments\model_optimizer\`



A python script `convert2ir.py` is available as part of the installation which will help users convert models of different topologies easily without requiring to go into complex set of parameters that model optimizer requires to convert, for example, a tensorflow model to an IR model. The model optimizer can convert TensorFlow and Caffe Models.

Related information:

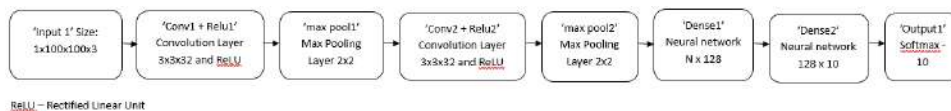
- [TensorFlow](#)
- https://en.wikipedia.org/wiki/Protocol_Buffers

In-Depth Discussion

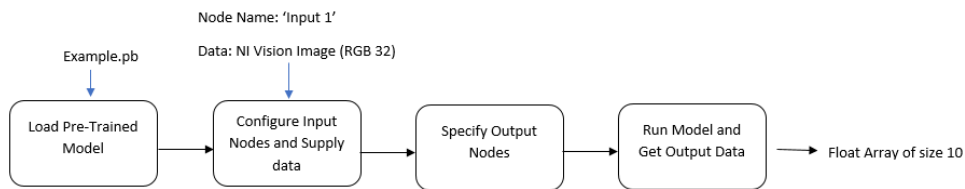
This section provides additional information you may need.

Case Study

The following model is considered for the case study:



The above model/graph accepts a single RGB image of size 100x100 as input and classifies it into 10 classes. For example, let us consider that the graph was saved as **Example.pb** file.



Using High level APIs/ Reference Design APIs

Vision provides the following high-level API for the Deep Learning Interface:

- Classification (IMAQ DL Model Classify Image)

This high-level API provides fixed design for supplying and receiving data. This can be used as a reference API to create a custom high level API.

Classification

The API accepts Vision Image as an Input. In this case the loaded model's input node should accept tensors of similar dimensions. The output node data must be two-dimensional Float array. The following diagram depicts input and output data dimension requirements.



X— Image Width

Y— Image Height

C— Channels (3 for RGB and 1 for Grayscale)

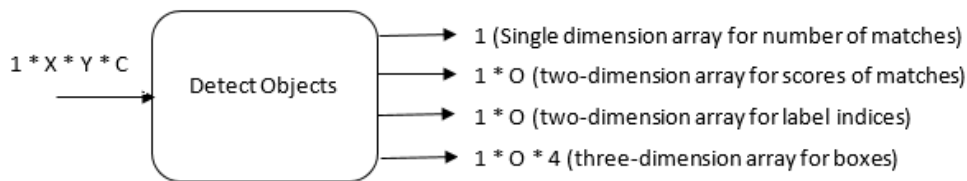
N— Number of Classes

The API will display an error if the input image dimensions does not match with the input node tensor dimensions. It is recommended to develop a graph which accepts any image dimension and resize them in the graph itself.

Object Detection

The API accepts Vision Image as an Input. In this case the loaded model's input should accept tensor of similar dimensions. There must be 4 output nodes and must follow the order below.

- Number of Matches - Float (dimensional float array).
- Confident Scores for Matches - Two Dimensional Float Array.
- Label Index of Matches - Two Dimensional Float Array.
- Locations for Matches - Three Dimensional Foat Array.
 - Each leaf row contains locations as [Top, Left, Bottom, Right].
 - Coordinates ranges between 0 to 1 (unit bounding boxes).



X— Image Width

Y— Image Height

C—Channels (3 RGB and 1 for Grayscale)

O—Number of detected objects

The object detection API converts output node data to LabVIEW data, while doing so it also converts unit bounding boxes to supplied image dimensions. This API will display an error if input image dimensions does not match with the input node tensor dimensions. It is recommended to develop a graph which accepts any image dimension and resize them in the graph itself.

Error and exception handling

Model Importer functions propagate errors from third party libraries to LabVIEW. The error description shows the actual error code and explanation received from the third party library.

Related information:

- [Pretrained Models and Algorithms](#)

Classification

This section contains information about classification.

Introduction 36

Classification identifies an unknown sample by comparing a set of its significant features to a set of features that conceptually represent classes of known samples. A particle classifier uses feature vectors to identify samples based on their shape. A color classifier uses color features to identify samples based on their color.

Classification involves two phases: training and classifying. Training is a phase during which you teach the machine vision software the types of samples you want to classify during the classifying phase. You can train any number of samples to create a set of classes, which you later compare to unknown samples during the classifying phase. You store the classes in a classifier file. Training might be a one-time process, or it might be an incremental process you repeat to add new samples to existing classes or to create several classes, thus broadening the scope of samples you want to classify.

Classifying is a phase during which your custom machine vision application classifies an unknown sample in an inspection image into one of the classes you trained. The classifying phase classifies a sample according to how similar the sample features are to the same features of the trained samples.

When to Use

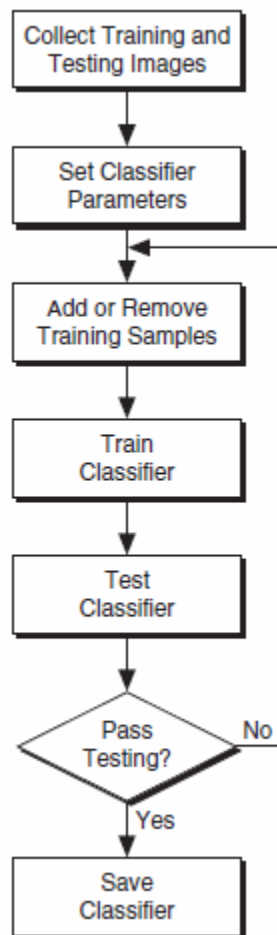
The need to classify is common in many machine vision applications. Typical applications involving classification include the following:

- **Sorting**—Sorts samples of varied shapes or colors. For example, a classifier can sort different items on a conveyor belt into different bins. A particle classifier can sort mechanical parts of different shapes, and a color classifier can sort items of different colors. Example outputs of a sorting or identification application could be user-defined labels of certain classes.
- **Inspection**—Inspects samples by assigning each sample an identification score

and then rejecting samples that do not closely match members of the training set. Example outputs of a sample inspection application could be Pass or Fail.

Training the Classifier

The following figure illustrates the process of training and testing a classifier.



Based on your specific application, predefine and label a set of training samples that represent the properties of the entire population of samples you want to classify. Configure the classifier by selecting the proper **classification method** and **distance metric** for your application. For example, you can configure the Particle Classifier to distinguish the following:

- Small differences between sample shapes independent of scale, rotation, and mirror symmetry,
- Shapes that differ only by scale,

- Shapes that differ only by mirror symmetry,
- Any combination of the above points.

If testing indicates that the classifier is not performing as expected, you can restart the training process by collecting better representative samples or trying different training settings. In some machine vision applications, new parts or colors need to be added to an existing classification system. This can be done by incrementally adding samples of the new parts or colors to the existing classifier.

Related concepts:

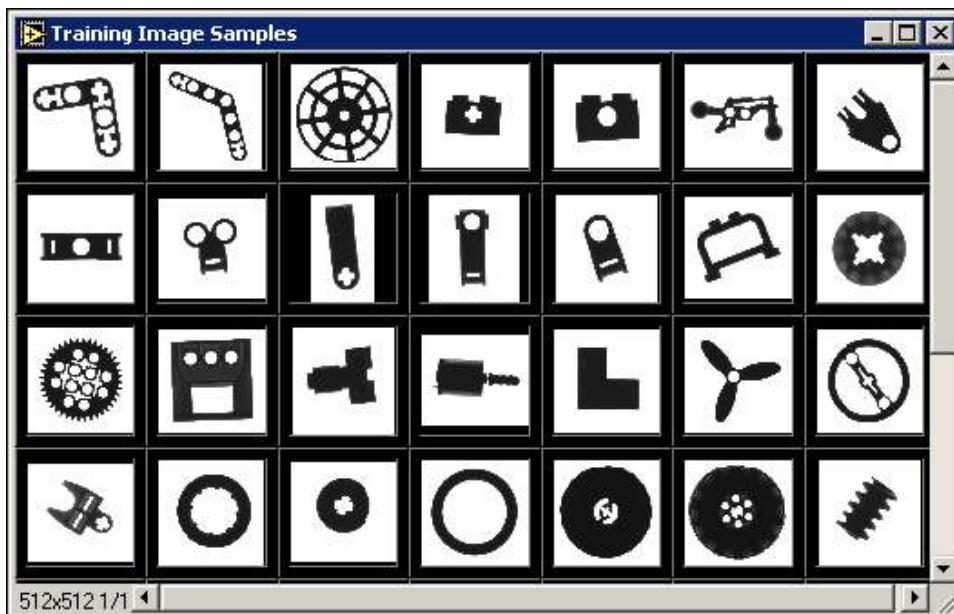
- [Nearest Neighbor](#)

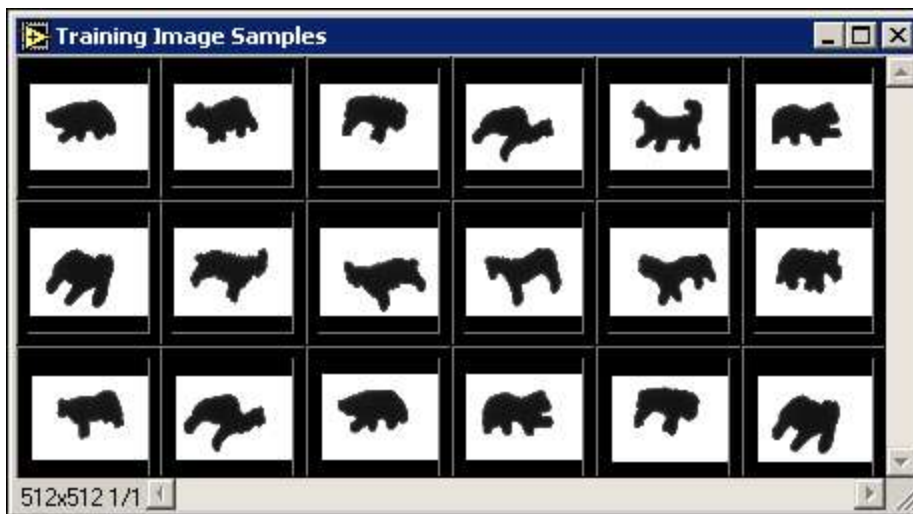
Binary Particle Classification

Use binary particle classification to identify samples based on their shape.

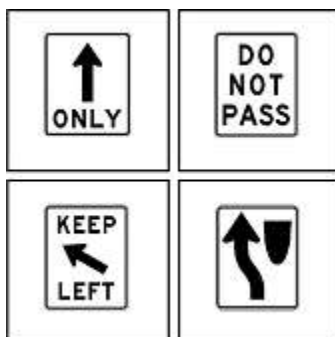
Ideal Images for Classification

Images of samples acquired in a backlit environment are ideal for particle classification. The following figures show examples images of backlit samples.





The following figures show samples that are not ideal for particle classification because they contain several unconnected parts or are grayscale and have an internal pattern.



General Classification Procedure

Consider an example application whose purpose is to sort nuts and bolts. The classes

in this example are Nut and Bolt.

Before you can train a classification application, you must determine a set of features, known as a feature vector, on which to base the comparison of the unknown sample to the classes of known samples. Features in the feature vector must uniquely describe the classes of known samples. An appropriate feature vector for the example application would be {Heywood Circularity, Elongation Factor}.

The following table shows good feature values for the nuts and bolts shown in the subsequent figure. The closer the shape of a sample is to a circle, the closer its Heywood circularity factor is to 1. The more elongated the shape of a sample, the higher its elongation factor.

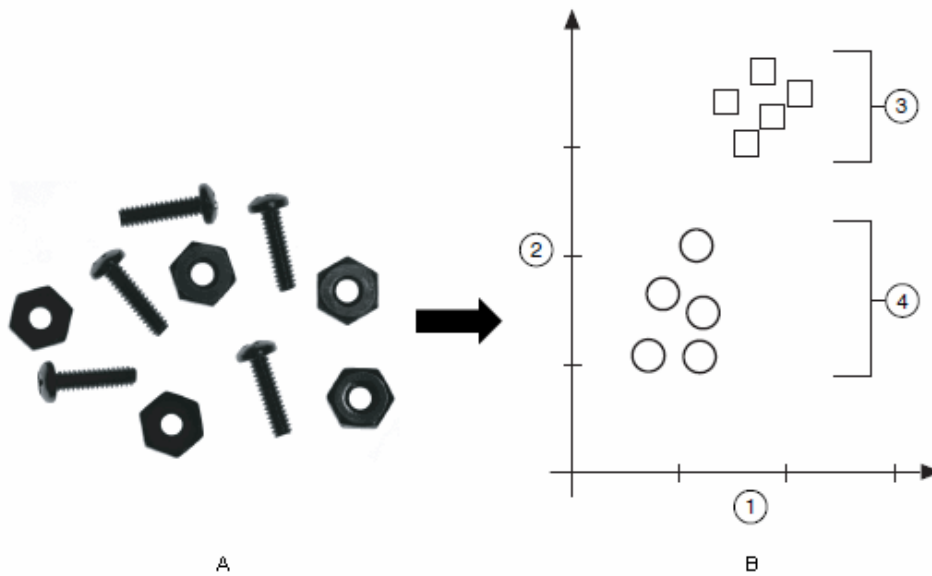
Class	Average HeywoodCircularity	Average ElongationFactor
Nut	1.109	1.505
Bolt	1.914	3.380

The class Nut is characterized by a strong circularity feature and a weak elongation feature. The class Bolt is characterized by a weak circularity feature and a strong elongation feature.

After you determine a feature vector, gather examples of the samples you want to classify. A robust classification system contains many example samples for each class. All the samples belonging to a class should have similar feature vector values to prevent mismatches.

After you have gathered the samples, train the classifier by computing the feature vector values for all of the samples. Then you can begin to classify samples by calculating the same feature vector for the unknown sample and comparing those values to the feature vector values of the known samples. The classifier assigns the unknown sample a class name based on how similar its feature values are to the values of a known sample.

Illustration A shows a binary image of nuts and bolts. Illustration B shows these samples classified by circularity and elongation.



1. Circularity
2. Elongation
3. Bolts
4. Nuts

Preprocessing

Preprocessing operations prepare images for better feature extraction. Preprocessing includes noise filtering; thresholding; rejecting particles that touch the image border; and removing small, insignificant particles.

For best results, acquire the inspection images under the same lighting conditions in which you acquired the training images. Also, apply the same preprocessing options to the inspection images that you used to preprocess the training images.

Feature Extraction

Feature extraction computes the feature vector in the feature space from an input image. Feature extraction reduces the input image data by measuring certain features or properties that distinguish images of different classes. Which features to use depends on the goal of the classification system. The features could be raw pixel values or some abstract representation of the image data. For identification applications, select features that most efficiently preserve class separability—feature values for one class should be significantly different from the values for another class.

For inspection applications, select features that distinguish the acceptable from the defective.

The Particle Classifier classifies samples using different types of shape descriptors. A shape descriptor is a feature vector based on particle analysis measurements. Each type of shape descriptor contains one or more shape measurements made from a sample.

The default Particle Classifier shape descriptor is based on shape characteristics that are invariant to scale changes, rotation, and mirror symmetry. Another type of shape descriptor is based on the size of the sample and is used along with the default shape descriptor to distinguish samples with the same shape but different scale, such as different sized coins. The Particle Classifier also uses a reflection-dependent shape descriptor to distinguish samples that are the same shape but exhibit mirror symmetry, such as a lowercase letter p and a lowercase letter q. The Particle Classifier uses these different types of shape descriptors in a multi-classifier system to achieve scale-dependent classification, reflection-dependent classification, or scale and reflection-dependent classification.

Invariant Features

The Particle Classifier uses the following features for scale-invariant, rotation-invariant, and reflection-invariant shape descriptors:

- Feature 1 describes the circularity of the sample.
- Feature 2 describes the degree of elongation of the sample.
- Feature 3 represents the convexity of the sample shape.
- Feature 4 is a more detailed description of the convexity of a sample shape.
- Feature 5 is used for the discrimination of samples with holes.
- Feature 6 is used for more detailed discrimination of samples with holes.
- Feature 7 represents the spread of the sample.
- Feature 8 represents the slenderness of the sample.

Classification

The Particle Classifier can apply the following **classification algorithms**: Minimum Mean Distance, Nearest Neighbor, and K-Nearest Neighbor. Each of these methods may employ different **distance metrics**: Maximum distance (L^∞), Sum distance (L_1), and

Euclidean distance (L2).

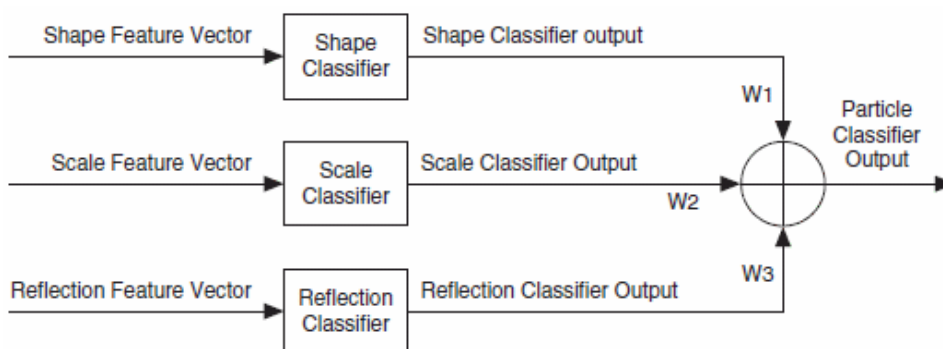
Cascaded Classification System

In a cascaded classification system, cascaded multiple classifiers make classification decisions based on multiple classification stages. Classifier 1 outputs several candidates for Classifier 2 in the second stage. Classification is based on different features.

Parallel Classification Systems

Combining results from multiple classifiers may generate more accurate classification results than any of the constituent classifiers alone. Combining results is often based on fixed combination rules, such as the product and/or average of the classifier outputs.

The Particle Classifier uses a parallel classification system with three classifiers, as illustrated in the following figure. Two classifiers are used for scale-dependent classification. One of these classifiers uses scale-invariant features, and the other uses a scale-dependant feature. Additionally, the Particle Classifier uses a third classifier to distinguish samples with mirror symmetry. The outputs of the classifiers are combined using user-specified weights to get the result.



Related concepts:

- [Nearest Neighbor](#)

Color Classification

Use color classification to identify samples based on their color.

Concepts

A color classifier has a training phase and a classifying phase. In the training phase, you provide the classifier with known samples. A known sample consists of a region in the image containing the color you want the classifier to learn and a label for the color. For every sample that is added during the training phase, the color classifier calculates a color feature and assigns the associated class label to the feature. Eventually, all the trained samples (color feature with the label) added to the classifier are saved into a file which represents a trained color classifier.

After you train the classifier, you can classify regions in an image into their corresponding classes for color identification and color inspection type machine vision applications. In the classifying phase, the classification engine calculates the color feature of the sample that you want to identify and classifies them among trained sample using one of the existing classification algorithms. Vision color classification uses the same classification algorithms as the particle classifier including the Minimum Mean Distance, Nearest Neighbor, and K-Nearest Neighbor classifiers.

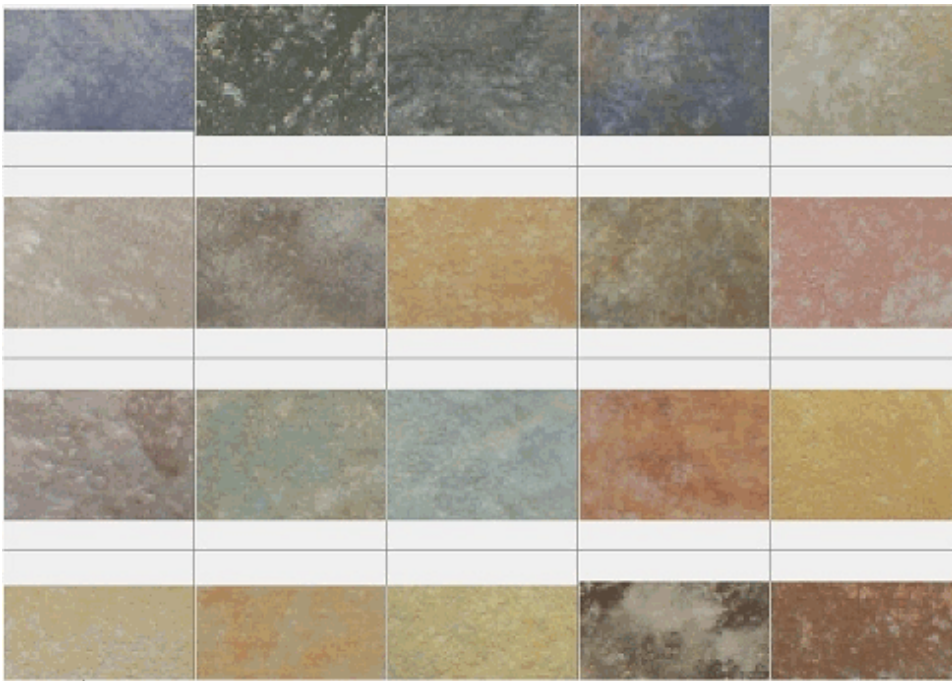
Sample Images

The following figure shows samples that are not ideal for color classification because they include a large amount of background color.



Samples with only one hue (samples of a pure color) are also not ideal. Samples should include enough variation to capture any close change in hue.

The following figure shows a set of images that are ideal for color classification. Each image has a textured color pattern that illustrates the range of colors for each class.



Preprocessing

There are no preprocessing algorithms associated with the NI Color Classifier. For example, if you supply a color sample that includes background regions, then the background color is included in the calculated color feature. You must use separate preprocessing algorithms to separate the background region from the color sample before you add the color sample to the NI Color Classifier.



Note Because the Color Classifier supports all closed ROI types, you can use any closed shape of a color region as a sample.

Feature Extraction

The Color Classifier uses the HSL color space to calculate a color feature for every sample to be trained or classified. The color feature represents the three dimensional color information of the sample in a one dimensional format. The Color Classifier calculates the color feature according to the following steps:

1. Convert the color sample to the HSL color space.
2. Calculate the hue, saturation, and luminance histograms of the color sample. The hue and saturation histograms each contain 256 values.
3. Reduce the luminance histogram to 8 values which are suppressed by 12.5%. By

suppressing the luminance histogram, the Color Classifier accentuates the color information for the sample.



Note Because it suppresses the luminance histogram, the NI Color Classifier cannot identify more than eight pure gray colors.










4. Combine the 520 hue, saturation, and luminance values to produce a high resolution color feature.
5. Obtain medium and low resolution color features by applying a dynamic mask to the high resolution color feature. The medium and low resolution color features are subsets of the high resolution color feature. The medium resolution color feature contains 128 hue and saturation values and 8 luminance values for a total of 136 values. The low resolution color feature contains 64 hue and saturation values and 8 luminance values for a total of 72 values.

You can select a high resolution, medium resolution, or low resolution color feature. Select the medium or low resolution color features to speed up color classification.

The dynamic mask that is applied to the high resolution color feature to produce the medium and low color features selects the hue and saturation values that most distinctly differentiate the color class from other color classes. The dynamic mask varies based on the trained color samples. The Color Classifier calculates the dynamic mask according to the following steps:

1. Calculate the mean hue and saturation histograms For each class label based on trained samples.
2. Calculate the standard deviation on the mean histogram values across all the class labels.
3. Identify the 128 locations in the mean histograms with the highest standard deviation values to produce the medium resolution color feature.
4. Identify the 64 locations in the mean histograms with the highest standard deviation values to produce the low resolution color feature.
5. Ensure that the dynamic mask contains at least one significant hue value and one significant saturation value for the class.

The following figure illustrates example values of medium resolution and low resolution masks for an example class.

	Hue	Saturation	Luminance
High			
	512 of 512 locations enabled	512 of 512 locations enabled	8 locations
Medium			
	128 of 512 locations enabled	128 of 512 locations enabled	8 locations
Low			
	64 of 512 locations enabled	64 of 512 locations enabled	8 locations

The Color Classifier stores the high resolution color features for each sample in the classifier file. If you select a medium or low resolution color feature, the Color Classifier stores the dynamic mask for the medium or low resolution feature with the classifier file.

Classification

During classification, the Color Classifier calculates the high resolution color feature for each class, then applies any medium or low resolution mask stored in the classifier file to produce the final color feature. The mask is applied to trained samples and the color sample to be classified.

The general concepts of the ***general classification procedure*** for binary particle classification also apply to color classification.

Related concepts:

- [Binary Particle Classification](#)

Nearest Neighbor

Nearest neighbor classification includes Nearest Neighbor, K-Nearest Neighbor, and Minimum Mean Distance algorithms. The most intuitive way of determining the class of a feature vector is to find its proximity to a class or features of a class using a distance function. Based on the definition of the proximity, there are several different algorithms, as follows.

Distance Metrics

The Particle Classifier and the Color Classifier provide three distance metrics: Euclidean distance, Sum distance, and Maximum distance.

Let $X = [x_1, x_2, \dots, x_n]$ and $Y = [y_1, y_2, \dots, y_n]$ be the feature vectors.

Euclidean distance (L2)	$d(X, Y) = \sqrt{\sum_{i=1}^n (X_i - Y_i)^2}$
Sum distance, also known as the City-Block metric or Manhattan metric (L1)	$d(X, Y) = \sum_{i=1}^n (X_i - Y_i)$
Maximum distance (L^∞)	$d(X, Y) = \max_i (X_i - Y_i)$

Nearest Neighbor Classifier

In Nearest Neighbor classification, the distance of an input feature vector X of unknown class to a class C_j is defined as the distance to the closest sample that is used to represent the class.

$$d(X, C_j) = \min_i d(X, X_i^j)$$

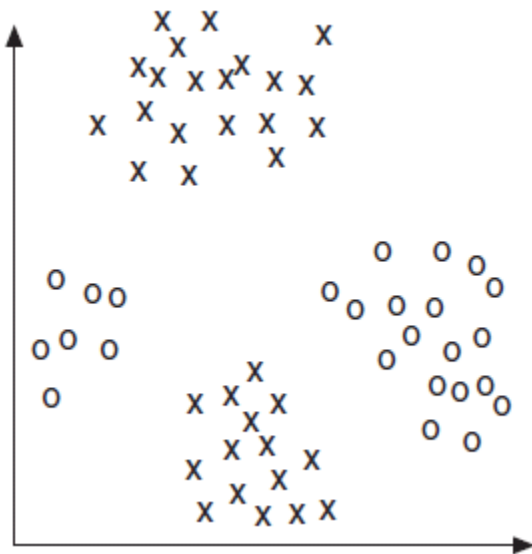
where $d(X, X_i^j)$ is the distance between X and X_i^j .

The classification rule assigns a pattern X of unknown classification to the class of its nearest neighbor.

$$X \in \text{Class } C_j, \text{ if } d(X, C_j) = \min_i d(X, C_i)$$

Nearest neighbor classification is the most intuitive approach for classification. If representative feature vectors for each class are available, Nearest Neighbor classification works well in most classification applications.

In some classification applications, a class may be represented by multiple samples that are not in the same cluster, as shown in the following figure. In such applications, the Nearest Neighbor classifier is more effective than the Minimum Mean Distance classifier.



o = Class 1

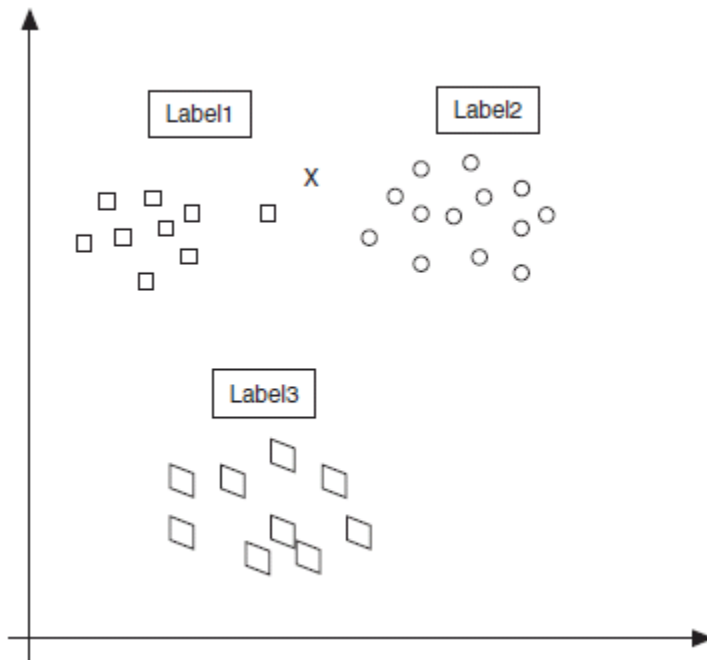
x = Class 2

K-Nearest Neighbor Classifier

In K-Nearest Neighbor classification, an input feature vector X is classified into class C_j based on a voting mechanism. The classifier finds the K nearest samples from all of the classes. The input feature vector of the unknown class is assigned to the class with the majority of the votes in the K nearest samples.

The outlier feature patterns caused by noise in real-world applications can cause erroneous classifications when Nearest Neighbor classification is used. As the

following figure illustrates, K-Nearest Neighbor classification is more robust to noise compared with Nearest Neighbor classification. With X as an input, K = 1 outputs Label 1, and K = 3 outputs Label 2.



Minimum Mean Distance Classifier

Let $\{X_1^j, X_2^j, \dots, X_{n_j}^j\}$ be n_j feature vectors that represent class C_j . Each feature vector has the label of class j that you have selected to represent the class. The center of the class j is defined as

$$M_j = \frac{1}{n_j} \sum_{i=1}^{n_j} X_i^j$$

The classification phase classifies an input feature vector X of unknown class based on its distance to each class center.

$$X \in \text{Class } C_j, \quad \text{if } d(X, M_j) = \min_i d(X, M_i)$$

where $d(X, M_j)$ is defined as the distance function based on the distance metric selected during the training phase.

In applications that have little to no feature pattern variability or a lot of noise, the feature patterns of each class tend to cluster tightly around the class center. Under these conditions, Minimum Mean Distance classifiers perform effectively—only the input vector distances to the centers of the classes need to be calculated instead of all the representative samples in real-time classification.

Support Vector Machines

A Support Vector Machine (SVM) is a supervised learning method that generalizes a large set of trained samples into a smaller number of support vectors to predict the class of unknown samples.

A SVM classifier is mathematically more complex than a distance-based classifier. However a SVM classifier has better generalization capabilities than a distance-based classifier, and is faster when the sample set is large because the SVM classifier operates only on the support vectors.

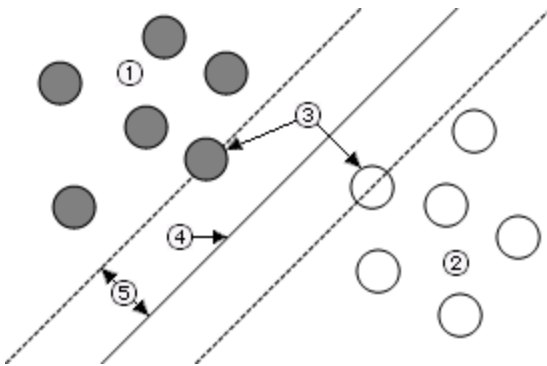
When to use

Use a SVM classifier in the following types of applications:

- The application has one class of good samples but an unknown number of classes for bad samples. An example of this type of application is defect detection. For this type of application, use a one-class SVM classifier to train samples of the known good class. Samples, such as defects, that cannot be classified as the known class are classified as unknown.
- The application requires a large number of training samples. During training, the SVM classifier identifies support vectors for the training samples. During classification the SVM classifier operates only on the support vectors, which reduces the time required for classification.

In-Depth Discussion

The SVM algorithm builds a model to classify samples. The model represents the samples in a multi-dimensional space where the samples are separated by the maximum possible distance. For example, the following figure illustrates an application that involves linearly-separable two classes represented in a two-dimensional space.



1. Samples of Class A
2. Samples of Class B
3. Support Vectors
4. Hyperplane
5. Margin

The SVM algorithm uses a quadratic function to identify the support vectors for each class. A support vector is a sample in one class that is closest to another class. The SVM algorithm then identifies a hyperplane that separates the support vectors of each class. The distance between the support vector and the hyperplane is called the margin. The SVM algorithm selects a support surface that produces the largest possible margin for each support vector.

Training

When you train the SVM classifier, the SVM algorithm uses an iterative process to optimize the support vector function. You can control the optimization by using the tolerance parameter in the software. Training is terminated when the gradient of the optimized function is less than or equal to tolerance. A tolerance value that is too high may cause the SVM algorithm to terminate training before the support vector function is adequately optimized. A tolerance value that is too low will cause the SVM algorithm to try to achieve a very high level of optimization, which may be too time-consuming and computationally expensive.

Classification

When you use the SVM classifier, the SVM algorithm determines the class of an unknown sample by comparing it with the support vectors of the trained samples. The SVM algorithm uses the following formula to classify an unknown sample x :

$$\text{sgn}\left(\sum y_i a_i K(x_i, x) + b\right)$$

where:

- y_i is the class association (-1 or $+1$),
- a_i is the weight coefficient,
- K is the kernel function x_i is the number of support vectors,
- b is the distance of the hyperplane from origin.

Classification speed depends on the number of support vectors and the selected kernel function. The weight coefficient a_i , which is an output of the optimized support vector function, determines the number of support vectors. If the weight coefficient of a sample is not equal to 0, the sample is a support vector.

Multi-Class SVM

SVM classification typically involves two classes. For applications that involve more than two classes, the SVM algorithm uses a one-versus-one approach. In a one-versus-one approach, the algorithm creates a binary classification model for every possible combination of classes, so that n number of classes produces $n \times (n - 1)/2$ classification models. During classification, the algorithm uses a voting mechanism to identify the best class. If the voting mechanism identifies multiple classes, the algorithm selects the class that is closest to the sample.

Models

The following sections describe the models that the SVM algorithm uses to classify samples. Select a model based on the classes involved in your application. For applications that involve a single class, such as texture defect detection, select the one-class model. For applications that involve multiple classes, select the C-SVC or nu-SVC models. For applications that involve multiple classes, always start with the nu-SVC model.

C-SVC

The C-SVC model allows the SVM algorithm to clearly separate samples that are separated by a very narrow margin. Training involves minimizing the error function:

$$\min_{w, b, \xi} \frac{1}{2} W^T W + C \sum_{i=1}^l \xi_i$$

Subject to $Y_i(W^T K(X_i) + b) \geq 1 - \xi_i; \xi_i \geq 0, i = 1 \dots l$

where:

- W is the normal vector of the hyperplane to origin,
- C is the cost parameter,
- ξ is the slack variable.

If the SVM algorithm cannot define a clear margin, then it uses the cost parameter to allow some training errors and produce a soft margin. If the cost value is too high it prohibits training errors, producing a narrow margin and rigid classification.

Nu-SVC

In the Nu-SVC model, the nu parameter controls training errors and the number of support vectors. Training involves minimizing the error function:

$$\min_{w, b, \xi} \frac{1}{2} W^T W - \nu \rho + \frac{1}{l} \sum_{i=1}^l \xi_i$$

Subject to $Y_i(W^T K(X_i) + b) \geq \rho - \xi_i; \xi_i \geq 0, i = 1 \dots l; \rho \geq 0$

where:

- W is the normal vector of the hyperplane to origin,
- ν is the nu parameter,
- ξ is the slack variable.

The nu value specifies both the maximum ratio of training errors and the minimum number of support vectors relative to the number of samples. Nu must be greater than 0 and cannot exceed 1. A higher nu value increases tolerance for variation in the texture, but may also increase tolerance for texture defects. If nu is too high, training produces too many training errors to be useful.

One-Class SVM

In the one-class model, the SVM algorithm considers the spatial distribution information for each sample to determine whether the sample belongs to the known class. Training involves minimizing the error function:

$$\min_{w, b, \xi} \frac{1}{2} W^T W - \rho + \frac{1}{\nu l} \sum_{i=1}^l \xi_i$$

Subject to $W^T K(X_i) \geq \rho - \xi_i$; $\xi_i \geq 0$, $i = 1 \dots l$; $\rho \geq 0$

where:

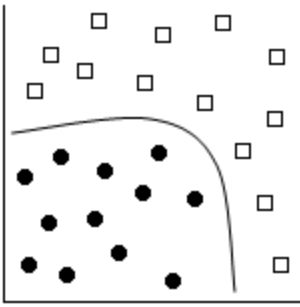
- W is the normal vector of the hyperplane to origin,
- ν is the nu parameter,
- ξ is the slack variable.

Kernels

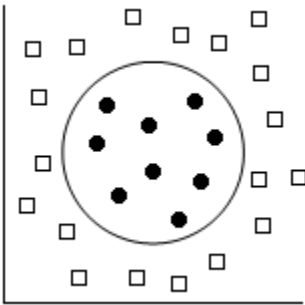
A SVM classifier is a linear classifier. Typically, a SVM classifier uses a linear kernel, which is the product of the sample feature vector multiplied by the sample support vector. A SVM classifier can also use the following nonlinear kernels.

Polynomial	$\left(\text{Gamma} \times \text{Kernel}(x_i, x) + \text{Coefficient} \right)^{\text{Degree}}$
Radial BasisFunction (RBF)	$e^{-\text{Gamma}(x_i + x)^2}$
Gaussian	$\frac{(x_i - x)^2}{e^{2 \times \text{Sigma}^2}}$

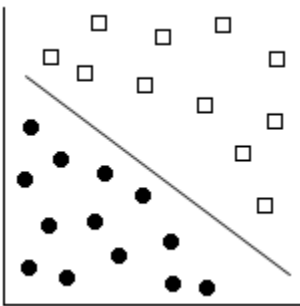
Use a nonlinear kernel to transform samples with nonlinear feature information to a dimension where the feature information is linearly separable, as illustrated in the following figures.



A



B



C

Figure A illustrates how a polynomial kernel separates nonlinear feature information. Figure B illustrates how a RBF kernel separates nonlinear feature information. Figure C illustrates the clearly devisable nonlinear feature information obtained after using a nonlinear kernel to transform the sample to a dimension where the feature information is linearly separable.

Choosing the Right Parameters

The following list provides information for selecting the right SVM parameters for your

application.

- **Model**—If your application involves only one class, use the one-class model. If your application involves more than one class, always start with the nu-SVC model.
- **Tolerance**—Specifies the maximum gradient of the quadratic function used to compute support vectors. Training is terminated when the gradient of the optimized function is less than or equal to the tolerance value. The default value is 0.001. You typically do not need to change this value.
- **nu**—Specifies both the maximum ratio of training errors and the minimum number of support vectors relative to the number of samples. Values must be greater than 0 and cannot exceed 1. The default value is 0.1. A higher nu value increases tolerance for variation in the texture, but may also increase tolerance for texture defects. If the texture classifier does not perform as expected because the trained texture samples do not represent every possible variation of the texture, try increasing the value of nu.
- **Cost**—Specifies the penalty for training errors. If the cost value is too high it prohibits training errors, producing a narrow margin and rigid classification. Decrease the cost value to allow more training errors and produce a softer margin between classes.
- **Kernel**—Specifies the kernel that the classifier uses. RBF is the default value. In general, you do not need to modify this setting. If the number of sample features is high, try the linear kernel.
- **Degree**—Specifies the degree of the polynomial kernel. In general, select a value less than 10.
- **Gamma**—Specifies the gamma value for the polynomial and RBF kernels. A high value requires more support vectors to classify the sample. Use a high value for samples with regularly distributed feature information, and a low value for samples with irregularly distributed feature information. You may need to change this value to support the values selected for **Cost** or **nu**. For example, if you specify a high nu value, which raises the minimum number of support vectors, you may also need to increase the value of **Gamma**.

If you use a custom classifier, specify a feature vector value for the custom classifier that is greater than 0 but less than 1. Scaling the feature vector reduces overflow issues and improves the classification rate.

Custom Classification

You can define a custom feature extraction process for specific machine vision applications using Vision.

When to Use

Typical applications include sorting and inspection applications for which you can define a feature descriptor to represent the different classes in a specific application. Examples of such feature descriptors include statistics about the grayscale pixel distribution in an image, measurements from a Vision gauging tool, or color spectra from Vision color learning algorithms.

Concepts

With custom classification, you create a classifier by training it with prelabeled training feature vectors. NI Vision custom classification uses the same classification algorithms as the Particle Classifier, including the Minimum Mean Distance, Nearest Neighbor, and K-Nearest Neighbor classifications.

In-Depth Discussion

This section provides additional information you may need for making a successful classification application.

Training Feature Data Evaluation

A good training data set should have both small intraclass variation and large interclass variation. The NI Particle Classifier outputs an intraclass deviation array to represent the deviation in each class, and a class distance table to represent the deviation between the classes.

Intraclass Deviation Array

$[Q_j, N_j]$, $j=1,2, \dots L$, where N_j is the number of samples in class j and L is the number of classes. The number of samples N_j represents the statistical significance of Q_j that is defined as follows:

Let $\{X_1^j, X_2^j, \dots, X_{N_j}^j\}$ be N_j n -dimensional feature vectors that represent class C_j with $X_i^j = [x_{i1}^j, x_{i2}^j, \dots, x_{in}^j]^T$. Each feature vector has the label of class j that you have selected to represent the class. Let $M_j = [m_1^j, m_2^j, \dots, m_n^j]^T$ be the mean vector of the class j . Then

$$M^j = \frac{1}{n_j} \sum_{i=1}^{n_j} X_i^j$$

where each element of the mean vector:

$$M_k^j = \frac{1}{n_j} \sum_{i=1}^{n_j} X_{ik}^j$$

The standard deviation of feature element k of class j is defined as:

$$\sigma_k^j = \frac{1}{n_j} \sum_{i=1}^{n_j} (X_{ik}^j - m_k^j)^2$$

The quality of feature data in class j is defined as:

$$Q_j = \max_k \sigma_k^j$$

A small Q_j indicates that the training data in class j is tightly clustered about the class center. A large Q_j indicates that the training data is spread out from the class center, which may increase chances for misclassification.

Class Distance Table

Let $M_j = [m_1^j, m_2^j, \dots, m_n^j]^T$ be the mean vector of the class j as defined before. The distance between two classes i and j is defined as follows:

$$d_{ij} = D(M^i, M^j)$$

where D is the distance metric selected from the training option.

You can use the class distance table to examine statistical information, such as the two closest class distances and the two most widely separated classes. Additionally, you can use the class distance table with the intraclass deviation array to evaluate the quality of different training data sets.

Determining the Quality of a Trained Classifier

The Particle Classifier outputs a classification distribution table that you can use to determine the quality of a trained classifier. the Example Classification Distribution Table table shows an example classification distribution table.

Example Classification Distribution Table

	C1	C2	C3	Total	Accuracy
Samples of Class C1	10	0	0	10	$10 / 10 = 100\%$
Samples of Class C2	0	8	2	10	$8 / 10 = 80\%$
Samples of Class C3	4	0	6	10	$6 / 10 = 60\%$
Total	14	8	8	30	$24 / 30 = 80\%$
Predictive Value	$10 / 14 = 71\%$	$8 / 8 = 100\%$	$6 / 8 = 75\%$	-	-

In this example, assume that the classifier was given 30 samples to classify: 10 samples known to be in class C1, 10 samples known to be in class C2, and 10 samples known to be in class C3.

Classifier Predictability

The classification predictive value indicates the probability that a sample classified into a given class belongs to that class. Use the columns of the table to determine the predictive value, per class, of the classifier. Each column represents a class into which the classifier classifies samples. The values in the columns indicate how many samples of each class have been classified into the class represented by the column. For example, 10 samples known to be in class C1 were correctly classified into class C1. However, 4 samples known to be in class C3 were also classified into C1.



Note The number of samples classified correctly into a class is located at the

intersection of row **Samples of Class x** and column **Cx**.

Looking down a column, notice the number of samples that were classified correctly into the class. Count the total number of samples classified into the class. The predictive value of the class is the ratio of:

$$\frac{\text{Number of Samples Classified Correctly}}{\text{Total Number of Samples Classified into the Class}}$$

For example, the predictive value of class C1 is 71%.

$$\frac{10}{10 + 4} = 0.71 = 71 \%$$

Classifier Accuracy

The classification accuracy indicates the probability that a sample is classified into the class to which it belongs. Use the rows of the table to determine the accuracy, per class, of the trained classifier. The accuracy indicates the probability that the classifier classifies a sample into the correct class. Each row shows how the classifier classified all of the samples known to be in a certain class. In the example classification distribution table, 8 of the samples known to be in class C2 were correctly classified into class C2, but 2 of the samples known to be in class C2 were erroneously classified into class C3.

Looking across a row, the accuracy of a class is the ratio of:

$$\frac{\text{Number of Samples Classified Correctly}}{\text{Total Number of Samples Known to Be the Class}}$$

For example, the accuracy of class C1 is 100%.

$$\frac{10}{10} = 1 = 100 \%$$

Identification and Classification Score

The Particle Classifier outputs both identification confidence and classification confidence for the evaluation of classification results. The classification confidence outputs a meaningful score for both sorting and inspection applications. Use the identification confidence only when you cannot reach a decision about the class of a

sample by using the classification confidence score alone.

Classification Confidence

The classification confidence indicates the degree to which the assigned class represents the input better than the other classes represent the input. It is defined as follows:

$$\text{Classification Confidence} = (1 - d_1 - d_2) \times 1000$$

where d_1 is the distance to the closest class, and d_2 is the distance to the second closest class.

The distance is dependent on the classification algorithm used. Because $0 \leq d_1 \leq 1$ and $0 \leq d_2 \leq 1$, the classification confidence is a score between 0 and 1000.

Identification Confidence

The identification confidence indicates the similarity between the input and the assigned class. It is defined as follows:

$$\text{Identification Confidence} = (1 - d) \times 1000$$

where d is the normalized distance between the input vector and the assigned class.

Distance d is dependent on the classification algorithm used.

$$d = \frac{\text{Distance Between Input Sample and its Assigned Class}}{\text{Normalization Factor}}$$

The normalization factor is defined as the maximum interclass distance.

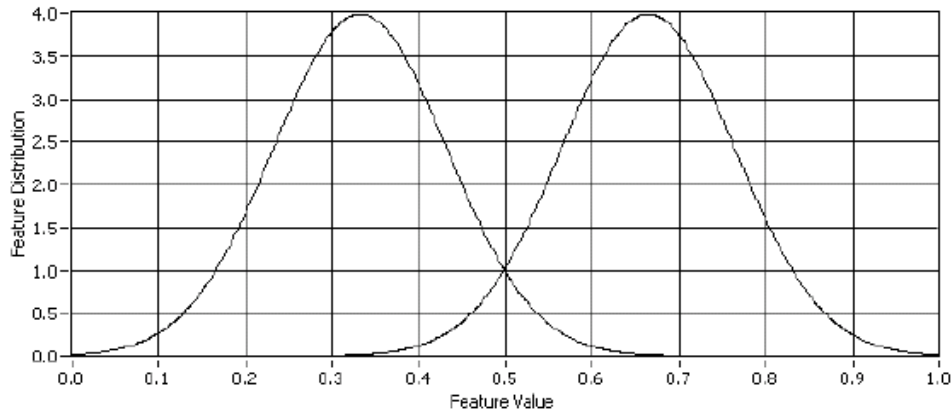
Calculating Example Classification and Identification Confidences

Assume a normalized scalar feature with a distribution in $[0,1]$ from two classes of patterns, as shown in the following figure. The centers of the two classes are 0.33 and 0.67, respectively. If the Minimum Mean Distance is used for classification with input feature $x = 0.6$, the classification output is class 2, and the classification confidence is calculated as:

$$\text{Classification confidence} = \left(1 - \frac{|0.60 - 0.67|}{|0.60 - 0.33|} \right) \times 1000 = 740$$

and the identification confidence is calculated as:

$$\text{Identification confidence} = \left(1 - \frac{|0.60 - 0.67|}{|0.67 - 0.33|} \right) \times 1000 = 794$$

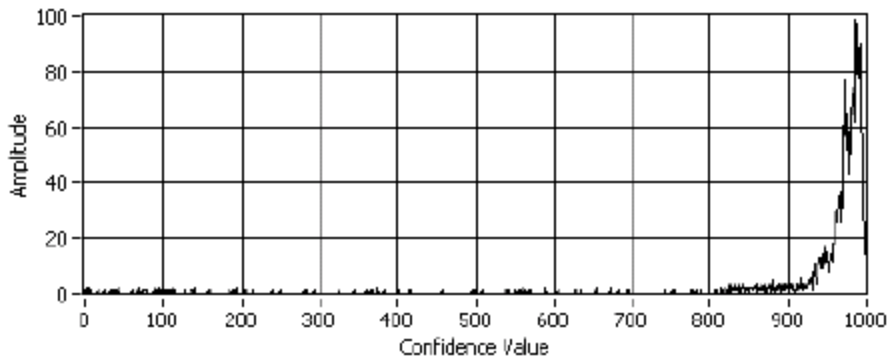


For a feature value $x = 0.5$, the sample can be classified into class 1 or class 2 with the classification confidence value equal to 0. For $0.4 < x < 0.5$, the sample is classified into class 1 with low classification confidence, while $0.5 < x < 0.6$ is classified into class 2 with low classification confidence in a Minimum Mean Distance classification system.

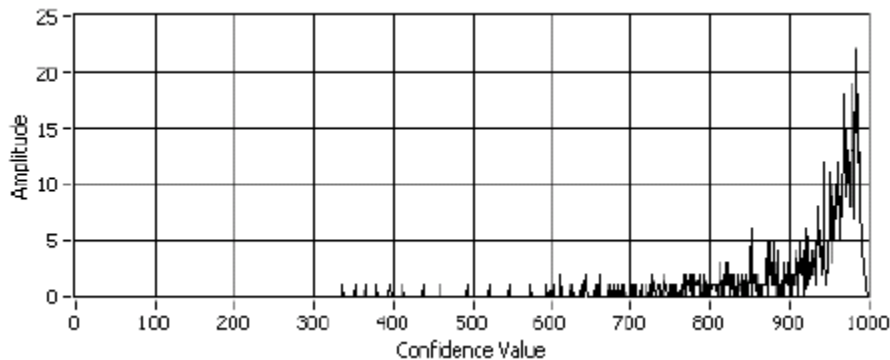
Evaluating Classifier Performance

For a systematic approach to evaluating a classifier in the design phase, define a testing data set in addition to a training data set. After you train the classifier using the training data set, run the classifier using the testing data set. The output of the classification confidence distribution is a good indicator of the classifier performance. The classification confidence distribution is a histogram of the classification score. The amplitude is the number of testing samples in a specific classification score.

the following figure shows the classification confidence distribution from a testing database of mechanical parts. You can set a minimum classification score of 800 and get a high classification rate for this testing database.



the following figure shows the classification confidence distribution from a testing database of animal crackers. If you use the same minimum classification score for cracker image classification that you used for mechanical parts classification, you get a high rate of false negatives because a large portion of the cracker classification scores are less than 800.



A classification confidence distribution from a representative testing database is a good indicator for selecting a good score threshold for a specific inspection or sorting application.



Note A score threshold that can be used to reject classification results is application dependent. Experiment with your classifier to determine an effective threshold for your application.

Cross-Validation

Use cross-validation to check the accuracy of the classifier. For cross-validation, trained samples are randomly divided into K groups, with the sample to class ratio roughly equal for each group. The classifier trains $K - 1$ groups, reserving a group to classify among the trained groups for validation. The cross-validation process is

repeated to validate each group. The accuracy of the classifier is calculated with the following formula.

$$\frac{\text{Number of Correctly Classified Samples}}{\text{Total Number of Samples}} \times 1000$$

Because the samples are randomly assigned to groups, the accuracy of the classifier may change each time you perform cross-validation even if you do not add samples or change settings. You can use this behavior to test the stability of the classifier. Minor variations indicate a stable texture classifier and large variations indicate an unstable texture classifier.

Defect Inspection

This section contains information about defect inspection.

When to Use Defect Maps

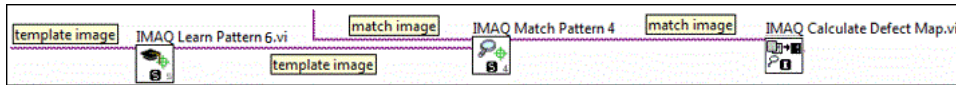
Defect maps aid in detecting defects in images and patterns when a template of the same is known. The Pattern Matching algorithm provides an overall score for a match (or matches). A more localized scoring mechanism has been developed which provides more information about how each pixel of the match differs from the template and hence is a great tool in detecting defects.

Defect Map Concepts

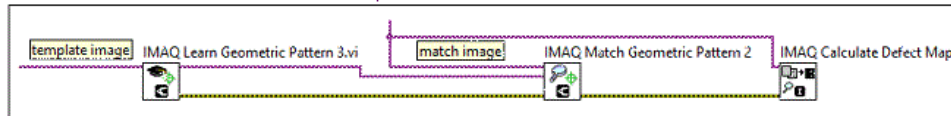
A defect map is a float-point image that has a score for every pixel. The higher the score, the higher the probability of the pixel being a defect. The score ranges from 0 to the square of the bit depth of the image. For example, an 8-bit image will have a score range from 0 to 65535.

Pattern Matching/Geometric Matching provides an overall score for a match (or matches). Pattern Matching/Geometric Matching stores the intermediate results of every match found in the match image. These values are used to calculate the defect map for every match.

Pattern Matching workflow:

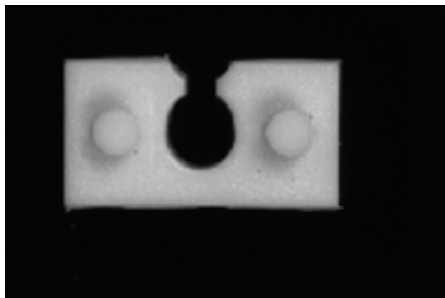


Geometric Matching workflow:

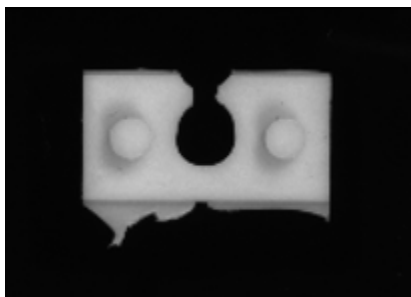


Weight Map

A weight map is the image used to specify weights to suppress noise and false defects in the defect map. Pixels with lower weights are enhanced and pixels with higher weights are suppressed in the defect map. The known patterns and noise in the template can be learned and captured as a weight map. The weight map will then be applied on the defect map to suppress this noise and enhance the real defect regions.



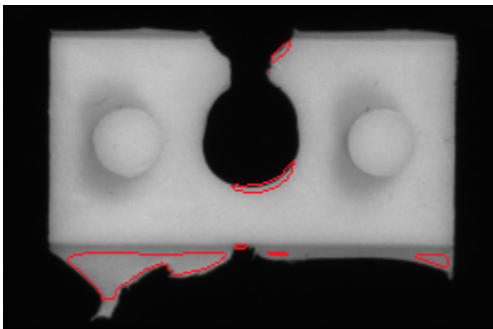
Template Image



Inspection Image



Defect Map



Defect Overlayed

Related concepts:

- [Introduction](#)
- [Introduction](#)

Golden Template Comparison

This section contains information about inspection based on golden template comparison.

Introduction

Golden template comparison compares the pixel intensities of an image under inspection to a golden template. A golden template is an image containing an ideal representation of an object under inspection. A pixel in an inspection image is returned as a defect if it does not match the corresponding pixel in the golden template within a specified tolerance.

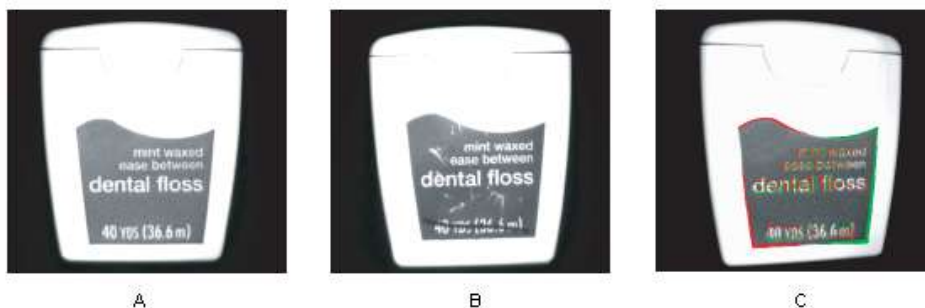
When to Use

Inspection based on golden template comparison is a common vision application. Use golden template comparison when you want to inspect for defects, and other methods of defect detection are not feasible. To use golden template comparison, you must be able to acquire an image that represents the ideal inspection image for your application.

Example applications in which golden template comparison would be effective include validating a printed label or a logo stamped on a part.

Concepts

Conceptually, inspection based on golden template comparison is simple: Subtract an image of an ideal part and another image of a part under inspection. Any visible defects on the inspected part show up as differences in intensity in the resulting defect image. The following figure illustrates this concept.



- Figure A shows the golden template in a label inspection application.
- Figure B shows the inspection image.
- Figure C shows the defect image.

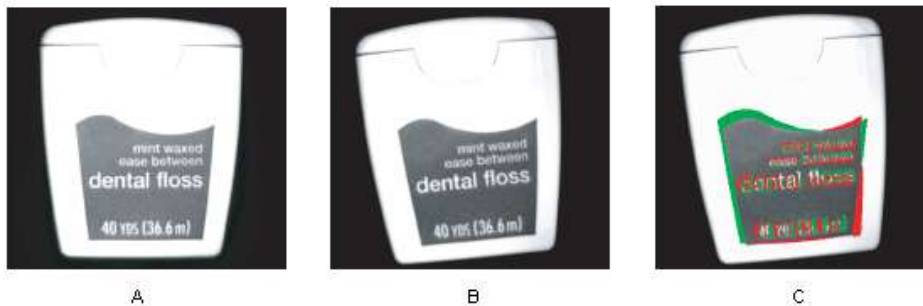
Defect areas in which the inspection image was brighter than the template are overlaid in green in the defect image. Defect areas in which the inspection image was darker than the template are overlaid in red.

Using simple subtraction to detect flaws does not take into account several factors about the application that may affect the comparison result. The following sections discuss these factors and explain how Vision compensates for them during golden

template comparison.

Alignment

In most applications, the location of the part in the golden template and the location of the part in the inspection image differ. The following figure illustrates this concept and shows how differing part locations affect inspection.



- Figure A shows the golden template.
- Figure B shows the inspection image.
- Figure C shows the defect image.

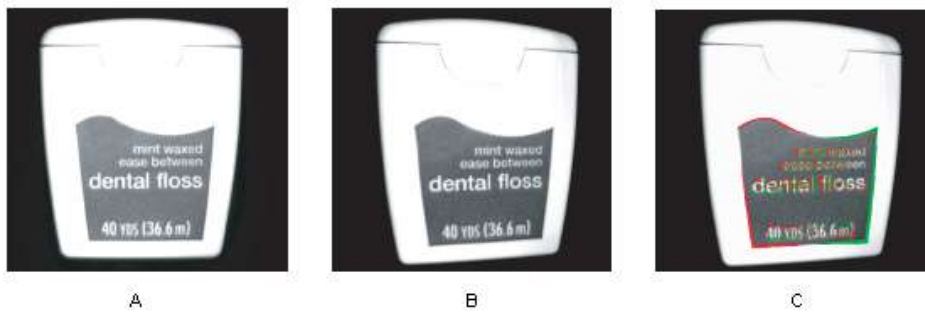
The label in the inspection image is identical to the label in the golden template. However, the part in the inspection image is located slightly higher and to the right compared to the part in the golden template. Due to this the top and right areas of the label are detected as dark defects compared to their corresponding pixels in the template, which are white background pixels. Similarly, the left and bottom appear as bright defects. The text and logo inside the label also appear as defects because of the part misalignment.

Aligning the part in the template with the part in the inspection image is necessary for an effective golden template comparison. To align the parts, you must specify a location, angle, and scale at which to superimpose the golden template on the inspection image. You can use the position, angle, and scale defined by other Vision functions, such as pattern matching, or geometric matching, or edge detection.

Perspective Correction

The part under inspection may appear at a different perspective in the inspection image than the perspective of the part in the golden template. The following figure

illustrates this concept and shows how differing image perspectives affect inspection.



- Figure A shows the golden template.
- Figure B shows the inspection image.
- Figure C shows the defect image.

The label in the inspection image is identical to the label in the golden template. However, the left side of the part in the inspection image is closer to the camera than the right side of the part, giving the part a warped perspective appearance. Although the angles and scales of the labels are the same, the template is still misaligned because of the perspective difference.

Golden template comparison corrects for perspective differences by correlating the template and inspection image at several points. Not only does this correlation compute a more accurate alignment, but it also can correct for errors of up to two pixels in the input alignment.

Histogram Matching

The inspection images may be acquired under different lighting conditions than the golden template. As a result the intensities between a pixel in the golden template and its corresponding pixel in an inspection image may vary significantly. The following figure illustrates this concept and shows how differing pixel intensities affect inspection.

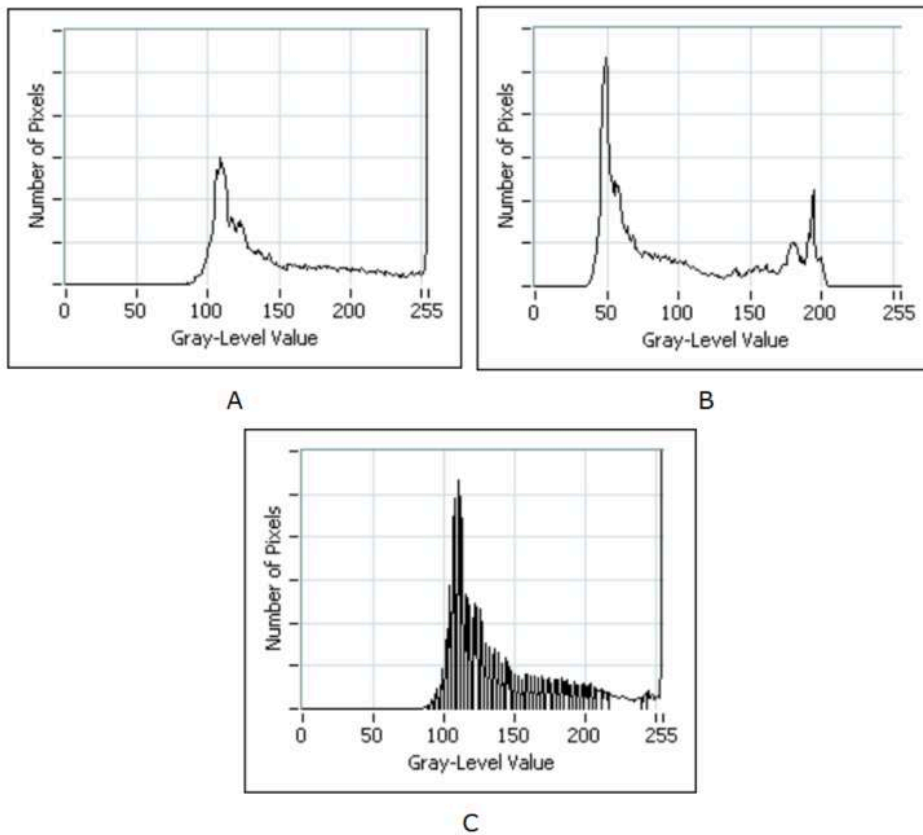


- Figure A shows the golden template.
- Figure B shows the inspection image.
- Figure C shows the defect image.

The label in the inspection image is identical to the label in the golden template. However, the inspection image was acquired under dimmer lighting. Although the images are aligned and corrected for perspective differences, the defect image displays a single, large, dark defect because of the shift in lighting intensity.

Golden template comparison normalizes the pixel intensities in the inspection image using histogram matching. Figure A shows the histogram of the golden template, which peaks in intensity near 110 and then stays low until it saturates at 255. Figure B shows the histogram of the inspection image, which peaks in intensity near 50 and peaks again near 200.

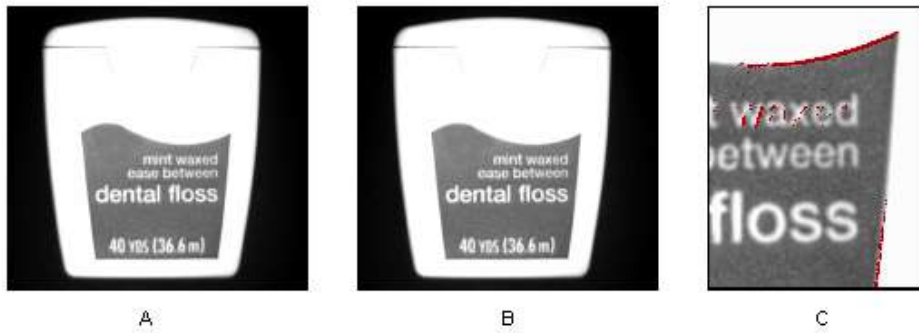
Using a histogram matching algorithm, golden template comparison computes a lookup table to apply to the inspection image. After the lookup table is applied, the histogram of the resulting defect image, shown in figure C, exhibits the same general characteristics as the template histogram. Notice the peak near 110 and the saturation at 255.



Ignoring Edges

Even after alignment, perspective correction, and histogram matching, the defect image may return small defects even when the part under inspection seems identical to the golden template. These small defects are usually confined to edges, or sharp transitions in pixel intensities.

Figure A shows the golden template. Figure B shows the inspection image. The label in the inspection image is almost identical to the label in the golden template. Figure C shows insignificant defects resulting from a small, residual misalignment or quantization errors from the image acquisition. Although these minor variations do not affect the quality of the inspected product, a similarly sized scratch or smudge not on an edge would be a significant defect.



To distinguish minor edge defects from significant defects, you can define edge areas for golden template comparison to ignore using the Vision Template Editor. Differences in areas you want to ignore are not returned as defects. You can preview different edge thicknesses in the training interface, and optionally change edge thickness during runtime.

Using Defect Information for Inspection

Golden template comparison isolates areas in the inspection image that differ from the golden template. To use the defect information in a machine vision application, you need to analyze and process the information using other NI Vision functions. Examples of functions you can use to analyze and process the defect information include particle filters, binary morphology, particle analysis, and binary particle classification.

Optical Character Recognition

This section contains information about optical character recognition (OCR).

Introduction

OCR provides machine vision functions you can use in an application to perform OCR. OCR is the process by which the machine vision software reads text and/or characters in an image. OCR consists of a training phase and either a reading or a verifying phase.

Training characters is the process by which you teach the machine vision software the types of characters or patterns you want to read in the image during the reading procedure. You can use OCR to train any number of characters, creating a character set. The set of characters is later compared with objects during the reading and

verifying procedures. You store the character set in a character set file. Training might be a one-time process, or it might be a process you repeat several times, creating several character sets to broaden the scope of characters you want to detect in an image.

Reading characters is the process by which the machine vision application you create analyzes an image to determine if the objects match the characters you trained. The machine vision application reads characters in an image using the character set that you created when you trained characters.

Verifying characters is a process by which the machine vision application you create inspects an image to verify the quality of the characters it read. The application verifies characters in an image using the reference characters of the character set you created during the training process.

When to Use

Typically, machine vision OCR is used in automated inspection applications to identify or classify components. For example, you can use OCR to detect and analyze the serial number on an automobile engine that is moving along a production line. Using OCR in this instance helps you identify the part quickly, which in turn helps you quickly select the appropriate inspection process for the part.

You can use OCR in a wide variety of other machine vision applications, such as the following:

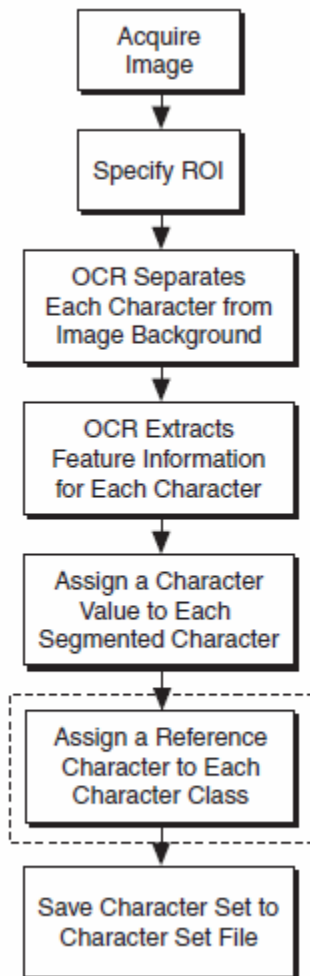
- Inspecting pill bottle labels and lot codes in pharmaceutical applications
- Verifying wafers and IC package codes in semiconductor applications
- Controlling the quality of stamped machine parts.
- Sorting and tracking mail packages and parcels
- Reading alphanumeric characters on automotive parts

Training Characters

Training involves teaching OCR the characters and/or patterns you want to detect during the reading procedure.

All the characters that have been trained with the same character value form a character class. You can designate the trained character that best represents the character value as the reference character for the character class.

The following figure illustrates the steps involved in the training procedure.



Note The diagram item enclosed in dashed lines is an optional step.

The process of locating characters in an image is often referred to as character segmentation. Before you can train characters, you must set up OCR to determine the criteria that segment the characters you want to train. When you finish segmenting the characters, use OCR to train the characters, storing information that enables OCR to recognize the same characters in other images. You train the OCR software by providing a character value for each of the segmented characters, creating a unique

representation of each segmented character. You then save the character set to a character set file to use later in an OCR reading procedure.

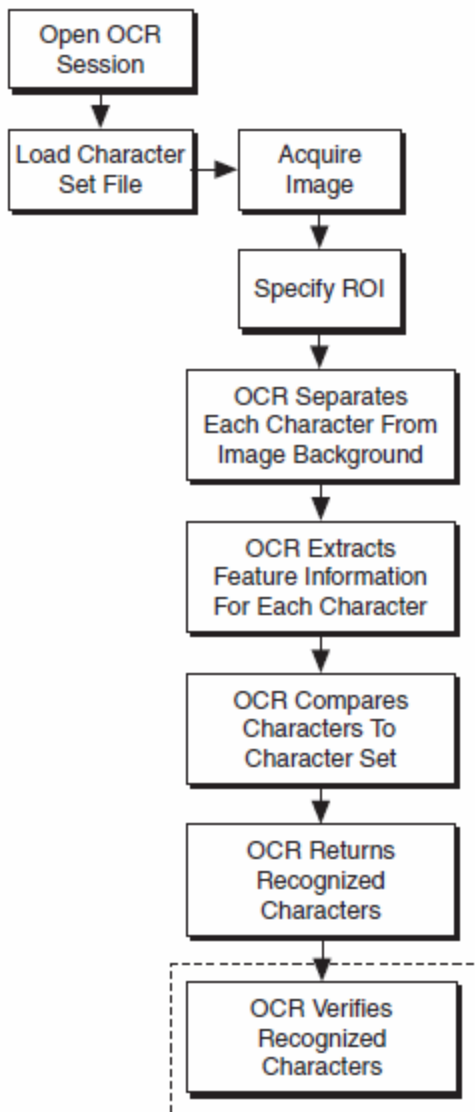
Refer to the NI OCR Training Interface Help that ships with the OCR Training Interface for information about setting up and training characters using OCR.

Reading Characters

When you perform the reading procedure, the machine vision application you create with OCR functions segments each object in the image and compares it to characters in the character set you created during the training procedure. OCR extracts unique features from each segmented object in the image and compares each object to each character stored in the character set. OCR returns the character value of the character in the character set that best matches the object and returns a nonzero classification score. If no character in the character set matches the object, OCR returns the substitution character as the character value and returns a classification score of zero. After reading, you can perform an optional verifying procedure to verify the quality of printed characters.

Refer to Chapter 5, Performing Machine Vision Tasks, of the Vision user manual for your ADE for information about using OCR to read and analyze images for trained characters.

The following figure illustrates the steps involved in the reading procedure.



Note The diagram item enclosed in dashed lines is an optional step.

OCR Session

An OCR session applies to both the training and reading procedures. An OCR session prepares the software to identify a set of characters during either the training procedure or the reading procedure. A session consists of the properties you set and the character set that you train or read from a file. OCR uses session information to compare objects with trained characters to determine if they match. If you want to process an image containing characters that you stored in multiple character sets, use multiple OCR sessions simultaneously to read all the characters simultaneously.

You also can merge several character sets in one session. If you choose to merge multiple character sets, train each of the character sets with the same segmentation parameters.

Concepts and Terminology

The following sections describe OCR concepts and terminology.

Region of Interest (ROI)

The ROI applies to both the training and reading procedures. During training, the ROI is the region that contains the objects you want to train. During reading, the ROI is the region that contains the objects you want to read by comparing the objects to the character set. You can use the ROI to effectively increase the accuracy and efficiency of OCR. During training, you can use the ROI to carefully specify the region in the image that contains the objects you want to train while excluding artifacts. During reading, you can use the ROI to enclose only the objects you want to read, which reduces processing time by limiting the area OCR must analyze.

Particles, Elements, Objects, and Characters

Particles, elements, objects, and characters apply to both the training and reading procedures. Particles are groups of connected pixels. Elements are particles that are part of an object. For example, the dots in a dot-matrix object are elements. A group of one or more elements forms an object based on the element spacing criteria. A character is a trained object.

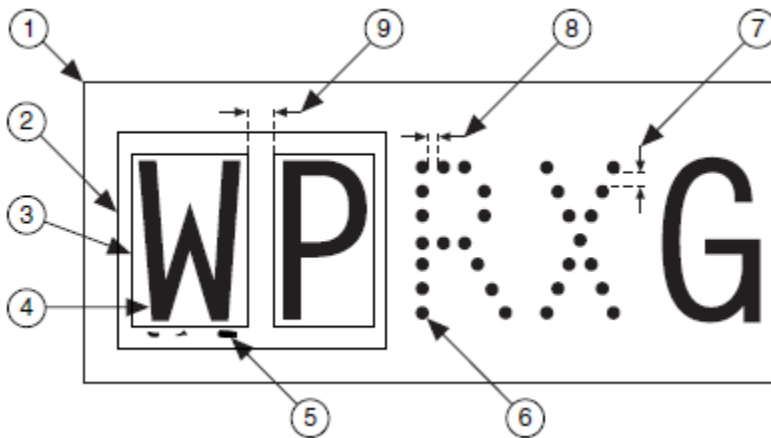
Patterns

Patterns are characters for which the character value is a string of more than one character. For example, a logo is a pattern because it requires a string of more than one character to describe it. Non-ASCII characters are also patterns.

Character Segmentation

Character segmentation applies to both the training and reading procedures. Character segmentation refers to the process of locating and separating each character in the image from the background.

The following illustration shows the concepts included in the character segmentation process.



1. Acquired Image
2. ROI
3. Character Bounding Rectangle
4. Character
5. Artifact
6. Element
7. Vertical Element Spacing
8. Horizontal Element Spacing
9. Character Spacing

Thresholding

Thresholding is one of the most important concepts in the segmentation process. Thresholding is separating image pixels into foreground and background pixels. The standard thresholding method is referred to as global grayscale thresholding. Global grayscale thresholding separates pixels based on their intensity values. Foreground pixels are those whose intensity values are within the lower and upper threshold values of the threshold range. Background pixels are pixels whose intensity values lie outside the lower and upper threshold values of the threshold range.

OCR includes one manual method and three automatic methods of calculating the thresholding range:

- Fixed Range is a method by which you manually set the threshold value. This method processes grayscale images quickly, but requires that lighting remain

uniform across the ROI and constant from image to image.

The following three automatic thresholding methods are affected by the pixel intensity of the objects in the ROI. If the objects are dark on a light background, the automatic methods calculate the high threshold value and set the low threshold value to the lower value of the threshold limits. If the objects are light on a dark background, the automatic methods calculate the low threshold value and set the high threshold value to the upper value of the threshold limits.

- Uniform is a method by which OCR calculates a single threshold value and uses that value to extract pixels from items across the entire ROI. This method is fast and is the best option when lighting remains uniform across the ROI.
- Linear is a method that divides the ROI into blocks, calculates different threshold values for the blocks on the left and right side of an ROI, and linearly interpolates values for the blocks in between. This method is useful when one side of the ROI is brighter than the other and the light intensity changes uniformly across the ROI.
- Non linear is a method that divides the ROI into blocks, calculates a threshold value for each block, and uses the resulting value to extract pixel data.

OCR includes a method by which you can improve performance during automatic thresholding, which includes the Uniform, Linear, and Non linear methods:

- Optimize for Speed allows you to determine if accuracy or speed takes precedence in the threshold calculation algorithm. If speed takes precedence, enable Optimize for Speed to perform the thresholding calculation more quickly, but less accurately. If accuracy takes precedence, disable Optimize for Speed to perform the thresholding calculation more slowly, but more accurately.

If you enable Optimize for Speed, you also can enable Bi modal calculation to configure OCR to calculate both the lower and upper threshold levels for images that are dominated by two pixel intensity levels.

Local Thresholding

Local thresholding, also known as "locally adaptive thresholding" is similar to global grayscale thresholding. However instead of using a thresholding value based on the entirety of the image to determine whether a pixel is part of the foreground or part of the background, local thresholding categorizes a pixel based on the intensity statistics

of neighboring pixels.

Color Thresholding

Thresholding a color image is similar to global grayscale thresholding, however an individual threshold interval must be established for each of the color components.

Threshold Limits

Threshold limits are bounds on the value of the threshold calculated by the automatic threshold calculation algorithms. For example, if the threshold limits are 10 and 240, OCR uses only intensities between 10 and 240 as the threshold value. Use the threshold limits to prevent the OCR automatic threshold algorithms from returning too low or too high values for the threshold in a noisy image or an image that contains a low population of dark or light pixels. The default range is 0 to 255.

Character Spacing

Character spacing is the horizontal distance, in pixels, between the right edge of one character bounding rectangle and the left edge of the next character bounding rectangle.

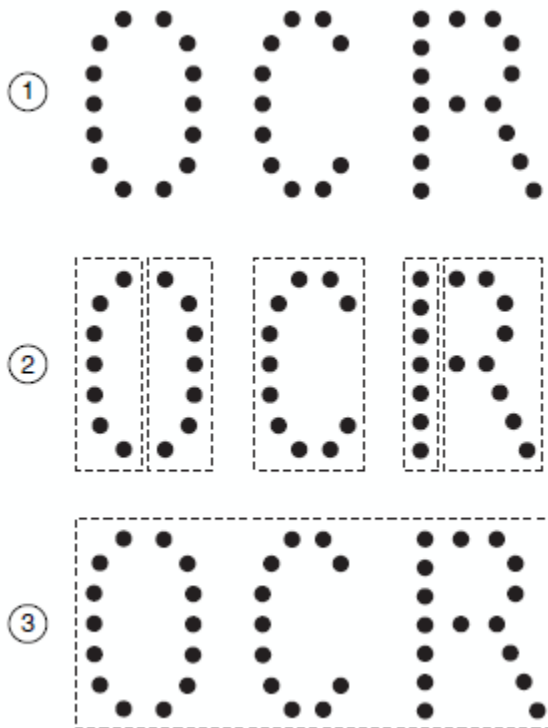
If an image consists of segmented or dot-matrix characters and the spacing between two characters is less than the spacing between the elements of a character, you must use individual ROIs around each character.

Element Spacing

Element spacing consists of horizontal element spacing and vertical element spacing. Horizontal element spacing is the space between two horizontally adjacent elements. Set this value to 1 or 2 for stroke characters and 4 or 5 for dot-matrix or segmented characters. Dot-matrix or segmented characters are characters comprised of a series of small elements. Stroke characters are continuous characters in which breaks are due only to imperfections in the image. If you set the horizontal element spacing too low, you might accidentally eliminate elements of an object. If you set the horizontal element spacing too high, you might include extraneous elements in the object, resulting in a trained object that does not represent a matchable character.

Vertical element spacing is the space between two vertically adjacent elements. Use the default value, 0, to consider all elements within the vertical direction of the ROI to be part of an object. If you set vertical element spacing too high, you might include artifacts as part of an object. If you set vertical element spacing too low, you might eliminate elements that are part of a valid object.

The following illustrations shows how character spacing and element spacing affect OCR.



1. Correct Image
2. Incorrect Element Spacing
3. Incorrect Character Spacing

Item 2 represents an image for which the horizontal element spacing was set incorrectly. The letters O and R are divided vertically because horizontal element spacing was set too low and the OCR segmentation process did not detect that the elements represent a single character. The letter C is trained correctly because the horizontal element spacing value falls within the range that applies to this character. Item 3 represents an image for which the character spacing value was set too high, and thus OCR segments all three letters into one character.

Character Bounding Rectangle


The character bounding rectangle is the smallest rectangle that completely encloses a character.

AutoSplit

AutoSplit applies to both the training and reading procedures. Use AutoSplit when an image contains characters that are slanted. AutoSplit, which works in conjunction with the maximum character bounding rectangle width, uses an algorithm to analyze the right side of a character bounding rectangle and determine the rightmost vertical line in the object that contains the fewest number of pixels. AutoSplit moves the rightmost edge of the character bounding rectangle to that location. The default value is False.

Character Size

Character size is the total number of pixels in a character. Generally, character size should be between 25 and 40 pixels. If characters are too small, training becomes difficult because of the limited data. The additional data included with large characters is not helpful in the OCR process, and the large characters can cause the reading process to become very slow.

 **Tip** You can adjust the character size to filter small particles.

Shortest Segment

Segmentation by the shortest segment algorithm ensures valid segmentation even when the characters are merged. The algorithm observes the max character with configured by the user, or will automatically calculate a value if none has been set.

The algorithm uses the grayscale value of a character as a cost, and executes a shortest path traversal from the top to the bottom of the character.

The algorithm works in three steps:

- Attempt to divide the characters by applying multiple shortest path cuts.
- Choose the cuts that are closest to the max character width.

- Intelligently choose the cuts which segment a character correctly based on classification during reading.



Multiple shortest path cuts applied



Selected shortest path cuts based on max character width



Tip For the best performance with the Shortest Segment algorithm, ensure the bounding rectangle width parameters accurately match the character width during Training. During Reading, if a character set file is loaded, the user should only adjust the bounding box width if the default values are unexpected.

Text Location

Text Location allows a user to set an ROI enclosing multiple lines of text. Multiline detection will identify and return the number of lines bound within the specified ROI. Text Location can detect lines irrespective of small rotations ($\pm 20^\circ$) and differing character heights.

Text location uses particle analysis and clustering based on vertical overlap to detect the lines in a specified ROI. Setting **Number of Lines Expected** to Auto Detect will automatically detect the number of lines and apply character segmentation to all lines.

If the number of lines expected is less than the number of lines identified, will return the number of expected lines, choosing the lines with the highest ranked classification score.



Auto Detect



Number of Lines Expected = 2

Users should avoid using Text Location if the characters are printed using a dot matrix printer and elements in the characters are widely spaced.

Substitution Character

Substitution character applies to the reading procedure only. OCR uses the substitution character for unrecognized characters. The substitution character is a question mark (?) by default.

Acceptance Level

Acceptance level applies to the reading procedure. Acceptance level is a value that indicates how closely a read character must match a trained character to be recognized. Refer to the classification score section of this chapter for more information about how the acceptance level affects character recognition. The valid range for this value is 0 to 1000. The default value is 700. Experiment with different values to determine which value works best for your application.

Read Strategy

Read strategy applies only to the reading procedure. Read strategy refers to the criteria OCR uses to determine if a character matches a trained character in the character set. The possible modes are Aggressive and Conservative. In Aggressive mode, the reading

procedure uses fewer criteria than Conservative mode to determine if an object matches a trained character. Aggressive mode works well for most applications. In Conservative mode, the reading procedure uses extensive criteria to determine if an object matches a trained character.



Note Conservative mode might result in OCR not recognizing characters. Test your application with **Conservative** mode before deciding to use it.

Read Resolution

Read resolution applies to the reading procedure. When you save a character set, OCR saves a variety of information about each character in the character set. Read resolution is the level of character detail OCR uses to determine if an object matches a trained character. By default, OCR uses a low read resolution, using few details to determine if there is a match between an object and a trained character. The low read resolution enables OCR to perform the reading procedure more quickly. You can configure OCR to use a medium or high read resolution, and therefore use more details to determine if an object matches a trained character. Using a high read resolution reduces the speed at which OCR processes.

The low resolution works well with most applications, but some applications might require the higher level of detail available in medium or high resolutions.



Note Using medium or high resolution might result in OCR not recognizing characters. If you choose to use medium or high resolution, test your application thoroughly.

Valid Characters

Valid characters applies only to the reading procedure. Valid characters refers to the practice of limiting the characters that the reading procedure uses when analyzing an image. For example, if you know that the first character in an ROI should be a number, you can limit the reading procedure to comparing the first character in the ROI only to numbers in the character set. Limiting the characters that the reading procedure uses when analyzing an image increases the speed and accuracy of OCR.

Aspect Ratio Independence

Aspect ratio independence applies only to the reading procedure. Aspect ratio independence is the ability to read characters at a different size and height/width ratio than the training size and height/width ratio. To maintain performance in the OCR process, National Instruments recommends you limit the difference to $\pm 50\%$. Avoid creating character sets whose characters differ only in height and width. Consider separating the characters into different character sets, using valid characters to restrict trained characters, and enforcing the aspect ratio.

OCR Scores

The following sections describe the scores returned by the reading procedure.

Classification Score

The classification score indicates the degree to which the assigned character class represents the input object better than other character classes in the character set. It is defined as follows:

$$\text{Classification Score} = (1 - d_1 / d_2) \times 1000$$

where d_1 is the distance of the object to the best match in the closest class, and d_2 is the distance of the object to the best match in the second closest class. Distance is defined as a measure of the differences between the object and a trained character. The smaller the distance, the closer the object is to the trained character. Because $d_1 \leq d_2$, the classification score is between 0 and 1000. A trained character is considered a match only if the distance between the object and the trained character is smaller than a value controlled by the acceptance level. The larger the acceptance level, the smaller the distance between the object and the trained character has to be for OCR to match the object.

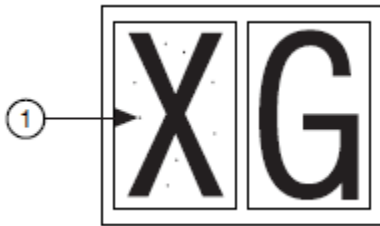
Verification Score

If an input object belongs to a character class for which a reference character has been designated, OCR compares the object to the reference character and outputs a score that indicates how closely the input object matches the reference character. The score ranges from 0 to 1000, where 0 represents no similarity and 1000 represents a perfect

match. You can use this score to verify the quality of printed characters.

Removing Small Particles

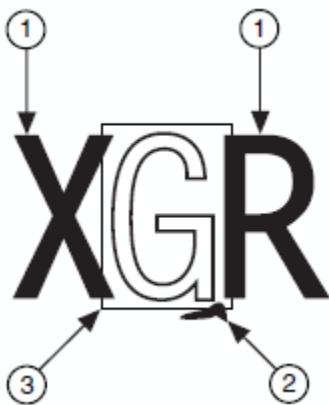
Removing small particles applies to both the training and reading procedures. The process of removing small particles involves applying a user-specified number of $\times 3$ erosions to the thresholded image. OCR fully restores any objects that remain after applying the erosions. For example, in the following figure, if any portion of the letters X and G remains after removing small particles, OCR fully restores the X and G.



1. Particle

Removing Particles That Touch the ROI

Removing particles that touch the ROI applies to both the training and reading procedures. You can configure OCR to remove small particles that touch an ROI you specified. Refer to the following figure for examples of particles that touch the ROI.



1. Particle to Remove
2. Incorrect Artifact to Remove
3. ROI

Instrument Readers

This section contains information about instrument readers that read meters, liquid crystal displays (LCDs), barcodes, and 2D codes.

Introduction

Instrument readers are functions you can use to accelerate the development of applications that require reading meters, seven segment displays, barcodes, and 2D codes.

When to Use

Use instrument readers when you need to obtain information from images of simple meters, LCD displays, barcodes, and 2D codes.

Meter Functions

Meter functions simplify and accelerate the development of applications that require reading values from meters or gauges. These functions provide high-level vision processes to extract the position of a meter or gauge needle.

You can use this information to build different applications such as the calibration of a gauge. Use the functions to compute the base of the needle and its extremities from an area of interest indicating the initial and the full-scale position of the needle. You then can use these VIs to read the position of the needle using parameters computed earlier.

The recognition process consists of the following two phases:

- A learning phase during which the user must specify the extremities of the needle.
- An analysis phase during which the current position of the needle is determined.

The meter functions are designed to work with meters or gauges that have either a dark needle on a light background or a light needle on a dark background.

Meter Algorithm Limits

This section explains the limit conditions of the algorithm used for the meter functions. The algorithm is fairly insensitive to light variations.

The position of the base of the needle is very important in the detection process. Carefully draw the lines that indicate the initial and the full-scale position of the needle. The coordinates of the base and of the points of the arc curved by the tip of the needle are computed during the setup phase. These coordinates are used to read the meter during inspection.

LCD Functions

LCD functions simplify and accelerate the development of applications that require reading values from seven-segment displays.

Use these functions to extract seven-segment digit information from an image.

The reading process consists of two phases:

- A learning phase during which the user specifies an area of interest in the image to locate the seven-segment display.
- A reading phase during which the area specified by the user is analyzed to read the seven-segment digit.

The NI Vision LCD functions provide the high-level vision processes required for recognizing and reading seven-segment digit indicators. The LCD functions are designed for seven-segment displays that use either LCDs or LEDs composed of electroluminescent indicators or light-emitting diodes, respectively.

The LCD functions can perform the following tasks:

- Detect the area around each seven-segment digit from a rectangular area that contains multiple digits
- Read the value of a single digit
- Read the value, sign, and decimal separator of the displayed number

LCD Algorithm Limits

The following factors can cause a bad detection:

- Very high horizontal or vertical light drift
- Very low contrast between the background and the segments
- Very high level of noise
- Very low resolution of the image

Each of these factors is quantified to indicate when the algorithm might not give accurate results.

Light drift is quantified by the difference between the average pixel values at the top left and the bottom right of the background of the LCD screen. Detection results might be inaccurate when light drift is greater than 90 in 8-bit images.

Contrast is measured as the difference between the average pixel values in a rectangular region in the background and a rectangular region in a segment. This difference must be greater than 30 in 8-bit images, which have 256 gray levels, to obtain accurate results.

Noise is defined as the standard deviation of the pixel values contained in a rectangular region in the background. This value must be less than 15 for 8-bit images, which have 256 gray levels, to obtain accurate results.

Each digit must be larger than 18×12 pixels to obtain accurate results.

Barcode Functions

Vision currently supports the following barcode formats: Code 25, Code 39, Code 93, Code 128, EAN 8, EAN 13, Codabar, MSI, UPC A, Pharmacode, and GS1 DataBar Limited (previously referred to as RSS-14 Limited).

The process used to recognize barcodes consists of two phases:

- A learning phase in which the user specifies an area of interest in the image which helps to localize the region occupied by the barcod.
- The recognition phase during which the region specified by the user is analyzed to

decode the barcode.

Barcode Algorithm Limits

The following factors can cause errors in the decoding process:

- Very low resolution of the image
- Very high horizontal or vertical light drift
- Contrast along the bars of the image
- High level of noise

The limit conditions are different for barcodes that have two different widths of bars and spaces—such as Code 39, Codabar, Code 25, MSI, and Pharmacode—and for barcodes that have more than two widths of bars and spaces—such as Code 93, Code 128, EAN 13, EAN 8, and UPC A, and GS1 DataBar Limited (previously referred to as RSS-14 Limited).

The resolution of an image is determined by the width of the smallest bar and space. These widths must be at least 3 pixels for all barcodes.

Light drift is quantified by the difference between the average of the gray level of the left, or upper, line and the right, or bottom, line of the background of the barcode. Decoding inaccuracies can occur if the light drift is greater than 120 for barcodes with two different widths of bars and spaces and greater than 100 for barcodes with four different widths of bars and spaces.

In overexposed images, the gray levels of the wide and narrow bars in the barcode tend to differ. Decoding results may not be accurate when the difference in gray levels is less than 80 for barcodes with two different widths of bars and spaces, and less than 100 for barcodes with four different widths of bars and spaces.

Consider the difference in gray levels between the narrow bars and the wide bars. The narrow bars are scarcely visible. If this difference of gray level exceeds 115 on 8-bit images (256 gray levels) for barcodes with two different widths of bars and spaces and 100 for barcodes with four different widths of bars and spaces, the results may be inaccurate.

Noise is defined as the standard deviation of a rectangular region of interest drawn in

the background. It must be less than 57 for barcodes with two different widths of bars and spaces and less than 27 for barcodes with four different widths of bars and spaces.

Reflections on the barcode can introduce errors in the value read from the barcode. Similarly, bars and spaces that are masked by the reflection produce errors.

2D Code Recognition

The term 2D code refers to both matrix codes and multi-row barcodes. Matrix codes encode data based on the position of square, hexagonal, or round cells within a matrix. Multi-row barcodes are codes that consist of multiple stacked rows of barcode data. NI Vision currently supports the PDF417, Data Matrix, QR Code, and Micro QR Code formats. The process used to recognize 2D codes consists of two phases:

- A coarse locating phase during which the user specifies an ROI in the image, which helps localize the region occupied by the 2D code. This phase is optional, but it can increase the performance of the second phase by reducing the size of the search region.
- A locating and decoding phase during which the software searches the ROI for one or more 2D codes and decodes each located 2D code.

What to Expect from 2D Code Recognition

The following factors can cause errors in the search and decoding phases or 2D code recognition:

- Very low resolution of the image.
- Very high horizontal or vertical light drift.
- Contrast along the bars of the image.
- High level of noise or blurring.
- Inconsistent printing or stamping techniques, such as misaligned code elements, inconsistent element size, or elements with inconsistent borders.
- In PDF417 codes, a quiet zone that is too small or contains too much noise.

Related concepts:

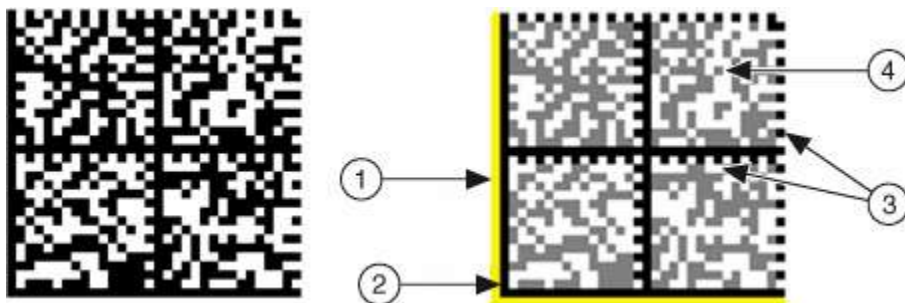
- [2D Code Recognition](#)

Data Matrix Concepts

A Data Matrix code is a matrix built on a square or rectangular grid with a finder pattern around the perimeter of the matrix. Each cell of the matrix contains a single data cell. The cells can be either square or circular.

Locating and decoding Data Matrix codes requires a minimum cell size of 2.5 pixels. Locating and decoding Data Matrix codes also requires a quiet zone of at least one cell width around the perimeter of the code. However, a larger quiet zone increases the likelihood of successful location. Each symbol character value is encoded in a series of data cells called a code word.

Data Matrix codes use one of two error checking and correction (ECC) schemes. Data Matrix codes that use the ECC schemes 000 to 140 are based on the original specification. These codes use a convolution error correction scheme and use a less efficient data packing mechanism that often requires only encoding characters from a particular portion of the ASCII character set. Data Matrix codes that use the ECC 200 scheme use a Reed-Solomon error correction algorithm and a more efficient data packing mechanism. The ECC 200 scheme also allows for the generation of multiple connected matrices, which enables the encoding of larger data sets. The following figure shows an example of a Data Matrix code:



1. Quiet Zone
2. Finder Pattern
3. Clock Pattern
4. Data Cell

Related concepts:

- [2D Code Recognition](#)

Quality Grading

Vision can assess the quality of a Data Matrix code based on how well the code meets certain parameters. For each parameter, Vision returns one of the following letter grades: A, B, C, D, or F. An A indicates that the code meets the highest standard for a particular parameter. An F indicates that the code is of the lowest quality for that parameter. NI Vision support the following grading standards:

- ISO 16022
- ISO 15415
- AIM DPM

Related concepts:

- [Data Matrix Concepts](#)

Decode

The decoding process tests whether the Data Matrix features are correct enough to be readable when the code is optimally imaged. The code is assigned an A or F, based on whether the decoding is successful or not. The decoding process also locates and defines the area covered by the code in the image, adaptively creates a grid mapping of the data cell centers, and performs error correction.

Related concepts:

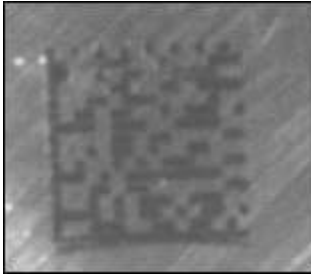
- [Quality Grading](#)

Symbol Contrast

The symbol contrast test determines whether the light and dark pixels in the image are sufficiently and consistently distinct throughout the code. All pixels are sorted by their reflectance values to determine the darkest 10% and lightest 10%. The mean reflectance of the darkest 10% and the mean reflectance of the lightest 10% are calculated. The difference of the two means is the symbol contrast. The following list shows how the symbol contrast is graded.

- A (4.0) if symbol contrast $\geq 70\%$
- B (3.0) if symbol contrast $\geq 55\%$
- C (2.0) if symbol contrast $\geq 40\%$
- D (1.0) if symbol contrast $\geq 20\%$
- F (0.0) if symbol contrast $< 20\%$

The following figure shows a Data Matrix code with a symbol contrast value of 8.87%, which returns a grade of F.



Related concepts:

- [Quality Grading](#)

Print Growth

The print growth test determines the extent to which dark or light markings appropriately fill their cell boundaries. This parameter is an important indication of process quality, which affects the reading performance of the function. The print growth grade is based on the dimension with the largest print growth (D'). The dimensions (D) of the markings are determined by counting pixels in the image. Horizontal and vertical dimensions are checked separately. For each dimension, the following values are specified:

- nominal value (D_{nom}) = 0.50
- maximum value (D_{max}) = 0.65
- minimum value (D_{min}) = 0.35

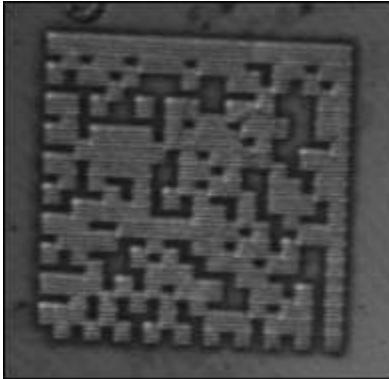
Normalize each measured D to its corresponding nominal and limit values:

if $D > D_{\text{nom}}$, then $D' = (D - D_{\text{nom}}) / (D_{\text{max}} - D_{\text{nom}})$ otherwise $D' = (D - D_{\text{nom}}) / (D_{\text{nom}} - D_{\text{min}})$

The following list shows how print growth is graded:

- A (4.0) if $-0.50 \leq D' \leq 0.50$
- B (3.0) if $-0.70 \leq D' \leq 0.70$
- C (2.0) if $-0.85 \leq D' \leq 0.85$
- D (1.0) if $-1.00 \leq D' \leq 1.00$
- F (0.0) if $D' < -1.00$ or $D' > 1.00$

The following figure shows a Data Matrix code with a print growth value of 0.79, which returns a grade of C.



Related concepts:

- [Quality Grading](#)

Axial Nonuniformity

Axial nonuniformity is a measure of how much the sampling point spacing differs from one axis to another.

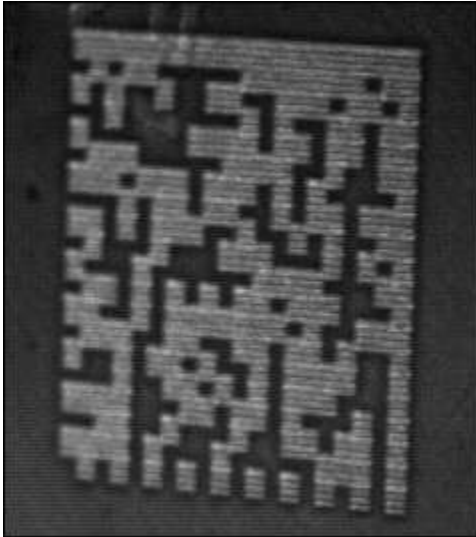
Axial nonuniformity measures and grades the spacing of the cell centers. Axial nonuniformity tests for uneven scaling of the code, which would inhibit readability at some atypical viewing angles. The spacings between adjacent sampling points are independently sorted for each polygonal axis. Then the average spacing (X_{avg}) along each axis is computed.

$$\text{Axial Nonuniformity} = \text{abs}(X_{avg} - Y_{avg}) / ((X_{avg} + Y_{avg}) / 2)$$

where $\text{abs}()$ yields the absolute value. The following list shows how axial nonuniformity is graded.

- A (4.0) if axial nonuniformity ≤ 0.06
- B (3.0) if axial nonuniformity ≤ 0.08
- C (2.0) if axial nonuniformity ≤ 0.10
- D (1.0) if axial nonuniformity ≤ 0.12
- F (0.0) if axial nonuniformity > 0.12

The following figure shows a Data Matrix code with an axial nonuniformity value of 0.2714, which returns a grade of F.



Related concepts:

- [Quality Grading](#)

Unused Error Correction

Unused error correction tests the extent to which regional or spot damage in the symbol has eroded the reading safety margin that the error correction provides. The convolutional error encoding for Data Matrix codes ECC 000-ECC 140 can correct for the following maximum percentages of bit errors (E_{\max}):

- ECC 000: $E_{\max} = 0.0\%$
- ECC 050: $E_{\max} = 2.8\%$
- ECC 080: $E_{\max} = 5.5\%$
- ECC 100: $E_{\max} = 12.6\%$
- ECC 140: $E_{\max} = 25.0\%$

The actual percentage of bit errors (E_{act}) is the number of bits that were corrected divided by the total number of bits in the symbol data fields. The unused error correction for Data Matrix codes ECC 000-ECC 140 is expressed as:

$$\text{Unused Error Correction} = 1.0 - (E_{act} / E_{max})$$

For ECC 200 codes, the correction capacity of the Reed-Solomon decoding is expressed as:

$$e + 2t \leq d - p$$

where

e is the number of erasures

t is the number of errors

d is the number of error correction code words

p is the number of code words reserved for error detection

Values for d and p are defined by the specification for the given symbol. Values e and t are determined during a successful decode process. The amount of unused error correction is computed as:

$$\text{Unused Error Correction} = 1.0 - (e + 2t) / (d - p)$$

In codes with more than one Reed-Solomon block, the unused error correction is calculated for each block independently, and the lowest value is used to calculate the unused error correction grade. The following list shows how unused error correction is graded.

- A (4.0) if unused error correction ≥ 0.62
- B (3.0) if unused error correction ≥ 0.50
- C (2.0) if unused error correction ≥ 0.37
- D (1.0) if unused error correction ≥ 0.25
- F (0.0) if unused error correction < 0.25

The following figure shows a Data Matrix code with an unused error correction value of

0.00, which returns a grade of F.



Related concepts:

- [Quality Grading](#)

Overall Symbol Grade

The overall symbol grade is the lowest of the grades from the other symbol parameters.

Related concepts:

- [Quality Grading](#)

ISO 15415 Grading Standard Concepts

ISO 15415 is an extension of the ISO 16022 grading system. ISO 15415 uses the grading parameters of the ISO 16022 grading scheme, as well as the following additional parameters:

- Modulation
- Grid Nonuniformity
- Fixed Pattern Damage

Related concepts:

- [Quality Grading](#)
- [Decode](#)
- [Symbol Contrast](#)
- [Print Growth](#)

- [Axial Nonuniformity](#)
- [Unused Error Correction](#)
- [Overall Symbol Grade](#)

Modulation

Modulation is a measure of the uniformity of reflectance of the dark and light modules in a 2D barcode. Lower modulation may increase the probability of a module being incorrectly identified as dark or light.

Modulation is affected by print growth or loss, defects, reflectance, and variation of the ink coverage. For example, the following figure illustrates a Data Matrix with variations in reflectance:



Modulation is expressed as:

$$\text{Modulation} = 2 \cdot (\text{abs}(R - GT)) / SC$$

where,

R is the reflectance of the module closest to the global threshold in the codeword,

GT is the global threshold.

The mean reflectance of the darkest 10% and the mean reflectance of the lightest 10% is determined. The average of the two means is taken as the global threshold.

The modulation grade for a codeword is computed as the minimum grade of all the data cells in a particular codeword. The final modulation grade is determined by comparing the number of codewords with a particular grade or higher and the error

correction capacity of the given data matrix barcode. The following list shows how codeword grading for modulation is calculated.

- A (4.0) if modulation ≥ 0.50
- B (3.0) if modulation ≥ 0.40
- C (2.0) if modulation ≥ 0.30
- D (1.0) if modulation ≥ 0.20
- F (0.0) if modulation < 0.20

The following figure shows a Data Matrix code with a modulation grade of F.



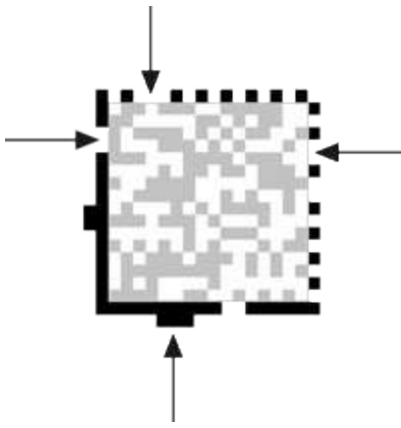
Related concepts:

- [ISO 15415 Grading Standard Concepts](#)

Fixed Pattern Damage

Fixed pattern damage measures the damage in the finder pattern, quiet zone, clock pattern and solid area segment regions. Pattern damage can be caused by an improper printer nozzle, a faulty thermal element, or physical damage to the barcode.

The following figure illustrates damage to the fixed pattern and the clock pattern:



The overall fixed pattern grade is the average of all grades. The final grade is the minimum of the finder pattern grade, quiet zone grade, clock pattern grade, solid area grade, or overall fixed pattern grade. The following list shows how overall fixed pattern damage is calculated.

- A (4.0) if overall FPD grade = 4.0
- B (3.0) if overall FPD grade ≥ 3.5
- C (2.0) if overall FPD grade ≥ 3.0
- D (1.0) if overall FPD grade ≥ 2.5
- F (0.0) if overall FPD grade < 2.5

The following figure shows a Data Matrix code with a fixed pattern damage score of 2.4, which returns a grade of F and modulation grade of C.



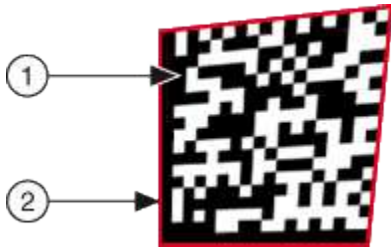
Related concepts:

- [ISO 15415 Grading Standard Concepts](#)

Grid Nonuniformity

Grid nonuniformity measures the largest deviation of the grid centers from their ideal

theoretical position, as determined by the reference decode algorithm. The measurements are expressed as a fraction of the width of the symbol under test. The following image illustrates grid nonuniformity by comparing an actual symbol under test with the border of the ideal grid position.



1. Symbol under test
2. Ideal grid position outline

The following list shows how grid non-uniformity is graded.

- A (4.0) if grid non uniformity ≤ 0.38
- B (3.0) if grid non uniformity ≤ 0.50
- C (2.0) if grid non uniformity ≤ 0.63
- D (1.0) if grid non uniformity ≤ 0.75
- F (0.0) if grid non uniformity > 0.75

The following figure shows a Data Matrix code with a grid nonuniformity score of 1.88, which returns a grade of F.



Related concepts:

- [ISO 15415 Grading Standard Concepts](#)

Scan Grade

The overall symbol grade is the lowest of the grades from the other symbol parameters. The scan grade applies to the current image and contributes to the overall symbol grade.

Related concepts:

- [ISO 15415 Grading Standard Concepts](#)

Overall Symbol Grade

The overall symbol grade evaluates apparent variation in symbol characteristics when the Data Matrix code is viewed from different orientations relative to optical axis of camera.

The overall symbol grade requires five images of the Data Matrix code, acquired using the same aperture and light source, in which the Data Matrix code is rotated by 72° ($\pm 5^\circ$) for each image. The overall symbol grade is the mean of the scan grades for each of the five images. If any two acquired images provide different decoded data, the overall symbol grade is 0.

Related concepts:

- [ISO 15415 Grading Standard Concepts](#)

AIM DPM Grading Standard Concepts

AIM DPM is an extension of the ISO 15415 grading system that requires an initial system adjustment for gain and exposure parameters prior to grading.

To perform the adjustment, calculate the mean of the light lobes of a given data matrix barcode. If the mean of the light lobes is not within the range of 70-86%, adjust the gain and exposure parameters and repeat the calculation. AIM DPM uses the decode, print growth, axial nonuniformity, unused error correction, grid nonuniformity, fixed

pattern damage, scan grade, and overall symbol grade parameters of the ISO 15415 grading scheme, as well as the following additional parameters:

- Cell Contrast
- Cell Modulation
- Minimum Reflectance

Cell Contrast

Cell contrast tests whether the light and dark pixels in the image are sufficiently and consistently distinct throughout the code. The angle of illumination, reflectance, and variation of ink coverage can all affect cell contrast. The cell contrast parameter is similar to the symbol contrast parameter used in the ISO 15415 grading scheme; however, the process for computing cell contrast is different from the process used to compute symbol contrast.

To compute cell contrast, a histogram is constructed using the grid centers of the data matrix code. Mean light and mean dark values are computed for the light and dark elements of the grid center, and are used to calculate the cell contrast as shown in the following equation:

$$\text{Cell Contrast} = (\text{Mean Light} - \text{Mean Dark}) / \text{Mean Light}$$

The following list shows how cell contrast is graded.

1. A (4.0) if cell contrast ≥ 0.3
2. B (3.0) if cell contrast ≥ 0.25
3. C (2.0) if cell contrast ≥ 0.20
4. D (1.0) if cell contrast ≥ 0.15
5. F (0.0) if cell contrast < 0.15

The following figure shows a Data Matrix code with a cell contrast of 0.7045, which produces a grade of A.



Related concepts:

- [AIM DPM Grading Standard Concepts](#)

Cell Modulation

The cell modulation parameter is similar to the modulation parameter used in the ISO 15415 grading scheme; however, the process for computing cell modulation is different from the process used to compute modulation. Cell modulation is computed according to the following formula:

If $(R < T2)$ then $CM = (T2 - R)/(T2 - MD)$ otherwise $CM = (R - T2)/(\text{Mean Light Target} - T2)$

where R is the reflectance of the cell T2 is the threshold created using the histogram of the grid center MD is the mean of the dark lobe from the final grid-point histogram Mean Light Target is the mean of the light lobe from the final grid-point histogram The following list shows how codeword grading for modulation is calculated.

- A (4.0) if modulation ≥ 0.50
- B (3.0) if modulation ≥ 0.40
- C (2.0) if modulation ≥ 0.30
- D (1.0) if modulation ≥ 0.20
- F (0.0) if modulation < 0.20

The following figure shows a Data Matrix code with a cell modulation grade of D.



Related concepts:

- [AIM DPM Grading Standard Concepts](#)

Minimum Reflectance

Minimum reflectance is the minimum reflectance required by the symbol under test compared to a standard calibration card.

To produce the calibration values required to calculate minimum reflectance, use a calibration card and perform the initial system adjustment required for AIM PDF. The gain and exposure settings used during the initial system adjustment define the calibrated system parameter for the minimum reflectance calculation. The mean light value produced by the adjustment defines the calibrated mean light parameter for the minimum reflectance calculation.

Following calibration, repeat the process with the data matrix symbol under test. If the mean light value does not fall within the desired range of 70-86%, adjust the system parameter. The gain and exposure settings used under test define the target system parameter for the minimum reflectance calculation. The mean light value produced under test defines the target mean light parameter for the minimum reflectance calculation. The reflectance for the symbol to grade is defined by the following equation:

$$\text{Minimum Reflectance} = \frac{\text{Calibrated System Parameter}}{\text{Target System Parameter}} \cdot \frac{\text{Target Mean Light}}{\text{Calibrated Mean Light}}$$

The following list shows how minimum reflectance contrast is graded.

- A (4.0) if Minimum Reflectance $\geq 5\%$
- F (0.0) if Minimum Reflectance $< 5\%$

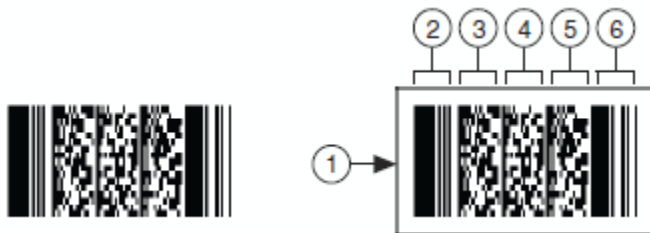
Related concepts:

- [AIM DPM Grading Standard Concepts](#)

PDF417 Concepts

A PDF417 code is a multi-row barcode in which each data element is encoded in a code word. Each row consists of a start pattern, a left row indicator code word, one to 30 data code words, a right row indicator code word, and a stop pattern. Each code word consists of 17 cells and encodes four bars and four spaces. Each bar and each space has a maximum width of six cells.

Locating and decoding PDF417 codes requires a minimum cell size of 1.5 pixels and a minimum row height of 4.5 pixels. Locating and decoding PDF417 codes also requires a quiet zone of at least one cell width around the perimeter of the code. However, a larger quiet zone increases the likelihood of successful location. The following figure shows an example of a PDF417 code:



1. Quiet Zone
2. Start Pattern
3. Left Row Indicator
4. Data Code Words
5. Right Row Indicator
6. Stop Pattern

Related concepts:

- [2D Code Recognition](#)

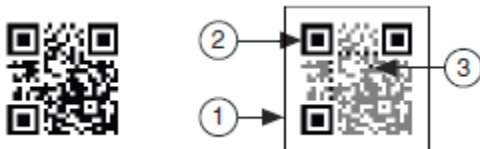
QR Code Concepts

A QR Code is a matrix built on a square grid with a set of finder patterns located at three corners of the matrix. Finder patterns consist of alternating black and white square rings. The size of the matrix can range from a minimum size of 21×21 up to a maximum size of 177×177 . Each cell of the matrix contains a single data cell. Matrix cells are square and represent a single binary 0 or 1.

Locating and decoding QR Codes requires a minimum cell size of 2.5 pixels. Locating and decoding PDF417 codes also requires a quiet zone of at least one cell width around the perimeter of the code. However, a larger quiet zone increases the likelihood of successful location. Each symbol character value is encoded in a unit called a code word consisting of 8 cells or one byte of data.

QR Codes have built in error checking and correction (ECC) using the standard Reed-Solomon scheme for error correction. The amount of error correction capability of each code is selectable during the printing process. In general, the QR Code can correct for anywhere from 7% to 30% of error depending upon the selection made at print time.

The following figure shows a example of a QR Code:



1. Quiet Zone
2. Finder Pattern
3. Data Cell

Related concepts:

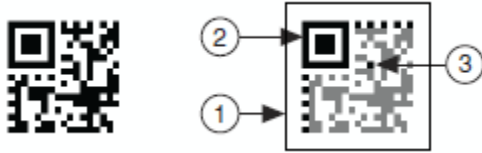
- [2D Code Recognition](#)

Micro QR Code Concepts

A Micro QR Code is a smaller version of the standard QR Code. Micro QR Codes have

only one finder pattern located at one corner of the matrix. The size of a Micro QR Code can range from a minimum size of 11×11 up to a maximum size of 17×17 .

The following figure shows an example of a Micro QR Code:



1. Quiet Zone
2. Finder Pattern
3. Data Cell

Related concepts:

- [QR Code Concepts](#)

Stereo Vision

A stereo vision system uses multiple cameras to acquire multiple overlapping images of a single region of interest within a scene. Use a stereo vision system to compute relative depth information or precise 3D measurements for a scene.

Stereo Vision in Vision

Vision supports binocular stereo vision systems. A binocular stereo vision system uses exactly two cameras. Ideally, the two cameras are separated by a short distance, or baseline, and are mounted almost parallel to one another.

Related concepts:

- [Stereo Vision](#)

When to Use Stereo Vision

A stereo vision system requires fixed camera settings and locations during and after

calibration. As a result, a stereo vision system is not suitable for applications where the platform on which cameras are mounted can experience strong disturbances. It is also not suitable for applications which require changing camera settings while making measurements. For best results, Vision prefers a horizontal baseline, or a system where the horizontal distance exceeds the vertical distance between the two cameras.

A typical stereo vision system is passive. Stereo vision systems use stationary cameras and do not require moving parts, such as a laser. A stereo vision system allows you to establish a stereo correspondence and calculate the disparity, or horizontal difference, between corresponding, or conjugate, points in images captured by the system. Disparity information is rendered as a disparity map, which you can use to determine the relative depth of an object.

Stereo images can also be processed to produce very dense depth information that can be mapped to real-world coordinates.

Disparity and depth information produced with a stereo vision system can be used in conjunction with other algorithms, such as pattern matching or object tracking, to profile stationary or moving objects.

Related concepts:

- [Stereo Vision](#)

Stereo Vision in Navigation Applications

Stereo vision systems are commonly used by robots for navigation. Autonomous mobile robots use relative distance information available from disparity maps to avert obstacles. Such robots might also use depth information to measure size and distance of obstacles for accurate path planning. Stereo vision systems are also used for navigation by outdoor autonomous vehicles, hospital service robots and automobile systems for providing depth information to human drivers.

Related concepts:

- [When to Use Stereo Vision](#)

Stereo Vision in Robotic Applications

A stereo vision system is useful in robotic industrial automation of tasks such as bin picking or crate handling.

A bin-picking application requires a robot arm to pick a specific object from a container that holds several different kinds of parts. With a single camera, part occlusion and lighting variation make it difficult to determine which part is on top of the pile and easy to grasp. A stereo vision system can provide an inexpensive way to obtain 3D information and determine which parts are free to be grasped.

Stereo vision is also effective in crate handling applications; for example, using a robotic arm to remove fruit or bottles from a crate. 3D information obtained from a stereo vision system can provide precise locations for individual fruits or the caps of bottles in a crate. This enables applications in which a robot arm picks an object from a pallet and moves it to another pallet or process.

Related concepts:

- [When to Use Stereo Vision](#)

Stereo Vision in Machine Vision Applications

3D information can be used to locate objects in machine vision applications. For example, you can use 3D information to identify individual fruit for inspection or to verify the presence of pills in a blister pack. 3D information can also be used to inspect and make measurements on automotive parts or electronic components such as solder paste or ball-grid arrays.

Related concepts:

- [When to Use Stereo Vision](#)

Stereo Vision in Surveillance Applications

Stereo vision systems are good for tracking applications because they are robust in the presence of lighting variations and shadows. A stereo vision system can accurately provide 3D information for tracked objects which can be used to detect abnormal

events, such as trespassing individuals or dropped baggage. Stereo vision systems can also enhance biometric systems such as facial recognition systems.

Related concepts:

- [When to Use Stereo Vision](#)

What to Expect from a Stereo Vision System

This topic describes conditions that can affect the performance of a stereo vision system. Depth resolution refers to the accuracy with which a stereo vision system can estimate changes in the depth of a surface. Depth resolution is represented by the following equation:

$$\Delta z = \frac{z^2}{f_b} \Delta d$$

where,

z is the depth of the object from the stereo system,

f is the focal length,

b is the baseline,

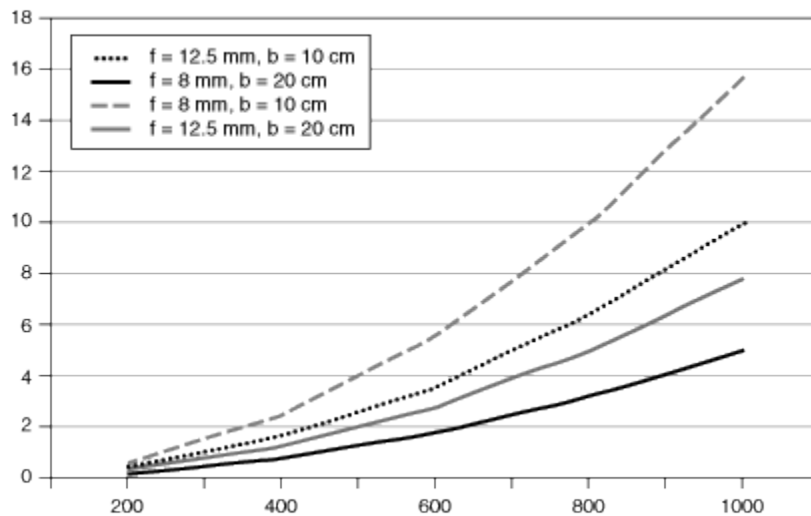
d is disparity.

Depth resolution is proportional to the square of the depth (z) and is inversely proportional to the focal length (f) and baseline (b), or distance between the cameras.

Good depth resolution requires a large baseline (b) value, a large focal length (f) value and a small depth (z) value.

Depth resolution depends on accurate disparity estimations. The accuracy of disparity estimation is directly proportional to pixel size, where smaller pixel sizes provide better resolution. Typically, disparity can be estimated accurately to about one fifth of a pixel. For a camera with a pixel size of 7.5 microns, this translates to a disparity resolution of 1.5 microns.

The following figure illustrates the relationship of the depth resolution to the depth of an object for a given focal length and baseline, and a disparity resolution of 1.5 microns.



Range of depth refers to the minimum and maximum distances of objects that can be measured by the stereo vision system for a given maximum disparity. For a simple stereo system, the depth of a point (z) is given by:

$$z = \frac{f^2}{d}$$

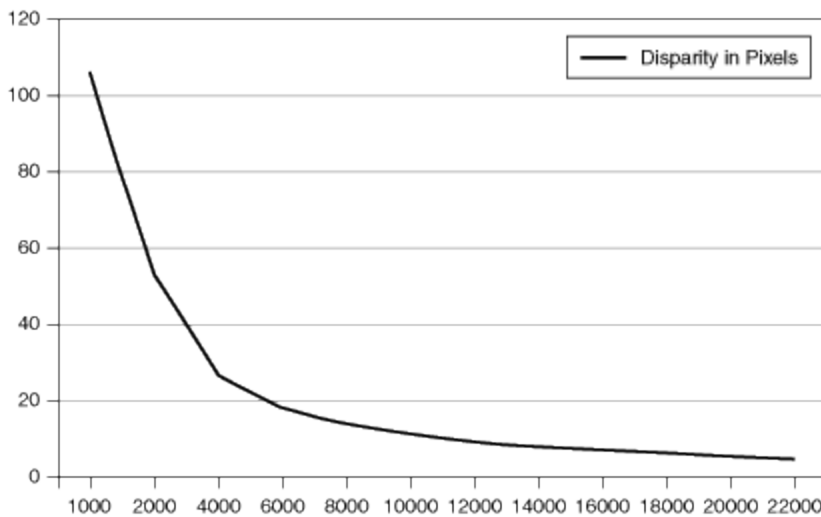
where,

f is the focal length,

b is the baseline,

d is disparity.

The following figure plots depth values as a function of disparity, assuming a focal length (f) of 8mm, baseline (b) of 10cm and pixel size of 7.5 microns:



The figure illustrates that for an object which is approximately 1000 mm from the stereo system the disparity value is more than 100 pixels. The disparity value reduces drastically as depth reaches approximately 6000 mm.

In Vision, maximum disparity indicates maximum difference between conjugate points in rectified stereo images. The horopter, or range of depth values, is limited by the number of disparities you specify when computing a disparity map. For example, if the number of disparities is set to 32 with a minimum disparity of 0, the horopter extends from approximately 3400 mm to infinity.

To estimate the depth of objects you want to inspect, make sure that the objects are within the horopter. In most cases, this involves specifying a minimum disparity value of 0 and specifying a number of disparities large enough to include the closest object.

Without increasing the number of disparities, the size of the horopter can only be increased by decreasing the baseline, decreasing the focal length, or increasing the pixel width of the sensor. Any of these changes cause an undesirable increase in depth resolution. If you increase the size of the baseline, ensure that the object under inspection remains within overlapping regions of interest. If you decrease the focal length, ensure that the object of interest remains in focus to ensure accurate matching.

Related concepts:

- [Stereo Vision](#)

Stereo Vision Concepts

This section explains the basic principles of binocular stereo vision system for a simplified set-up. A typical stereo vision system incorporates the following steps in order to compute 3D information:

1. Perform camera model calibration for each camera. A camera model calibration learns internal and external parameters for each camera setup. Camera model calibration allows you to subsequently perform image correction to remove lens distortion and produce undistorted images.
2. Perform stereo calibration for the stereo vision system. Stereo calibration computes the relative spatial relationship between two cameras.
3. Perform stereo image rectification. Stereo image rectification projects images acquired from the left and right cameras so that the images reside in the same plane. The rows of rectified images align perfectly so that a point in the left image falls on the same row in both left and right images.
4. Compute stereo image correspondence. Stereo correspondence establishes matches between the left and right rectified images to produce a disparity map. A disparity map is a 2D image that uses grayscale values to indicate the disparity, or distance, between features in the left and right image. Because disparity values indicate the relative depth of an object, a disparity map is sufficient for many stereo vision applications.
5. Optionally compute 3D planes for applications that require precise depth information. 3D planes provide detailed depth information which can be mapped to real-world coordinates.



Note Vision renders depth and disparity maps with respect to the left rectified image.

For best results, Vision prefers a horizontal baseline, or a system where the horizontal distance exceeds the vertical distance between the two cameras.

Related concepts:

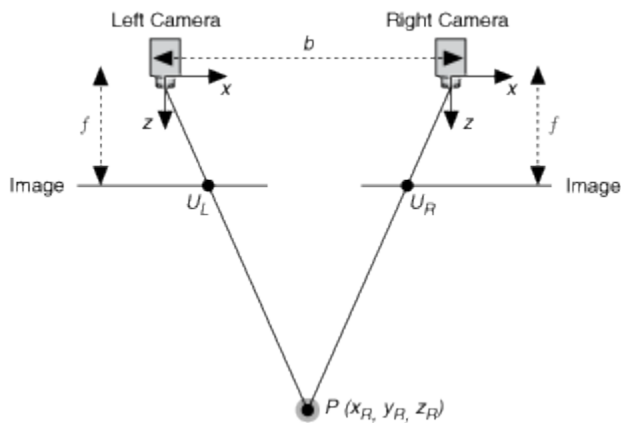
- [Stereo Vision](#)

Parts of a Stereo Vision System

This topic describes the elements of a stereo vision system.

The following figure illustrates a simple stereo vision system, which incorporates the following assumptions:

- Both cameras have the same focal length,
- The two cameras are parallel to each other,
- The X-axes of the two cameras intersect and align with the baseline,
- The origin of the real-world coordinate system coincides with the origin of the left camera coordinate system:



where,

b is the baseline, distance between the two cameras,

f is the focal length of a camera,

x is the X-axis of a camera,

z is the optical axis of a camera,

P is a real-world point defined by the coordinates X , Y , and Z ,

U_L is the coordinate of the real-world point P in an image acquired by the left camera,

U_R is the coordinate of the real-world point P in an image acquired by the right camera.

The X-coordinates of points obtained by projecting point P on the two camera image-planes are given by u_L and u_R :

$$U_L = f_z^x$$

$$U_R = f_z^{x-b}$$

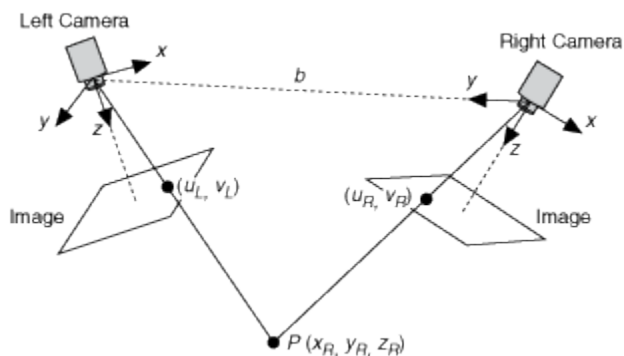
The tuple (U_L, U_R) is known as a correspondence and related projections U_L and U_R are known as conjugate or homologous points. The distance between conjugate points is referred to as disparity (d), which is calculated using the following equation:

$$d = (U_L - U_R) = f_z^b$$

Given a pair of conjugate points, the real-world distance of the original point from the stereo vision system can be calculated as follows:

$$z = f_z^b/d$$

The following figure illustrates a typical stereo vision system:



Most of the assumptions made for the simplified stereo vision system cannot be made for typical stereo vision applications. To compensate, a typical stereo vision system requires stereo calibration.

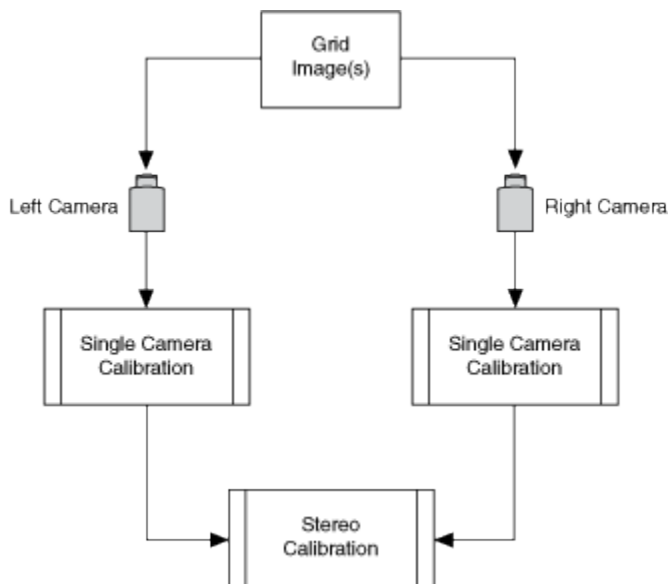
Related concepts:

- [Stereo Vision Concepts](#)

Stereo Calibration

A stereo vision system requires camera model calibration for each camera, followed by stereo calibration. Stereo calibration data allows Vision to produce rectified images, and to calculate disparity and depth information.

The following figure illustrates the calibration process:



Single-camera calibration computes the following parameters:

- **Internal Parameters**—Distortion model and coefficients, focal length, and optical center for each camera.
- **External Parameters**—Rotation and translation matrices between the corresponding camera-coordinate system and the real-world coordinate system.

You must perform stereo calibration to calculate the pose of each camera relative to the other. During stereo calibration, both cameras must view the same calibration grid. For the first frame, the entire calibration grid should be within the field of view for each camera. For subsequent frames, partial coverage of the calibration grid is acceptable. Stereo calibration produces the following matrices:

- **Rotation Matrix (R)**—A rotation matrix, which denotes rotation between left to right camera-coordinate systems.
- **Translation Matrix (T)**—A translation matrix, which denotes translation between

left to right camera-coordinate systems.

- **Essential Matrix (E)**—A geometrical matrix, which relates the location of a point seen by the left camera to the same point as seen by the right camera, in real-world coordinates.
- **Fundamental Matrix (F)**—A geometrical matrix, which relates the location of a point seen by the left camera to the same point as seen by the right camera, in pixel coordinates.

If a camera changes position or focal length, you must repeat camera model calibration for the camera and repeat stereo calibration for the entire stereo vision system.

Related concepts:

- [Stereo Vision Concepts](#)

Maximum Projection Error and Calibration Quality Metric

Vision provides a maximum projection error and a calibration quality metric for stereo calibration.

The maximum projection error indicates the maximum error obtained by projecting a left camera coordinate onto the right camera coordinate.

The calibration quality metric indicates the quality of the stereo calibration based upon an average root mean square error. The calibration quality metric is a value between 0 to 1, with 1 indicating the best calibration. If the calibration quality metric is less than 0.7, consider repeating the calibration process.

Related concepts:

- [Stereo Calibration](#)

Stereo Image Rectification

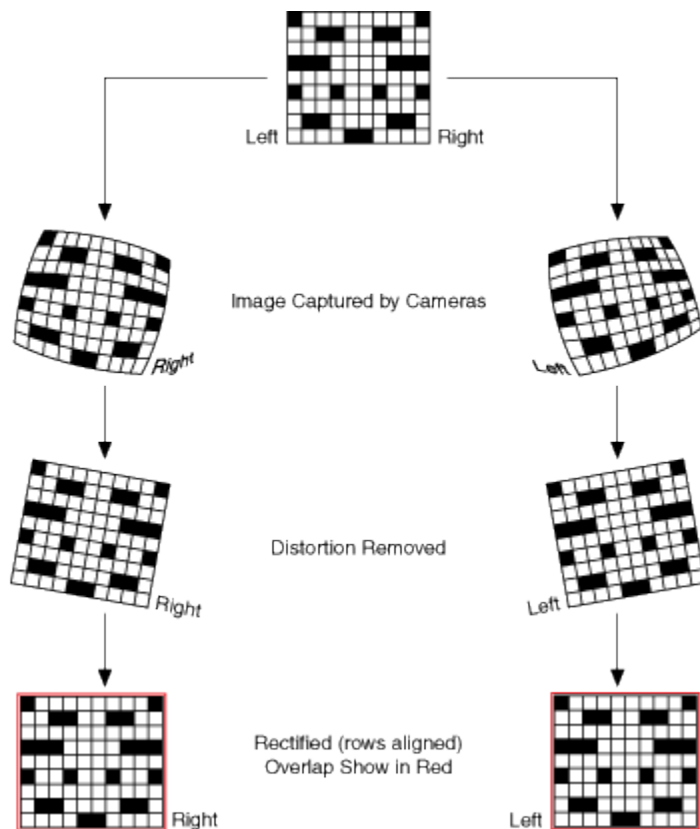
Stereo image rectification projects images acquired from the left and right cameras so that the images reside in the same plane. The rows of rectified images align perfectly so that a point falls on the same row in both in the left and the right images.

By row-aligning matches, stereo image rectification simplifies the processes of calculating stereo correspondence and computing a disparity map. Stereo image rectification is based upon the spatial relationship between the cameras, which is obtained from information produced during stereo calibration.

To speed up the rectification process, a look-up table can be computed for each camera to interpolate points from the original image and create a new rectified image. Because learning a look-up table is memory intensive, the process is optional in Vision.

Vision also accepts a scale parameter. Valid values are 0-1. A value of 1 constrains the rectified images to the original image size. Values smaller than 1 increase the scale of the rectified images.

The following figure illustrates the process of image rectification:



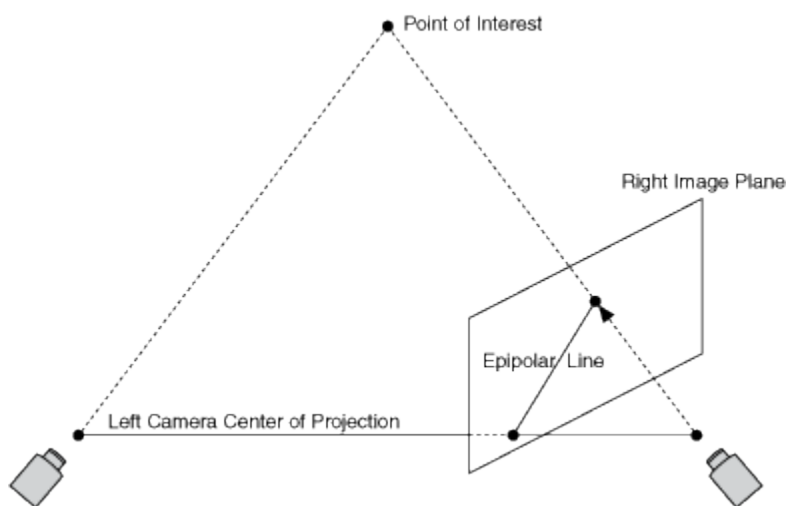
Related concepts:

- [Stereo Vision Concepts](#)

Maximum Rectification Error and Rectification Quality Metric

Vision provides a maximum rectification error and a rectification quality metric to indicate the quality of rectified images.

These parameters are computed using epipolar lines, which are obtained with the fundamental matrix produced during stereo calibration. For example, an epipolar line for an image obtained with the right camera describes the 3D vector of a point relative to the center of projection for the left camera. A valid match in the right camera image must lie on the epipolar line. The following figure illustrates an epipolar line in the right image plane:



The maximum rectification error indicates the greatest distance between a point and its corresponding epipolar line. If this error is greater than 1 consider repeating the calibration process.

The rectification quality metric is a value between 0 and 1, with 1 indicating perfectly aligned rectified images. If the rectification quality metric is less than 0.7, consider repeating the calibration process.

Related concepts:

- [Stereo Image Rectification](#)

Stereo Image Correspondence

Stereo correspondence establishes matches between the left and right rectified images to produce a disparity map.

A disparity map is a 2D image that uses grayscale values to indicate the disparity, or distance, between features in left and right rectified images. The grayscale value of each pixel in the disparity map indicates the distance of that point from the stereo vision system. For better display and resolution, NI Vision multiplies disparity by a factor of 16. Brighter pixels indicate points that are closer to the camera and darker pixels indicate points that are farther away from the camera. A disparity map can be directly used for many applications, including the following:

- Recognizing the relative distance of objects from the imaging system. For example, in an application that must pick up the closest object, it may be possible to segment the disparity map and identify the target object.
- Tracking an object of interest across a sequence of images. Instead of using user-specified point information to locate and track objects, you can segment a disparity map to identify objects and track the objects across a sequence of images.
- Modeling object 3D information, which can be computed from the disparity map.
- Path planning, using 3D information to locate obstacles and open areas.

Vision provides two algorithms for establishing stereo correspondence: a block-matching algorithm⁶ and a semi-global algorithm⁷⁸. The block-matching algorithm offers efficient performance while the semi-global algorithm provides a dense disparity map and works in regions with little or no apparent texture.

Vision provides options to compute disparity to sub-pixel accuracy and to specify a maximum difference between a valid disparity value for a pixel compared to the disparity values of pixels immediately to the left and right. When Vision cannot determine the disparity for a point, the point is set to the value specified by the user to

6. See K. Konolige, Small Vision System: Hardware and implementation, Proceedings of the International Symposium on Robotics Research, pp. 111-116, Hayama, Japan 1997.

7. See H. Hirschmuller, Stereo Processing by Semi-Global Matching and Mutual Information, IEEE Transactions on PAMI, Vol. 30 (2), pp. 328-341, 2008.

8. See S. Birchfield and C. Tomasi, Depth Discontinuities by Pixel-to-Pixel Stereo, IJCV, vol. 35(3), pp. 269-293, 1999.

represent invalid pixels.

Related concepts:

- [Stereo Vision Concepts](#)

Confidence Score Image

Vision provides a confidence score image, which indicates the confidence of the disparity for each pixel. Score images return values between 0-1000, where 1000 indicates the highest confidence. The block-matching algorithm computes confidence score based on how distinctive the match is compared to the second best match, and based on the similarity of the disparity to neighboring disparities. For the semi-global matching algorithm, confidence scores are based entirely on sum of absolute difference values for the match.

Confidence score functions are logistic functions, which vary slowly over a range before falling suddenly in unacceptable regions.

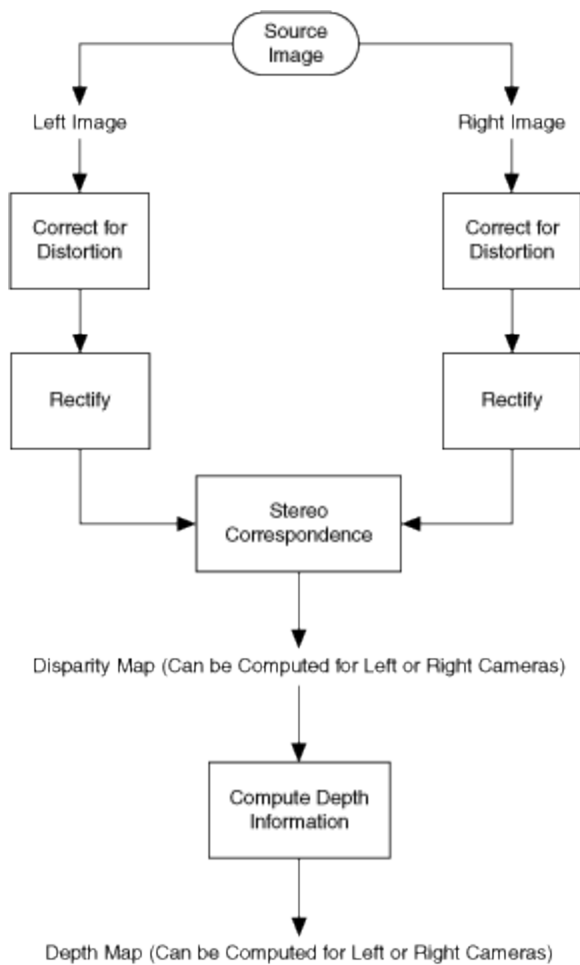
Related concepts:

- [Stereo Image Correspondence](#)

Depth Computation

Some applications might require only a disparity map. Other applications require precise depth information. Examples of such applications are bin-picking, de-palletizing, 3D matching, and 3D measurements. For applications that require precise depth information, 3D planes can be computed following stereo calibration and stereo image rectification.

The following figure illustrates the complete 3D reconstruction process:



3D planes provide comprehensive real-world information about the scene, which can optionally be rendered to a real-world coordinate system. All real-world coordinates use the unit of measure specified during single-camera calibration.

Origin for the X and Y planes is the optical center. Depth and disparity coordinates, provided in the Z plane, are relative to the left camera.

Vision provides an error map for depth computation, which indicates the inherent error in measurement for each pixel.

Related concepts:

- [Stereo Vision Concepts](#)

In-Depth Discussion of Stereo Vision Concepts

This section provides in-depth explanations of stereo vision concepts and the Vision implementation of stereo vision.

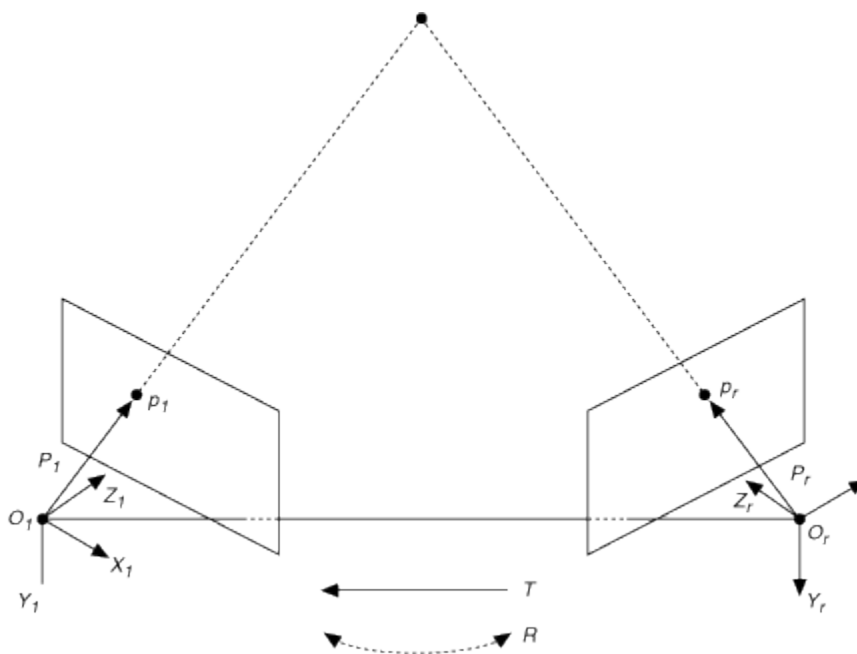
Related concepts:

- [Stereo Vision Concepts](#)

Stereo Calibration In-Depth

You must perform camera model calibration for each camera in the system before performing stereo calibration. Creating a camera model involves acquiring multiple images, usually of a calibration grid, in multiple planes.

For each plane, a camera model provides a set which consists of rotation and translation matrices. Corresponding sets, computed for the left and right camera based on the same plane, provide the information required to compute the spatial relationship between the two cameras. Stereo calibration returns a single rotation and translation matrix (R , T) that relates relative real-world coordinates for the left and right cameras. The following figure illustrates a calibrated stereo vision system:



where:

- O_l is the origin of the coordinate systems centered at the left camera principal point,
- O_r is the origin of the coordinate systems centered at the right camera principal point,
- P is a real-world point being imaged by both cameras, in real-world coordinates,
- p_l is the projection of P on the left-camera image plane,
- p_r is the projection of P on the right-camera image plane.

A camera model provides enough information to describe the relationship of the camera relative to a point (P) under inspection. Let, (R_l, T_l) and (R_r, T_r) be the rotation and translation matrices for the left and right cameras for the plane in which P lies. The real-world coordinates of the point P relative to the left and right cameras are given by the following equations:

$$P_l = R_l P + T_l$$

$$P_r = R_r P + T_r$$

The 3D coordinates of P are given by the following equation:

$$P_r = R T (P_r - T)$$

The stereo translation matrix is given by the following equation:

$$R = R_r (R_l)^T$$

After each camera is calibrated, the stereo vision system must be calibrated. Stereo calibration computes the essential matrix (E) and the fundamental matrix (F). The essential matrix (E) contains the rotation and translation information required to relate the location of a point (P_l) as seen by the left camera to the same point (P_r) as seen by the right camera, in real-world coordinates. Assuming the relationship $P_r = R(P_l - T)$, the relationship between points relative to the left and right cameras and the essential matrix is given by the following equation:

$$P_r^T \cdot E \cdot P_l = 0$$

The essential matrix does not contain information about the internal parameters of the cameras; therefore, it cannot be used to correlate pixel coordinates for conjugate

points. You must use the fundamental matrix F to relate pixel coordinates for conjugate points. The fundamental matrix (F) includes internal parameters for both cameras. Give a pixel point in the left image (q_l), a conjugate pixel point in the right image (q_r), and the fundamental matrix (F), you can compute the corresponding epipolar line in the right image using the following equation:

$$q_r^T \cdot F \cdot q_l = 0$$

Related concepts:

- [In-Depth Discussion of Stereo Vision Concepts](#)

Stereo Image Rectification In-Depth

Given the rotation matrix and translation vector between the two stereo images, NI Vision attempts to limit the amount of change that a new projection produces for each of the two images in order to reduce the resulting distortion while maximizing the common viewing area⁹. Image rectification is provided by the following rotation matrices:

$$R_1 = R_{epi} r_l$$

$$R_2 = R_{epi} r_r$$

where:

- R_1 is the rotation to be applied to the left image to get the left rectified image,
- R_2 is the rotation to be applied to the right image to get the right rectified image,
- R_{epi} is the rotation matrix which row-aligns coplanar images obtained by r_l and r_r ,
- r_l is the rotation matrix required to make left image coplanar with the right image rotated by matrix r_r ,
- R_r is the rotation matrix required to make right image coplanar with the left image rotated by matrix r_l .

Matrices r_l and r_r are obtained by applying half a rotation clockwise and counterclockwise, respectively, so that

9. Vision uses the algorithm proposed by Bouguet for obtaining rectified images. See G. Bradski and A. Kaebler, Learning OpenCV - Computer Vision with the OpenCV Library, First Edition, O'Reilly Media, 2008.

$$R = r_l^T r_r$$

. Matrix $\mathbf{R_{EPI}} = [R_1 \ R_2 \ R_3]^T$. Matrix $\mathbf{R_{EPI}}$ produces the left epipole, or the projection of the principal point of the right camera onto the left image plane, projected to infinity. Using the principal point of the left camera as the origin, translation matrix \mathbf{T} provides the direction of the left epipole on the image plane. Consequently, specific matrices r_1 , r_2 , r_3 are given by the following equations:

$$r_1 = \frac{T}{(T)}$$

$$r_1 = \frac{(-T_y T_x 0)}{\sqrt{T_x^2 + T_y^2}}$$

$$r_3 = r_1 \times r_2$$

Related concepts:

- [In-Depth Discussion of Stereo Vision Concepts](#)

Stereo Image Correspondence In-Depth

A typical correspondence algorithm consists of following three stages:

1. Pre-filtering to normalize the image brightness and enhance texture,
2. Matching points along horizontal lines in local windows,
3. Post-filtering to eliminate bad matches.

Related concepts:

- [In-Depth Discussion of Stereo Vision Concepts](#)

Pre-Filtering for Stereo Image Correspondence

Pre-filtering is a preparatory step for the block-matching algorithm. The semi-global algorithm operates on the original rectified stereo images to establish a stereo correspondence. Vision provides two pre-filtering options: a Sobel filter and a normalized response filter. To prevent horizontal lines, which may mislead the

matching algorithm, the Sobel filter uses only the following kernel:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

The Sobel filter output is given in the following equation: $\text{Min}(\text{Max}(I_{\text{Sobel}}, -I_{\text{cap}}), I_{\text{cap}})$.

Where:

- I_{Sobel} is the value obtained by applying the kernel, accumulated in a window of a user-defined size.
- I_{cap} is a positive number that limits the final pixel value.

The normalized response filter computes the filter response through the following equation: $\text{Min}(\text{Max}(I_{\text{center}}, -I_{\text{avg}} - I_{\text{cap}}), I_{\text{cap}})$

where,

- I_{center} is the pixel value at the point, accumulated over a window of a user-defined size,
- I_{avg} is the average of the pixel computed through 4-neighbors of the center pixel and accumulated over a specified window size,
- I_{cap} is a positive number which limits the final pixel value.

Related concepts:

- [Stereo Image Correspondence In-Depth](#)

Block Matching Algorithm

The block-matching algorithm establishes a correspondence by computing the sum of absolute differences (SAD) computed between small windows in rectified, pre-filtered stereo images.

Matching is constrained by minimum disparity and number of disparity parameters. For example, for a given point (x_l, y) in the left image, the search in the right image is restricted to points that lie within $(x_l - \text{minimum disparity}, y)$ and $(x_l + \text{minimum disparity}, y)$.

disparity - number of disparities, y).

The algorithm computes SAD values for each point in the specified range and selects the location in the right image with the smallest SAD value as the match. The complexity of this algorithm is given in the following equation:

$$O(w \cdot h \cdot n)$$

where:

- w is the width of the rectified image,
- h is the height of the rectified image,
- n is the number of disparities.

Related concepts:

- [Stereo Image Correspondence In-Depth](#)

Semi-Global Matching Algorithm

This algorithm aims to minimize the following global energy function, E , for disparity image, D .

$$E(D) = \sum_p \left(C(p, D_p) + \sum_{q \in N_p} P_1 I(|D_p - D_q| = 1) + \sum_{q \in N_p} P_2 I(|D_p - D_q| > 1) \right)$$

with $P_2 \geq P_1$

where:

- $E(D)$ is the energy for disparity image D ,
- p, q represent indices for pixels in the image,
- N_p is the neighborhood of the pixel p ,
- $C(p, D_p)$ is the cost of pixel matching with disparity in D_p ,
- P_1 is the penalty passed by the user for change in disparity between neighboring pixels by 1,

- P_2 is the penalty passed by the user for change in disparity between neighboring pixels by value greater than 1,
- $I[.]$ is the function which returns 1 if argument is true, 0 otherwise the minimized function produces a perfect disparity map with smoothing governed by parameters P_1 and P_2 ; however, minimizing the function for a 2D image space is a NP-complete problem.

The semi-global matching function approximates the 2D minimization by performing multiple 1D, or linear, minimizations. The matching function aggregates costs on multiple paths which converge on the pixel under examination. Cost is computed for the disparity range specified by the minimum disparity and number of disparities parameters. By default, the matching algorithm aggregates costs for 5 directions. You can set the full dynamic programming parameter to true to force the algorithm to aggregate costs for 8 directions. Let, $S(p, d)$ be the aggregate cost for pixel p and disparity d .

Then

$$S(p, d) = \sum_r L_r(p, d)$$

where r is a direction used for converging to the pixel p $L_r(p, d)$ is the minimum cost of the path taken in direction r from pixel p for disparity d The cost $L_r(p, d)$ is given in the following equation:

$$L_r(p, d) = C(p, d) + \min \left(L_r(p-r, d), L_r(p-r, d-1) + P_1, L_r(p-r, d+1) + P_1, \min_i L_r(p-r, i) + P_2 \right) - \min_k L_r(p-r, k)$$

The equation uses the following costs to find the disparity by adding current cost, $C(p, d)$, to previous pixel in direction r :

- The minimum of the cost at previous pixel with disparity d ,
- The cost at previous pixel with disparity $d - 1$ and $d + 1$ with added penalty P_1 ,
- The cost at previous pixel with disparities less than $d - 1$ and greater than $d + 1$ with added penalty P_2 .

In order to limit the ever increasing value of $L_r(p, d)$ on the path, minimum value of the previous pixel is subtracted. The upper value of $L_r(p, d)$ is bounded by $C_{\max} + P_2$, where

C_{\max} is the maximum value of cost C . The cost function $C(p, d)$ is computed in the following manner:

$$C(p, d) = \min(d(p, p-d, I_L, I_R), d(p-d, p, I_R, I_L))$$

where I_L and I_R are left and right rectified images, respectively

$$d(p, p-d, I_L, I_R) = \min_{p-d-0.5 \leq q \leq p-d+0.5} \|I_L(p) - I_L(q)\|$$

The value of C is aggregated over a window of a user-defined size¹⁰. After computing $S(p, d)$ for each pixel p for each disparity d , the algorithm chooses the disparity which provides the minimum cost for that pixel.

The complexity of this algorithm is given in the following equation:

$$O(w \cdot h \cdot n)$$

where:

- w equals the width of the rectified image,
- h equals the height of the rectified image,
- n equals the number of disparities.

Related concepts:

- [Stereo Image Correspondence In-Depth](#)

Post-Filtering for Stereo Image Correspondence

Post-filtering sets noise pixels to the value specified by the user to represent invalid pixels. Post-filtering consists of two steps. First, pixels that do not meet the specified uniqueness ratio and texture threshold are removed. Then a speckle filter is applied.

10. Birchfield and C. Tomasi, Depth Discontinuities by Pixel-to-Pixel Stereo, IJCV, vol. 35(3), pp. 269-293, 1999.

The uniqueness ratio specifies how unique the disparity of a valid match must be at each pixel relative to all other disparities. Valid values are 0-100, where a value of 0 causes the uniqueness ratio to have no effect.

The texture threshold specifies the minimum sum of absolute difference for a valid match at each pixel. A value of 0 causes the texture threshold to have no effect. A value that is too large will cause each point in the image to be rejected.

The speckle filter examines a user-defined window around each pixel and rejects pixels outside the user-specified speckle range, or range of disparities within the speckle window.

Vision also provides an option to interpolate the disparity map using polynomial interpolation. Interpolation allows pixels to be set to logical approximate values in cases where the stereo correspondence algorithm cannot determine a disparity value.

Related concepts:

- [Stereo Image Correspondence In-Depth](#)

Depth Computation In-Depth

Using a particular disparity value at a given pixel, 3D information can be computed in the following manner:

$$Q \begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix}$$

Q is given as:

$$Q = \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & -1/T_x & (c_x - c'_x)/T_x \end{bmatrix}$$

where:

- d is the disparity at a point (x, y) in the left rectified image,
- (c_x, c_y) represents the optical center in the left rectified image,

- f is the focal length in the left rectified image,
- T_x is the X-component of the translation parameter,
- c'_x is the X-coordinate of the right rectified image.

The 3D measurements can then be given by $(X/W, Y/W, Z/W)$, which are X, Y and Z real-world coordinates, respectively. By default, Vision renders 3D information with respect to the left rectified image such that the new optical center will be at $(0, 0, Z)$ position. Vision sets 3D information for a pixel to not a number (NaN) if disparity cannot be determined.

Related concepts:

- [In-Depth Discussion of Stereo Vision Concepts](#)

Error Mapping for Depth Computation

For 3D measurements, Vision computes error (e_p) for a given pixel (p) according to the following equation:

$$e_p = \frac{1}{2} \left(\frac{fT_x}{(c_x - c'_x) - (d-1)} - \frac{fT_x}{(c_x - c'_x) - (d+1)} \right)$$

This equation computes and averages the interval between the depth achieved from the previous disparity and the depth achieved from the next disparity. Consequently, distant objects produce larger errors than near objects.

Related concepts:

- [Depth Computation In-Depth](#)

Feature Detection and Matching

This section contains information about feature detection and matching.

Introduction to Feature Detection and Matching

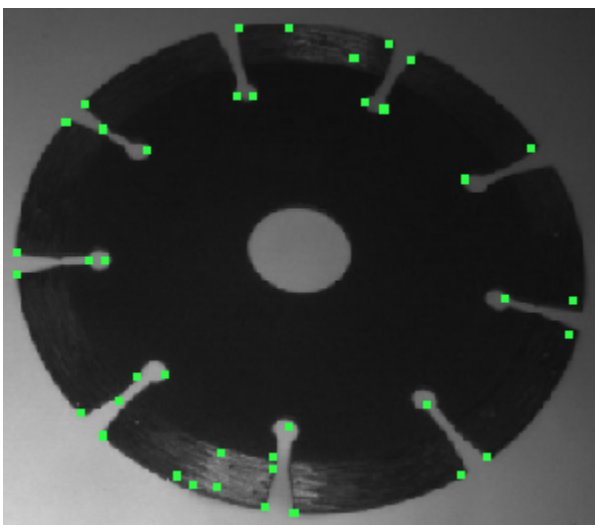
Feature detection and matching are an integral part of computer vision, and are fundamental for many applications. Finding feature correspondence between images is a fundamental part of many applications. Feature detection and matching is comprised of three stages.

1. Feature detection using one of the following methods: Features from Accelerated Segment Test (FAST) Detector, Harris Corner Detector, or Shi-Tomasi Corner Detector.
2. Feature description using one of the following methods: Binary Robust Invariant Scalable Keypoints Descriptor (BRISK) or Fast Retina Keypoint Descriptor (FREAK).
3. Feature matching.

When to Use Feature Detection and Matching

Feature detection, which is also known as corner detection, is useful in applications such as:

- Finding defects using missing corners,
- Shape fitting using a mathematical fit on detected corners,
- Using FAST feature points with optical flow,
- Analyzing an image using detected feature point strengths, for example, while auto-focusing.



Feature Matching can be used in:

- Object (or template) matching between images invariant to gradual changes in rotation, scaling, and perspective transformations,
- Finding correspondence between two images which can be used in many applications, such as stitching,
- Object recognition and pose detection,
- Establishing transformation between two images.

Feature Detection and Matching Concepts

The concept of feature detection refers to methods that aim at computing abstractions of image information. There is no universal or exact definition of what constitutes a feature, and the exact definition often depends on the problem or the type of application. A feature is defined as an interesting part of an image, and features are used as a starting point for many computer vision algorithms. The detector selects points that can be consistently detected across various transformations (blurring, rotation, and scale). Once features have been detected, the features are described mathematically. The result is a feature descriptor. This information can be used to find the matches between the two images.

Feature Detection

Corner Detector: A corner can be defined as the intersection of two edges. A corner can also be defined as a point for which, there are two dominant and different edge directions in a local neighborhood of the point. An interest point is a point in an image which has a well-defined position and can be robustly detected. This method detects all the corners in the image.

- Calculate the image gradients.
- Calculate the Eigen values (λ_1 and λ_2). There are three cases to be considered:
 - If both eigen values are small, the image region is of constant intensity,
 - If one eigen value is high and the other low, it indicates an edge,
 - If both eigen values are high, it indicates a corner.

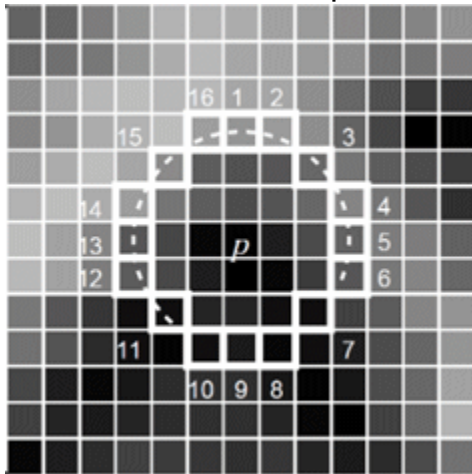
The Harris corner detector and Shi-Tomasi corner detector are similar except for the determination of a good corner.

- Harris Corner: Corner Score = $\lambda_1 \lambda_2 - k (\lambda_1 + \lambda_2)^2$.
- Shi-Tomasi Corner: Corner Score = $\min(\lambda_1, \lambda_2)$.

FAST Feature Point Detector: FAST is an algorithm for identifying all the interest points in an image, not just corners. Interest points have high local information content and ideally should be repeatable between different images.

For a pixel p of intensity I_p .

- Choose a circle of 16 pixels around it, as shown in the following image:



- For a pixel to be a feature, at least N ($N=12$) contiguous pixels should have Intensity $> (I_p + T)$ or $(I_p - T)$, where T is a threshold intensity value.

The detectors can be used for grayscale (U8, U16, and I16) images. Rectangle and Rotated Rectangle are the supported ROI types.

Feature Descriptors

BRISK¹¹ (Binary Robust Invariant Scalable Keypoints) and FREAK¹² (Fast Retina Keypoint) are both binary descriptors that provide information about the feature point.

BRISK is a 512-bit binary descriptor that computes the weighted Gaussian average

11. For more information about the BRISK descriptor, see S. Leutenegger, M. Chli, and R. Siegwart. Brisk: Binary robust invariant scalable keypoints, 2011.
12. For more information about the FREAK descriptor, see A. Alahi, R. Ortiz, P. Vandergheynst. FREAK: Fast Retina Keypoint, 2010.

over a select pattern of points near the keypoint. It compares the values of specific pairs of Gaussian windows, leading to either a 1 or a 0, depending on which window in the pair was greater, which creates binary descriptors.

FREAK is also a binary descriptor that improves upon the sampling pattern and method of pair selection that BRISK uses, but the pattern used in this method is inspired by the retinal pattern in the eye. FREAK provides better rotation invariance, while BRISK provides better matches to changes in perspective distortion. Both provide similar results to changes in scale.

Feature Matching

Given a feature in Image 1, the best match needs to be found in Image 2 for feature matching. Because both BRISK and FREAK are binary descriptors, matching these features requires a computation of the hamming distance, with the number of bits different in the two descriptors being a measure of their dissimilarity.

Kernels

A kernel is a structure that represents a pixel and its relationship to its neighbors. This section lists a number of predefined kernels supported by Vision.

Gradient Kernels

The following tables list the predefined gradient kernels.

3 × 3 Kernels

The following tables list the predefined gradient 3 × 3 kernels.

Prewitt Filters

The Prewitt filters have the following kernels. The notations West (W), South (S), East (E), and North (N) indicate which edges of bright regions they outline.

#0 W/Edge <table> <tr><td>-1</td><td>0</td><td>1</td><td></td></tr> <tr><td>-1</td><td>0</td><td>1</td><td></td></tr> <tr><td>-1</td><td>0</td><td>1</td><td></td></tr> </table>	-1	0	1		-1	0	1		-1	0	1		#1 W/Edge <table> <tr><td>-1</td><td>0</td><td>1</td><td></td></tr> <tr><td>-1</td><td>1</td><td>1</td><td></td></tr> <tr><td>-1</td><td>0</td><td>1</td><td></td></tr> </table>	-1	0	1		-1	1	1		-1	0	1		#2 SW/Edge <table> <tr><td>0</td><td>1</td><td>1</td><td></td></tr> <tr><td>-1</td><td>0</td><td>1</td><td></td></tr> <tr><td>-1</td><td>-1</td><td>0</td><td></td></tr> </table>	0	1	1		-1	0	1		-1	-1	0		#3 SW/Edge <table> <tr><td>0</td><td>1</td><td>1</td><td></td></tr> <tr><td>-1</td><td>1</td><td>1</td><td></td></tr> <tr><td>-1</td><td>-1</td><td>0</td><td></td></tr> </table>	0	1	1		-1	1	1		-1	-1	0	
-1	0	1																																																	
-1	0	1																																																	
-1	0	1																																																	
-1	0	1																																																	
-1	1	1																																																	
-1	0	1																																																	
0	1	1																																																	
-1	0	1																																																	
-1	-1	0																																																	
0	1	1																																																	
-1	1	1																																																	
-1	-1	0																																																	
#4 S/Edge <table> <tr><td>1</td><td>1</td><td>1</td><td></td></tr> <tr><td>0</td><td>0</td><td>0</td><td></td></tr> <tr><td>-1</td><td>-1</td><td>-1</td><td></td></tr> </table>	1	1	1		0	0	0		-1	-1	-1		#5 S/Edge <table> <tr><td>1</td><td>1</td><td>1</td><td></td></tr> <tr><td>0</td><td>1</td><td>0</td><td></td></tr> <tr><td>-1</td><td>-1</td><td>-1</td><td></td></tr> </table>	1	1	1		0	1	0		-1	-1	-1		#6 SE/Edge <table> <tr><td>1</td><td>1</td><td>0</td><td></td></tr> <tr><td>1</td><td>0</td><td>-1</td><td></td></tr> <tr><td>0</td><td>-1</td><td>-1</td><td></td></tr> </table>	1	1	0		1	0	-1		0	-1	-1		#6 SE/Edge <table> <tr><td>1</td><td>1</td><td>0</td><td></td></tr> <tr><td>1</td><td>1</td><td>-1</td><td></td></tr> <tr><td>0</td><td>-1</td><td>-1</td><td></td></tr> </table>	1	1	0		1	1	-1		0	-1	-1	
1	1	1																																																	
0	0	0																																																	
-1	-1	-1																																																	
1	1	1																																																	
0	1	0																																																	
-1	-1	-1																																																	
1	1	0																																																	
1	0	-1																																																	
0	-1	-1																																																	
1	1	0																																																	
1	1	-1																																																	
0	-1	-1																																																	
#8 E/Edge	#9 E/Edge	#10 NE/Edge	#11 NE/Edge																																																

<table> <tr><td>1</td><td>0</td><td>-1</td><td></td></tr> <tr><td>1</td><td>0</td><td>-1</td><td></td></tr> <tr><td>1</td><td>0</td><td>-1</td><td></td></tr> </table>	1	0	-1		1	0	-1		1	0	-1		<table> <tr><td>1</td><td>0</td><td>-1</td><td></td></tr> <tr><td>1</td><td>1</td><td>-1</td><td></td></tr> <tr><td>1</td><td>0</td><td>-1</td><td></td></tr> </table>	1	0	-1		1	1	-1		1	0	-1		<table> <tr><td>0</td><td>-1</td><td>-1</td><td></td></tr> <tr><td>1</td><td>0</td><td>-1</td><td></td></tr> <tr><td>1</td><td>1</td><td>0</td><td></td></tr> </table>	0	-1	-1		1	0	-1		1	1	0		<table> <tr><td>0</td><td>-1</td><td>-1</td><td></td></tr> <tr><td>1</td><td>1</td><td>-1</td><td></td></tr> <tr><td>1</td><td>1</td><td>0</td><td></td></tr> </table>	0	-1	-1		1	1	-1		1	1	0	
1	0	-1																																																	
1	0	-1																																																	
1	0	-1																																																	
1	0	-1																																																	
1	1	-1																																																	
1	0	-1																																																	
0	-1	-1																																																	
1	0	-1																																																	
1	1	0																																																	
0	-1	-1																																																	
1	1	-1																																																	
1	1	0																																																	
#12 N/Edge	#13 N/Edge	#14 NW/Edge	#15 NW/Edge																																																
<table> <tr><td>-1</td><td>-1</td><td>-1</td><td></td></tr> <tr><td>0</td><td>0</td><td>0</td><td></td></tr> <tr><td>1</td><td>1</td><td>1</td><td></td></tr> </table>	-1	-1	-1		0	0	0		1	1	1		<table> <tr><td>-1</td><td>-1</td><td>-1</td><td></td></tr> <tr><td>0</td><td>1</td><td>0</td><td></td></tr> <tr><td>1</td><td>1</td><td>1</td><td></td></tr> </table>	-1	-1	-1		0	1	0		1	1	1		<table> <tr><td>-1</td><td>-1</td><td>0</td><td></td></tr> <tr><td>-1</td><td>0</td><td>1</td><td></td></tr> <tr><td>0</td><td>1</td><td>1</td><td></td></tr> </table>	-1	-1	0		-1	0	1		0	1	1		<table> <tr><td>-1</td><td>-1</td><td>0</td><td></td></tr> <tr><td>-1</td><td>1</td><td>1</td><td></td></tr> <tr><td>0</td><td>1</td><td>1</td><td></td></tr> </table>	-1	-1	0		-1	1	1		0	1	1	
-1	-1	-1																																																	
0	0	0																																																	
1	1	1																																																	
-1	-1	-1																																																	
0	1	0																																																	
1	1	1																																																	
-1	-1	0																																																	
-1	0	1																																																	
0	1	1																																																	
-1	-1	0																																																	
-1	1	1																																																	
0	1	1																																																	

Sobel Filters

The Sobel filters are very similar to the Prewitt filters, except that they highlight light intensity variations along a particular axis that is assigned a stronger weight. The Sobel filters have the following kernels. The notations West (W), South (S), East (E), and North (N) indicate which edges of bright regions they outline.

#16 W/Edge	#17 W/Edge	#18 SW/Edge	#19 SW/Edge																																																
<table> <tr><td>-1</td><td>0</td><td>1</td><td></td></tr> <tr><td>-2</td><td>0</td><td>2</td><td></td></tr> <tr><td>-1</td><td>0</td><td>1</td><td></td></tr> </table>	-1	0	1		-2	0	2		-1	0	1		<table> <tr><td>-1</td><td>0</td><td>1</td><td></td></tr> <tr><td>-2</td><td>1</td><td>2</td><td></td></tr> <tr><td>-1</td><td>0</td><td>1</td><td></td></tr> </table>	-1	0	1		-2	1	2		-1	0	1		<table> <tr><td>0</td><td>1</td><td>2</td><td></td></tr> <tr><td>-1</td><td>0</td><td>1</td><td></td></tr> <tr><td>-2</td><td>-1</td><td>0</td><td></td></tr> </table>	0	1	2		-1	0	1		-2	-1	0		<table> <tr><td>0</td><td>1</td><td>2</td><td></td></tr> <tr><td>-1</td><td>1</td><td>1</td><td></td></tr> <tr><td>-2</td><td>-1</td><td>0</td><td></td></tr> </table>	0	1	2		-1	1	1		-2	-1	0	
-1	0	1																																																	
-2	0	2																																																	
-1	0	1																																																	
-1	0	1																																																	
-2	1	2																																																	
-1	0	1																																																	
0	1	2																																																	
-1	0	1																																																	
-2	-1	0																																																	
0	1	2																																																	
-1	1	1																																																	
-2	-1	0																																																	
#20 S/Edge	#21 S/Edge	#22 SE/Edge	#23 SE/Edge																																																
<table> <tr><td>1</td><td>2</td><td>1</td><td></td></tr> <tr><td>0</td><td>0</td><td>0</td><td></td></tr> <tr><td>-1</td><td>-2</td><td>-1</td><td></td></tr> </table>	1	2	1		0	0	0		-1	-2	-1		<table> <tr><td>1</td><td>2</td><td>1</td><td></td></tr> <tr><td>0</td><td>1</td><td>0</td><td></td></tr> <tr><td>-1</td><td>-2</td><td>-1</td><td></td></tr> </table>	1	2	1		0	1	0		-1	-2	-1		<table> <tr><td>2</td><td>1</td><td>0</td><td></td></tr> <tr><td>1</td><td>0</td><td>-1</td><td></td></tr> <tr><td>0</td><td>-1</td><td>-2</td><td></td></tr> </table>	2	1	0		1	0	-1		0	-1	-2		<table> <tr><td>2</td><td>1</td><td>0</td><td></td></tr> <tr><td>1</td><td>1</td><td>-1</td><td></td></tr> <tr><td>0</td><td>-1</td><td>-2</td><td></td></tr> </table>	2	1	0		1	1	-1		0	-1	-2	
1	2	1																																																	
0	0	0																																																	
-1	-2	-1																																																	
1	2	1																																																	
0	1	0																																																	
-1	-2	-1																																																	
2	1	0																																																	
1	0	-1																																																	
0	-1	-2																																																	
2	1	0																																																	
1	1	-1																																																	
0	-1	-2																																																	

#24 E/Edge	#25 E/Edge	#26 NE/Edge	#27 NE/Edge
1 0 -1	1 0 -1	0 -1 -2	0 -1 -2
2 0 -2	2 1 -2	1 0 -1	1 1 -1
1 0 -1	1 0 -1	2 1 0	2 1 0
#28 N/Edge	#29 N/Edge	#30 NW/Edge	#31 NW/Edge
-1 -2 -1	-1 -2 -1	-2 -1 0	-2 -1 0
0 0 0	0 1 0	-1 0 1	-1 1 1
1 2 1	1 2 1	0 1 2	0 1 2

5 × 5 Kernels

The following table lists the predefined gradient 5 × 5 kernels.

#0 W/Edge	#1 W/Edge	#2 SW/Edge	#3 SW/Edge
0 -1 0 1 0	0 -1 0 1 0	0 0 1 1 1	0 0 1 1 1
-1 -2 0 2 1	-1 -2 0 2 1	0 0 2 2 1	0 0 2 2 1
-1 -2 0 2 1	-1 -2 1 2 1	-1 -2 0 2 1	-1 -2 1 2 1
-1 -2 0 2 1	-1 -2 0 2 1	-1 -2 -2 0 0	-1 -2 -2 0 0
0 -1 0 1 0	0 -1 0 1 0	-1 -1 -1 0 0	-1 -1 -1 0 0
#4 S/Edge	#5 S/Edge	#6 SE/Edge	#7 SE/Edge
0 1 1 1 0	0 1 1 1 0	1 1 1 0 0	1 1 1 0 0
1 2 2 2 1	1 2 2 2 1	1 2 2 0 0	1 2 2 0 0
0 0 0 0 0	0 0 1 0 0	1 2 0 -2 -1	1 2 1 -2 -1

<table><tr><td>-1</td><td>-2</td><td>-2</td><td>-2</td><td>-1</td></tr><tr><td>0</td><td>-1</td><td>-1</td><td>-1</td><td>0</td></tr></table>	-1	-2	-2	-2	-1	0	-1	-1	-1	0	<table><tr><td>-1</td><td>-2</td><td>-2</td><td>-2</td><td>-1</td></tr><tr><td>0</td><td>-1</td><td>-1</td><td>-1</td><td>0</td></tr></table>	-1	-2	-2	-2	-1	0	-1	-1	-1	0	<table><tr><td>0</td><td>0</td><td>-2</td><td>-2</td><td>-1</td></tr><tr><td>0</td><td>0</td><td>-1</td><td>-1</td><td>-1</td></tr></table>	0	0	-2	-2	-1	0	0	-1	-1	-1	<table><tr><td>0</td><td>0</td><td>-2</td><td>-2</td><td>-1</td></tr><tr><td>0</td><td>0</td><td>-1</td><td>-1</td><td>-1</td></tr></table>	0	0	-2	-2	-1	0	0	-1	-1	-1																																																												
-1	-2	-2	-2	-1																																																																																																			
0	-1	-1	-1	0																																																																																																			
-1	-2	-2	-2	-1																																																																																																			
0	-1	-1	-1	0																																																																																																			
0	0	-2	-2	-1																																																																																																			
0	0	-1	-1	-1																																																																																																			
0	0	-2	-2	-1																																																																																																			
0	0	-1	-1	-1																																																																																																			
#8 E/Edge	#9 E/Edge	#10 NE/Edge	#11 NE/Edge																																																																																																				
<table><tr><td>0</td><td>1</td><td>0</td><td>-1</td><td>-0</td></tr><tr><td>1</td><td>2</td><td>0</td><td>-2</td><td>-1</td></tr><tr><td>1</td><td>2</td><td>0</td><td>-2</td><td>-1</td></tr><tr><td>1</td><td>2</td><td>0</td><td>-2</td><td>-1</td></tr><tr><td>0</td><td>1</td><td>0</td><td>-1</td><td>-0</td></tr></table>	0	1	0	-1	-0	1	2	0	-2	-1	1	2	0	-2	-1	1	2	0	-2	-1	0	1	0	-1	-0	<table><tr><td>0</td><td>1</td><td>0</td><td>-1</td><td>-0</td></tr><tr><td>1</td><td>2</td><td>0</td><td>-2</td><td>-1</td></tr><tr><td>1</td><td>2</td><td>1</td><td>-2</td><td>-1</td></tr><tr><td>1</td><td>2</td><td>0</td><td>-2</td><td>-1</td></tr><tr><td>0</td><td>1</td><td>0</td><td>-1</td><td>-0</td></tr></table>	0	1	0	-1	-0	1	2	0	-2	-1	1	2	1	-2	-1	1	2	0	-2	-1	0	1	0	-1	-0	<table><tr><td>0</td><td>0</td><td>-1</td><td>-1</td><td>-1</td></tr><tr><td>0</td><td>0</td><td>-2</td><td>2</td><td>-1</td></tr><tr><td>1</td><td>2</td><td>0</td><td>-2</td><td>-1</td></tr><tr><td>1</td><td>2</td><td>2</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td></tr></table>	0	0	-1	-1	-1	0	0	-2	2	-1	1	2	0	-2	-1	1	2	2	0	0	1	1	1	0	0	<table><tr><td>0</td><td>0</td><td>-1</td><td>-1</td><td>-1</td></tr><tr><td>0</td><td>0</td><td>-2</td><td>2</td><td>-1</td></tr><tr><td>1</td><td>2</td><td>1</td><td>-2</td><td>-1</td></tr><tr><td>1</td><td>2</td><td>2</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td></tr></table>	0	0	-1	-1	-1	0	0	-2	2	-1	1	2	1	-2	-1	1	2	2	0	0	1	1	1	0	0
0	1	0	-1	-0																																																																																																			
1	2	0	-2	-1																																																																																																			
1	2	0	-2	-1																																																																																																			
1	2	0	-2	-1																																																																																																			
0	1	0	-1	-0																																																																																																			
0	1	0	-1	-0																																																																																																			
1	2	0	-2	-1																																																																																																			
1	2	1	-2	-1																																																																																																			
1	2	0	-2	-1																																																																																																			
0	1	0	-1	-0																																																																																																			
0	0	-1	-1	-1																																																																																																			
0	0	-2	2	-1																																																																																																			
1	2	0	-2	-1																																																																																																			
1	2	2	0	0																																																																																																			
1	1	1	0	0																																																																																																			
0	0	-1	-1	-1																																																																																																			
0	0	-2	2	-1																																																																																																			
1	2	1	-2	-1																																																																																																			
1	2	2	0	0																																																																																																			
1	1	1	0	0																																																																																																			
#12 N/Edge	#13 N/Edge	#14 NW/Edge	#15 NW/Edge																																																																																																				
<table><tr><td>0</td><td>-1</td><td>-1</td><td>-1</td><td>0</td></tr><tr><td>-1</td><td>-2</td><td>-2</td><td>-2</td><td>-1</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>1</td><td>2</td><td>2</td><td>2</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td></tr></table>	0	-1	-1	-1	0	-1	-2	-2	-2	-1	0	0	0	0	0	1	2	2	2	1	0	1	1	1	0	<table><tr><td>0</td><td>-1</td><td>-1</td><td>-1</td><td>0</td></tr><tr><td>-1</td><td>-2</td><td>-2</td><td>-2</td><td>-1</td></tr><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>2</td><td>2</td><td>2</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td></tr></table>	0	-1	-1	-1	0	-1	-2	-2	-2	-1	0	0	1	0	0	1	2	2	2	1	0	1	1	1	0	<table><tr><td>-1</td><td>-1</td><td>-1</td><td>0</td><td>0</td></tr><tr><td>-1</td><td>-2</td><td>-2</td><td>0</td><td>0</td></tr><tr><td>-1</td><td>-2</td><td>0</td><td>2</td><td>1</td></tr><tr><td>0</td><td>0</td><td>2</td><td>2</td><td>1</td></tr><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td></tr></table>	-1	-1	-1	0	0	-1	-2	-2	0	0	-1	-2	0	2	1	0	0	2	2	1	0	0	1	1	1	<table><tr><td>-1</td><td>-1</td><td>-1</td><td>0</td><td>0</td></tr><tr><td>-1</td><td>-2</td><td>-2</td><td>0</td><td>0</td></tr><tr><td>-1</td><td>-2</td><td>1</td><td>2</td><td>1</td></tr><tr><td>0</td><td>0</td><td>2</td><td>2</td><td>1</td></tr><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td></tr></table>	-1	-1	-1	0	0	-1	-2	-2	0	0	-1	-2	1	2	1	0	0	2	2	1	0	0	1	1	1
0	-1	-1	-1	0																																																																																																			
-1	-2	-2	-2	-1																																																																																																			
0	0	0	0	0																																																																																																			
1	2	2	2	1																																																																																																			
0	1	1	1	0																																																																																																			
0	-1	-1	-1	0																																																																																																			
-1	-2	-2	-2	-1																																																																																																			
0	0	1	0	0																																																																																																			
1	2	2	2	1																																																																																																			
0	1	1	1	0																																																																																																			
-1	-1	-1	0	0																																																																																																			
-1	-2	-2	0	0																																																																																																			
-1	-2	0	2	1																																																																																																			
0	0	2	2	1																																																																																																			
0	0	1	1	1																																																																																																			
-1	-1	-1	0	0																																																																																																			
-1	-2	-2	0	0																																																																																																			
-1	-2	1	2	1																																																																																																			
0	0	2	2	1																																																																																																			
0	0	1	1	1																																																																																																			

7 × 7 Kernels

The following table lists the predefined gradient 7 × 7 kernels.

Table 1.

#0 W/Edge							#1 W/Edge						
0	-1	-1	0	1	1	0	0	-1	-1	0	1	1	0
-1	-2	-2	0	2	2	1	-1	-2	-2	0	2	2	1

-1	-2	-3	0	3	2	1
-1	-2	-3	0	3	2	1
-1	-2	-3	0	3	2	1
-1	-2	-2	0	2	2	1
0	-1	-1	0	1	1	0

#2 S/Edge

0	1	1	1	1	1	0
1	2	2	2	2	2	1
1	2	3	3	3	2	1
0	0	0	0	0	0	0
-1	-2	-3	-3	-3	-2	-1
-1	-2	-2	-2	-2	-2	-1
0	-1	-1	-1	-1	-1	0

#4 E/Edge

0	1	1	0	-1	-1	0
1	2	2	0	-2	-2	-1
1	2	3	0	-3	-2	-1
1	2	3	0	-3	-2	-1
1	2	3	0	-3	-2	-1
1	2	2	0	-2	-2	-1
0	1	1	0	-1	-1	0

#6 N/Edge

-1	-2	-3	0	3	2	1
-1	-2	-3	1	3	2	1
-1	-2	-3	0	3	2	1
-1	-2	-2	0	2	2	1
0	-1	-1	0	1	1	0

#3 S/Edge

0	1	1	1	1	1	0
1	2	2	2	2	2	1
1	2	3	3	3	2	1
0	0	0	1	0	0	0
-1	-2	-3	-3	-3	-2	-1
-1	-2	-2	-2	-2	-2	-1
0	-1	-1	-1	-1	-1	0

#5 E/Edge

0	1	1	0	-1	-1	0
1	2	2	0	-2	-2	-1
1	2	3	0	-3	-2	-1
1	2	3	1	-3	-2	-1
1	2	3	0	-3	-2	-1
1	2	2	0	-2	-2	-1
0	1	1	0	-1	-1	0

#7 N/Edge

0	-1	-1	-1	-1	-1	0	0	-1	-1	-1	-1	-1	-1	0
-1	-2	-2	-2	-2	-2	-1	-1	-2	-2	-2	-2	-2	-2	-1
-1	-2	-3	-3	-3	-2	-1	-1	-2	-3	-3	-3	-2	-1	-1
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	2	3	3	3	2	1	1	2	3	3	3	2	1	1
1	2	2	2	2	2	1	1	2	2	2	2	2	1	1
0	1	1	1	1	1	0	0	1	1	1	1	1	0	0

Laplacian Kernels

The following tables list the predefined Laplacian kernels.

3 × 3 Kernels

#0 Contour 4	#1 +Image×1	#2 +Image×1																											
<table> <tr><td>0</td><td>-1</td><td>0</td></tr> <tr><td>-1</td><td>4</td><td>-1</td></tr> <tr><td>0</td><td>-1</td><td>0</td></tr> </table>	0	-1	0	-1	4	-1	0	-1	0	<table> <tr><td>0</td><td>-1</td><td>0</td></tr> <tr><td>-1</td><td>5</td><td>-1</td></tr> <tr><td>0</td><td>-1</td><td>0</td></tr> </table>	0	-1	0	-1	5	-1	0	-1	0	<table> <tr><td>0</td><td>-1</td><td>0</td></tr> <tr><td>-1</td><td>6</td><td>-1</td></tr> <tr><td>0</td><td>-1</td><td>0</td></tr> </table>	0	-1	0	-1	6	-1	0	-1	0
0	-1	0																											
-1	4	-1																											
0	-1	0																											
0	-1	0																											
-1	5	-1																											
0	-1	0																											
0	-1	0																											
-1	6	-1																											
0	-1	0																											
#3 Contour 8	#4 +Image×1	#5 +Image×2																											
<table> <tr><td>-1</td><td>-1</td><td>-1</td></tr> <tr><td>-1</td><td>8</td><td>-1</td></tr> <tr><td>-1</td><td>-1</td><td>-1</td></tr> </table>	-1	-1	-1	-1	8	-1	-1	-1	-1	<table> <tr><td>-1</td><td>-1</td><td>-1</td></tr> <tr><td>-1</td><td>9</td><td>-1</td></tr> <tr><td>-1</td><td>-1</td><td>-1</td></tr> </table>	-1	-1	-1	-1	9	-1	-1	-1	-1	<table> <tr><td>-1</td><td>-1</td><td>-1</td></tr> <tr><td>-1</td><td>10</td><td>-1</td></tr> <tr><td>-1</td><td>-1</td><td>-1</td></tr> </table>	-1	-1	-1	-1	10	-1	-1	-1	-1
-1	-1	-1																											
-1	8	-1																											
-1	-1	-1																											
-1	-1	-1																											
-1	9	-1																											
-1	-1	-1																											
-1	-1	-1																											
-1	10	-1																											
-1	-1	-1																											
#6 Contour 12	#7 +Image×1																												

-1	-2	-1	-1	-2	-1
-2	12	-2	-2	13	-2
-1	-1	-2	-1	-2	-1

5 × 5 Kernels

#0 Contour 24					#1 +Image×1				
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	24	-1	-1	-1	-1	25	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

7 × 7 Kernels

#0 Contour 48							#1 +Image×1						
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	48	-1	-1	-1	-1	-1	-1	49	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Smoothing Kernels

The following tables list the predefined smoothing kernels.

3 × 3 Kernels

0	1	0		0	1	0		0	2	0		0	4	0	
1	0	1		1	1	1		2	1	2		4	1	4	
0	1	0		0	1	0		0	2	0		0	4	0	
1	1	1		1	1	1		2	2	2		4	4	4	
1	0	1		1	1	1		2	1	2		4	1	4	
1	1	1		1	1	1		2	2	2		4	4	4	

5 × 5 Kernels

Table 2.

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

7 × 7 Kernels

Table 3.

[illegible]

Gaussian Kernels

The following tables list the predefined Gaussian kernels.

3 × 3 Kernels

0	1	0	0	1	0	1	1	1
1	2	1	1	4	1	1	2	1
0	1	0	0	1	0	1	1	1

1	1	1	1	2	1	1	4	1
1	4	1	2	4	2	4	16	4
1	1	1	1	2	1	1	4	1

5 × 5 Kernel

1	2	4	2	1
2	4	8	4	2
4	8	16	8	4
2	4	8	4	2
1	2	4	2	1

7 × 7 Kernel

1	1	2	2	2	1	1
1	2	2	4	2	2	1
2	2	4	8	4	2	2
2	4	8	16	8	4	2
2	2	4	8	4	2	2
1	2	2	4	2	2	1
1	1	2	2	2	1	1