

NI Vision

NI Vision for LabWindows™/CVI™ User Manual

Worldwide Technical Support and Product Information

ni.com

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 683 0100

Worldwide Offices

Australia 1800 300 800, Austria 43 662 457990-0, Belgium 32 (0) 2 757 0020, Brazil 55 11 3262 3599,
Canada 800 433 3488, China 86 21 5050 9800, Czech Republic 420 224 235 774, Denmark 45 45 76 26 00,
Finland 358 (0) 9 725 72511, France 01 57 66 24 24, Germany 49 89 7413130, India 91 80 41190000,
Israel 972 3 6393737, Italy 39 02 41309277, Japan 0120-527196, Korea 82 02 3451 3400,
Lebanon 961 (0) 1 33 28 28, Malaysia 1800 887710, Mexico 01 800 010 0793, Netherlands 31 (0) 348 433 466,
New Zealand 0800 553 322, Norway 47 (0) 66 90 76 60, Poland 48 22 328 90 10, Portugal 351 210 311 210,
Russia 7 495 783 6851, Singapore 1800 226 5886, Slovenia 386 3 425 42 00, South Africa 27 0 11 805 8197,
Spain 34 91 640 0085, Sweden 46 (0) 8 587 895 00, Switzerland 41 56 2005151, Taiwan 886 02 2377 2222,
Thailand 662 278 6777, Turkey 90 212 279 3031, United Kingdom 44 (0) 1635 523545

For further support information, refer to the *Technical Support and Professional Services* appendix. To comment on National Instruments documentation, refer to the National Instruments Web site at ni.com/info and enter the info code `feedback`.

Important Information

Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THEREFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

National Instruments respects the intellectual property of others, and we ask our users to do the same. NI software is protected by copyright and other intellectual property laws. Where NI software may be used to reproduce software or other materials belonging to others, you may use NI software only to reproduce materials that you may reproduce in accordance with the terms of any applicable license or other legal restriction.

Trademarks

National Instruments, NI, ni.com, and LabVIEW are trademarks of National Instruments Corporation. Refer to the *Terms of Use* section on ni.com/legal for more information about National Instruments trademarks.

The mark LabWindows is used under a license from Microsoft Corporation. Windows is a registered trademark of Microsoft Corporation in the United States and other countries. Other product and company names mentioned herein are trademarks or trade names of their respective companies.

Members of the National Instruments Alliance Partner Program are business entities independent from National Instruments and have no agency, partnership, or joint-venture relationship with National Instruments.

Patents

For patents covering National Instruments products/technology, refer to the appropriate location: **Help»Patents** in your software, the `patents.txt` file on your media, or the *National Instruments Patent Notice* at ni.com/patents.

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Contents

About This Manual

Conventions	ix
Related Documentation.....	x
NI Vision	x
NI Vision Assistant.....	x
Other Documentation	x

Chapter 1

Introduction to NI Vision

About NI Vision.....	1-1
Application Development Environments.....	1-1
NI Vision Function Tree	1-2
NI Machine Vision Function Tree	1-4
Creating NI Vision Applications	1-4

Chapter 2

Getting Measurement-Ready Images

Set Up Your Imaging System	2-1
Calibrate Your Imaging System	2-2
Create an Image	2-2
Source and Destination Images	2-4
Acquire or Read an Image	2-5
Acquiring an Image	2-6
Reading a File.....	2-6
Converting an Array to an Image	2-7
Display an Image	2-7
Attach Calibration Information.....	2-8
Analyze an Image	2-8
Improve an Image	2-9
Lookup Tables	2-9
Filters	2-10
Convolution Filter	2-10
Nth Order Filter	2-10
Grayscale Morphology	2-11
FFT	2-11
Complex Image Operations	2-13

Chapter 3

Making Grayscale and Color Measurements

Define Regions of Interest	3-1
Defining Regions Interactively	3-1
Tools Palette Transformation	3-5
Defining Regions Programmatically	3-7
Defining Regions with Masks	3-7
Measure Grayscale Statistics	3-8
Measure Color Statistics	3-8
Comparing Colors	3-10
Learning Color Information	3-10
Specifying the Color Information to Learn	3-11
Choosing a Color Representation Sensitivity	3-13
Ignoring Learned Colors	3-14

Chapter 4

Performing Particle Analysis

Create a Binary Image	4-1
Improve the Binary Image	4-2
Removing Unwanted Particles	4-3
Separating Touching Particles	4-4
Using Watershed Transform	4-4
Using Binary Morphology	4-4
Improving Particle Shapes	4-4
Make Particle Measurements	4-5

Chapter 5

Performing Machine Vision Tasks

Locate Objects to Inspect	5-2
Using Edge Detection to Build a Coordinate Transform	5-3
Using Pattern Matching to Build a Coordinate Transform	5-5
Choosing a Method to Build the Coordinate Transform	5-6
Set Search Areas	5-7
Defining Regions Interactively	5-7
Defining Regions Programmatically	5-8
Find Measurement Points	5-8
Finding Features Using Edge Detection	5-8
Finding Lines or Circles	5-8
Finding Edge Points Along One Search Contour	5-10
Finding Edge Points Along Multiple Search Contours	5-10

Finding Points Using Pattern Matching	5-11
Defining and Creating Effective Template Images.....	5-11
Training the Pattern Matching Algorithm.....	5-13
Defining a Search Area	5-14
Setting Matching Parameters and Tolerances	5-15
Testing the Search Algorithm on Test Images.....	5-16
Using a Ranking Method to Verify Results	5-16
Finding Points Using Geometric Matching	5-17
Defining and Creating Effective Template Images.....	5-18
Training the Geometric Matching Algorithm	5-19
Setting Matching Parameters and Tolerances	5-20
Testing the Search Algorithm on Test Images.....	5-21
Finding Points Using Color Pattern Matching	5-22
Defining and Creating Good Color Template Images	5-23
Training the Color Pattern Matching Algorithm.....	5-24
Defining a Search Area	5-25
Setting Matching Parameters and Tolerances	5-26
Testing the Search Algorithm on Test Images.....	5-28
Finding Points Using Color Location.....	5-28
Convert Pixel Coordinates to Real-World Coordinates.....	5-29
Make Measurements	5-29
Distance Measurements.....	5-29
Analytic Geometry Measurements	5-30
Instrument Reader Measurements	5-30
Identify Parts Under Inspection	5-31
Classifying Samples	5-31
Color Classification.....	5-31
Particle Classification	5-31
Performing Classification	5-32
Reading Characters.....	5-33
Reading Barcodes	5-33
Reading 1D Barcodes.....	5-33
Reading Data Matrix Codes	5-34
Reading QR Codes.....	5-34
Reading PDF417 Codes	5-34
Inspect Image for Defects	5-35
Compare to Golden Template	5-35
Verify Characters.....	5-35
Display Results	5-36

Chapter 6

Calibrating Images

Perspective and Nonlinear Distortion Calibration.....	6-1
Defining a Calibration Template.....	6-2
Defining a Reference Coordinate System.....	6-3
Learning Calibration Information.....	6-5
Specifying Scaling Factors.....	6-6
Choosing a Region of Interest.....	6-6
Choosing a Learning Algorithm.....	6-6
Using the Learning Score.....	6-7
Learning the Error Map.....	6-8
Learning the Correction Table.....	6-8
Setting the Scaling Method.....	6-8
Calibration Invalidation.....	6-8
Simple Calibration.....	6-9
Save Calibration Information.....	6-10
Attach Calibration Information.....	6-10

Appendix A

Technical Support and Professional Services

Glossary

Index

About This Manual

This document is intended for engineers and scientists who have knowledge of the LabWindows™/CVI™ programming environment and need to create machine vision and image processing applications using C functions. The manual guides you through tasks beginning with setting up your imaging system to taking measurements.

Conventions

The following conventions are used in this manual:

»

The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.



This icon denotes a tip, which alerts you to advisory information.



This icon denotes a note, which alerts you to important information.

bold

Bold text denotes items that you must select or click in the software, such as menu items and dialog box options. Bold text also denotes parameter names.

italic

Italic text denotes variables, emphasis, a cross-reference, or an introduction to a key concept. Italic text also denotes text that is a placeholder for a word or value that you must supply.

`monospace`

Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames, and extensions.

Related Documentation

In addition to this manual, the following documentation resources are available to help you create your vision application.

NI Vision

- *NI Vision Concepts Help*—Describes the basic concepts of image analysis, image processing, and machine vision. This document also contains in-depth discussions about imaging functions for advanced users.
- *NI Vision for LabWindows/CVI Function Reference Help*—Contains reference information about NI Vision for LabWindows/CVI functions.
- Example programs—Illustrate common applications you can create with Vision for LabWindows/CVI. You can find these examples in the <CVI>\samples\vision directory.
- Application Notes—Contain information about advanced NI Vision concepts and applications. Application Notes are located on the National Instruments Web site at ni.com/appnotes.nsf.

NI Vision Assistant

- *NI Vision Assistant Tutorial*—Describes the NI Vision Assistant software interface and guides you through creating example image processing and machine vision applications.
- *NI Vision Assistant Help*—Contains descriptions of the NI Vision Assistant features and functions and provides instructions for using them.

Other Documentation

- National Instruments image acquisition device user manuals—Contain installation instructions and device-specific information.
- *NI-IMAQ Help*—Contains fundamental programming concepts for NI-IMAQ driver software and terminology for using NI image acquisition devices.
- *NI-IMAQ Function Reference Help*—Contains reference information about the LabWindows/CVI functions for NI-IMAQ driver software.

- *NI-IMAQdx Help*—Contains fundamental programming concepts for NI-IMAQdx driver software and terminology for using NI image acquisition devices.
- *NI-IMAQdx Function Reference Help*—Contains reference information about the LabWindows/CVI functions for the NI-IMAQdx driver software.
- **NI Developer Zone (NIDZ)**—Contains example programs, tutorials, technical presentations, the Instrument Driver Network, a measurement glossary, an online magazine, a product advisor, and a community area where you can share ideas, questions, and source code with developers around the world. The NI Developer Zone is located on the National Instruments Web site at ni.com/zone.

Introduction to NI Vision

This chapter describes the NI Vision for LabWindows/CVI software, outlines the NI Vision function organization, and lists the steps for creating a machine vision application.



Note Refer to the *NI Vision Development Module Readme* for information about the system requirements and installation procedures for NI Vision for LabWindows/CVI.

About NI Vision

NI Vision for LabWindows/CVI—a part of the NI Vision Development Module—is a library of C functions that you can use to develop machine vision and scientific imaging applications. The Vision Development Module also includes the same imaging functions for LabVIEW, and ActiveX controls for Microsoft Visual Basic. Vision Assistant, another Vision Development Module software product, enables you to prototype your application strategy quickly without having to do any programming. Additionally, NI offers Vision Builder for Automated Inspection: configurable machine vision software that you can use to prototype, benchmark, and deploy applications.

Application Development Environments

This release of NI Vision for LabWindows/CVI supports the following application development environments (ADEs) for Windows Vista/XP/2000.

- LabWindows/CVI version 7.1 and later
- Microsoft Visual C/C++ version 6.0 and later



Note NI Vision has been tested and found to work with these ADEs, although other ADEs may also work.

NI Vision Function Tree

The NI Vision function tree (`NI Vision.lfp`) contains separate classes corresponding to groups or types of functions. Table 1-1 lists the NI Vision function types and gives a description of each type.

Table 1-1. NI Vision Function Types

Function Type	Description
Image Management	Functions that create space in memory for images and perform basic image manipulation.
Memory Management	Function that returns, to the operating system, previously used memory that is no longer needed.
Error Management	Functions that set the current error, return the name of the function in which the last error occurred, return the error code of the last error, and clear any pending errors.
Acquisition	Functions that acquire images through an NI image acquisition device.
Display	Functions that cover all aspects of image visualization and image window management.
Overlay	Functions that create and manipulate overlays.
Regions of Interest	Functions that create and manipulate regions of interest.
File I/O	Functions that read and write images from and to files.
Calibration	Functions that learn calibration information and correct distorted images.
Image Analysis	Functions that compute the centroid of an image, profile of a line of pixels, and the mean line profile. This type also includes functions that calculate the pixel distribution and statistical parameters of an image.
Grayscale Processing	Functions for grayscale image processing and analysis.
Binary Processing	Functions for binary image processing and analysis.
Color Processing	Functions for color image processing and analysis.

Table 1-1. NI Vision Function Types (Continued)

Function Type	Description
Pattern Matching	Functions that learn patterns and search for patterns in images, detect shapes in images, and get geometric features from curves and geometric matching templates.
Caliper	Functions designed for gauging, measurement, and inspection applications.
Operators	Functions that perform arithmetic, logic, and comparison operations with two images or with an image and a constant value.
Analytic Geometry	Functions that perform basic geometric calculations on an image.
Frequency Domain Analysis	Functions for the extraction and manipulation of complex planes. Functions of this type perform Fast Fourier Transform (FFT), inverse FFT, truncation, attenuation, addition, subtraction, multiplication, and division of complex images.
Barcode I/O	Functions that find and read 1D barcodes and 2D codes.
LCD	Functions that find and read seven-segment LCD characters.
Meter	Functions that return the arc information of a meter and read the meter.
Utilities	Functions that return structures, and a function that returns a pointer to predefined convolution matrices.
OCR	Functions that perform optical character recognition/optical character verification on an image.
Classification	Functions that classify an image or feature vector.
Inspection	Functions that locate differences in images.
Obsolete	Functions that are no longer necessary but may exist in older applications.

NI Machine Vision Function Tree

The NI Machine Vision function tree (`NIMachineVision.fp`) contains separate classes corresponding to groups or types of functions. Table 1-2 lists the NI Machine Vision function types and gives a description of each type.

Table 1-2. NI Machine Vision Function Types

Function Type	Description
Coordinate Transform	Functions that find coordinate transforms based on image contents.
Count and Measure Objects	Function that counts and measures objects in an image.
Find Patterns	Function that finds patterns in an image.
Locate Edges	Functions that locate different types of edges in an image.
Measure Distances	Functions that measure distances between objects in an image.
Measure Intensities	Functions that measure light intensities in various shaped regions within an image.
Select Region of Interest	Functions that allow a user to select a specific region of interest in an image.

Creating NI Vision Applications

Figures 1-1 and 1-2 illustrate the steps for creating an application with NI Vision. Figure 1-1 describes the general steps to designing a Vision application. The last step in Figure 1-1 is expanded upon in Figure 1-2. You can use a combination of the items in the last step to create your NI Vision application. Refer to the corresponding chapter listed to the side of the item for more information about items in either diagram.

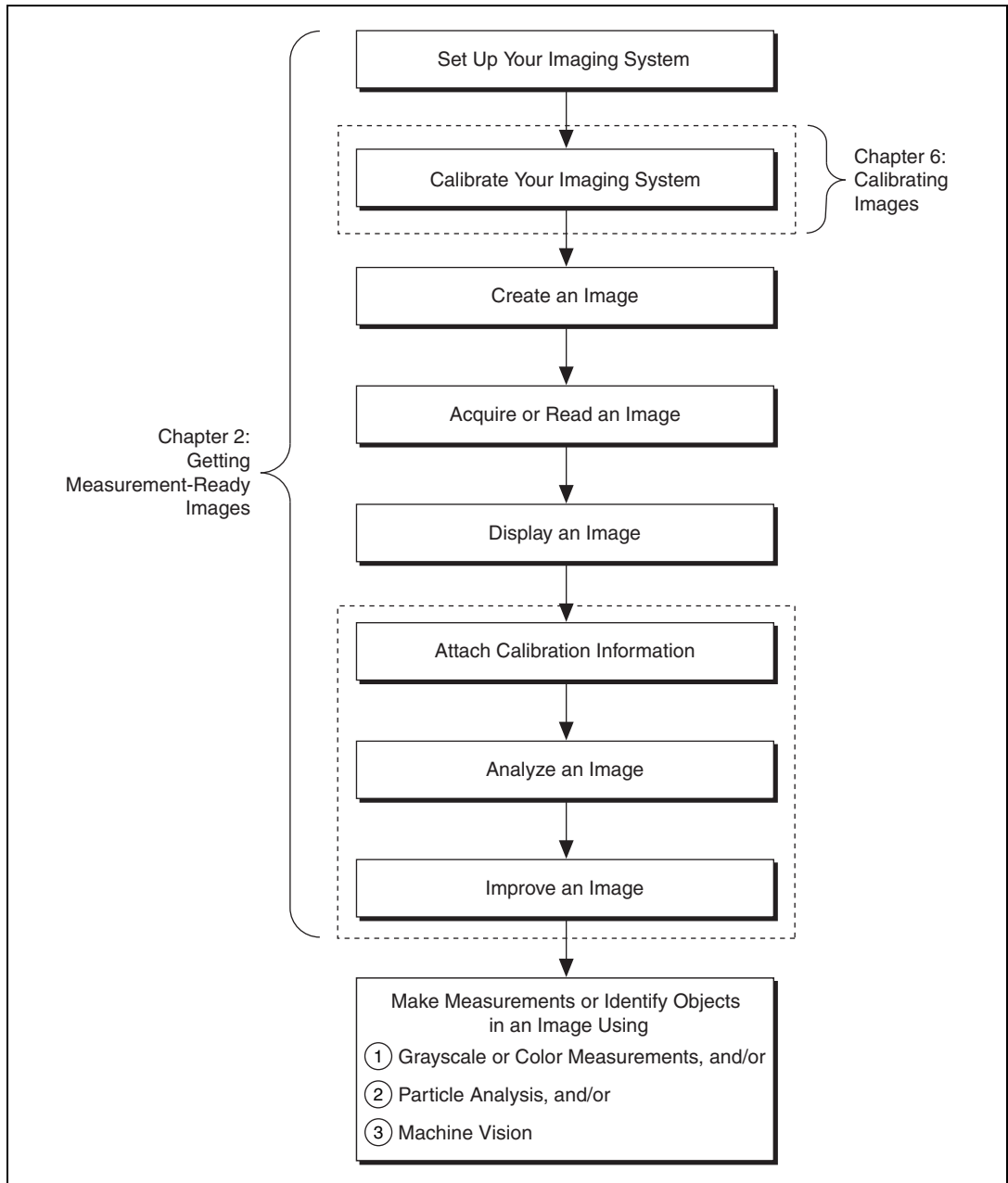


Figure 1-1. General Steps for Designing a Vision Application



Note Diagram items enclosed with dashed lines are optional steps.

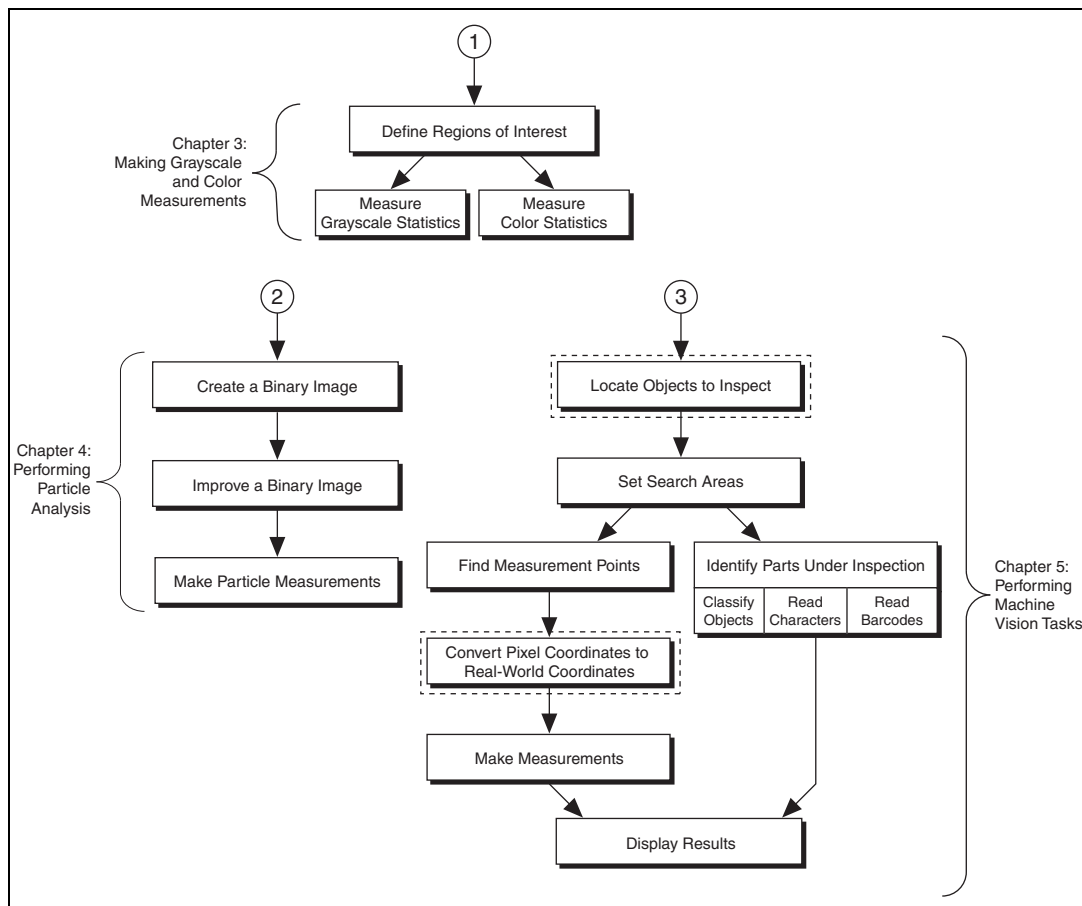


Figure 1-2. Inspection Steps for Building a Vision Application



Note Diagram items enclosed with dashed lines are optional steps.

Getting Measurement-Ready Images

This chapter describes how to set up your imaging system, acquire and display an image, analyze the image, and prepare the image for additional processing.

Set Up Your Imaging System

Before you acquire, analyze, and process images, you must set up your imaging system. How you set up your system depends on your imaging environment and 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.

Complete the following steps to set up your imaging system.

1. Determine the type of equipment you need given your space constraints and the size of the object you need to inspect. Refer to the *System Setup and Calibration* section of the *NI Vision Concepts Help* for more information.
 - a. Make sure your camera sensor is large enough to satisfy your minimum resolution requirement.
 - b. Make sure your lens has a depth of field high enough to keep all of your objects in focus regardless of their distance from the lens. Also, make sure your lens has a focal length that meets your needs.
 - c. Make sure your lighting provides enough contrast between the object under inspection and the background for you to extract the information you need from the image.
2. Position your camera so that it is perpendicular to the object under inspection. If your camera acquires images of the object from an angle, perspective errors occur. Even though you can compensate for these errors with software, NI recommends that you use a perpendicular inspection angle to obtain the most accurate results.



3. Select an NI image acquisition device that meets your needs. National Instruments offers several image acquisition devices, including analog color and monochrome devices as well as digital devices. Visit ni.com/vision for more information about NI image acquisition devices.
4. Configure the driver software for your image acquisition device. If you have a National Instruments image acquisition device, configure the NI-IMAQ or NI-IMAQdx driver software through MAX. Open MAX by double-clicking the Measurement & Automation Explorer icon on your desktop.

Refer to the *Measurement & Automation Explorer Help for NI-IMAQ* or *Measurement & Automation Explorer Help for NI-IMAQdx* for more information.

Calibrate Your Imaging System

After you set up your imaging system, you may want to calibrate your system to assign real-world coordinates to pixel coordinates. This allows you to compensate for perspective and nonlinear errors inherent in your imaging system.

Perspective errors occur when your camera axis is not perpendicular to the object under inspection. Nonlinear distortion may occur from aberrations in the camera lens. Perspective errors and lens aberrations cause images to appear distorted. This distortion misplaces information in an image, but it does not necessarily destroy the information in the image.

Use simple calibration if you only want to assign real-world coordinates to pixel coordinates. Use perspective and nonlinear distortion calibration if you need to compensate for perspective errors and nonlinear lens distortion. For detailed information about calibration, refer to Chapter 5, *Performing Machine Vision Tasks*.

Create an Image

To create an image in NI Vision for LabWindows/CVI, call `imaqCreateImage()`. This function returns an image reference you can use when calling other NI Vision functions. The only limitation to the size and number of images you can acquire and process is the amount of memory on your computer. When you create an image, specify the type of the image. Table 2-1 lists the valid image types.

Table 2-1. NI Vision for LabWindows/CVI Image Types

Value	Description
IMAQ_IMAGE_U8	8 bits per pixel—unsigned, standard monochrome
IMAQ_IMAGE_U16	16 bits per pixel—unsigned, standard monochrome
IMAQ_IMAGE_I16	16 bits per pixel—signed, monochrome
IMAQ_IMAGE_SGL	32 bits per pixel—floating point, monochrome
IMAQ_IMAGE_COMPLEX	2×32 bits per pixel—floating point, native format after a Fast Fourier Transform (FFT)
IMAQ_IMAGE_RGB	32 bits per pixel—standard color
IMAQ_IMAGE_HSL	32 bits per pixel—color
IMAQ_IMAGE_RGB_U64	64 bits per pixel—standard color

You can create multiple images by executing `imaqCreateImage()` as many times as you want. Determine the number of required images through an analysis of your intended application. The decision is based on different processing phases and your need to keep the original image after each processing step. The decision to keep an image occurs before each processing step.

When you create an image, NI Vision creates an internal image structure to hold properties of the image, such as its type and border size. However, no memory is allocated to store the image pixels at this time. NI Vision functions automatically allocate the appropriate amount of memory when the image size is modified. For example, functions that acquire or resample an image alter the image size, so they allocate the appropriate memory space for the image pixels. The return value of `imaqCreateImage()` is a pointer to the image structure. Supply this pointer as an input to all subsequent NI Vision functions.

Most functions in the NI Vision library require one or more image pointers. The number of image pointers a function takes depends on the image processing function and the type of image you want to use. Some NI Vision functions act directly on the image and require only one image pointer. Other functions that process the contents of images require pointers to the source image(s) and to a destination image.

At the end of your application, dispose of each image that you created using `imaqDispose()`.

Source and Destination Images

Some NI Vision functions that modify the contents of an image have source image and destination image input parameters. The source image receives the image to process. The destination image receives the processing results. The destination image can receive either another image or the original, depending on your goals. If you do not want the contents of the original image to change, use separate source and destination images. If you want to replace the original image with the processed image, pass the same image as both the source and destination.

Depending on the function, the image type of the destination image can be the same or different than the image type of the source image. The function descriptions in the *NI Vision for LabWindows/CVI Function Reference Help* include the type of images you can use as image inputs and outputs. NI Vision resizes the destination image to hold the result if the destination is not the appropriate size.

The following examples illustrate source and destination images with `imaqTranspose()`:

- `imaqTranspose(myImage, myImage);`
This function creates a transposed image using the same image for the source and destination. The contents of `myImage` change.
- `imaqTranspose(myTransposedImage, myImage);`
This function creates a transposed image and stores it in a destination different from the source. The `myImage` image remains unchanged, and `myTransposedImage` contains the result.

Functions that perform arithmetic or logical operations between two images have two source images and a destination image. You can perform an operation between two images and then either store the result in a separate destination image or in one of the two source images. In the latter case, make sure you no longer need the original data in the source image before storing the result over the data.

The following examples show the possible combinations using `imaqAdd()`:

- `imaqAdd(myResultImage, myImageA, myImageB);`
This function adds two source images (`myImageA` and `myImageB`) and stores the result in a third image (`myResultImage`). Both source images remain intact after processing.

- `imaqAdd(myImageA, myImageA, myImageB) ;`

This function adds two source images and stores the result in the first source image.

- `imaqAdd(myImageB, myImageA, myImageB) ;`

This function adds two source images and stores the result in the second source image.

Most operations between two images require that the images have the same type and size. However, some arithmetic operations can work between two different types of images, such as 8-bit and 16-bit images.

Some functions perform operations that populate an image. Examples of this type of operation include reading a file, acquiring an image from an image acquisition device, or transforming a 2D array into an image. This type of function can modify the size of an image.

Some functions take an additional **mask** parameter. The presence of this parameter indicates that the processing or analysis is dependent on the contents of another image, the image mask.



Note The image mask must be an 8-bit image for all NI Vision functions except `imaqQuantify()`, which supports both 8-bit and 16-bit image masks.

If you want to apply a processing or analysis function to the entire image, pass `NULL` for the image mask.

Acquire or Read an Image

After you create an image reference, you can acquire an image into your imaging system in three ways. You can acquire an image with a camera through your image acquisition device, load an image from a file stored on your computer, or convert data stored in a 2D array to an image. Functions that acquire images, load images from file, or convert data from a 2D array to an image automatically allocate the memory space required to accommodate the image data.

Acquiring an Image

Use one of the following methods to acquire images with a National Instruments image acquisition device.

- Acquire a single image using `imaqEasyAcquire()`. When you call this function, it initializes the image acquisition device and acquires the next incoming video frame. Use this function for low-speed single capture applications where ease of programming is essential.
- Acquire a single image using `imaqSnap()`. When you call this function, it acquires the next incoming video frame on an image acquisition device you have already initialized using `imgInterfaceOpen()` and `imgSessionOpen()`. Use this function for high-speed single capture applications.
- Acquire images continually through a grab acquisition. Grab functions perform high-speed acquisitions that loop continually on one buffer. Use `imaqSetupGrab()` to start the acquisition. Use `imaqGrab()` to return a copy of the current image. Use `imaqStopAcquisition()` to stop the acquisition.
- Acquire a fixed number of images using a sequence acquisition. Set up the acquisition using `imaqSetupSequence()`. Use `imaqStartAcquisition()` to acquire the number of images you requested during setup. If you want to acquire only certain images, supply `imaqSetupSequence()` with a table describing the number of frames to skip after each acquired frame.
- Acquire images continually through a ringed buffer acquisition. Set up the acquisition using `imaqSetupRing()`. Use `imaqStartAcquisition()` to start acquiring images into the acquired ring buffer. To get an image from the ring, call `imaqExtractFromRing()` or `imaqCopyRing()`. Use `imaqStopAcquisition()` to stop the acquisition.



Note You must use `imgClose()` to release resources associated with the image acquisition device.

Reading a File

Use `imaqReadFile()` to open and read data from a file stored on your computer into the image reference. You can read from image files stored in several standard formats: BMP, TIFF, JPEG, JPEG2000, PNG, and AIPD. The software automatically converts the pixels it reads into the type of image you pass in.

Use `imaqReadVisionFile()` to open an image file containing additional information, such as calibration information, template information for pattern matching, or overlay information. For more information about pattern matching templates and overlays, refer to Chapter 5, *Performing Machine Vision Tasks*.

You can also use `imaqGetFileInfo()` to retrieve image properties—such as image size, recommended image type, and calibration units—without actually reading all the image data.

Converting an Array to an Image

Use `imaqArrayToImage()` to convert a 2D array to an image. You can also use `imaqImageToArray()` to convert an image to a 2D array.

Display an Image

Display an image in an external window using `imaqDisplayImage()`. You can display images in up to 16 different external windows. Use the other display functions to configure the appearance of each external window. Properties you can set include whether the window has scroll bars, a title bar, and whether it is resizable. You can also use `imaqMoveWindow()` to position the external image window at a particular location on your monitor. Refer to the *NI Vision for LabWindows/CVI Function Reference Help* for a complete list of Display functions.



Note Image windows are not LabWindows/CVI panels. They are managed directly by NI Vision.

You can use a color palette to display grayscale images by applying a color palette to the window. Use `imaqSetWindowPalette()` to set predefined color palettes. For example, if you need to display a binary image—an image containing particle regions with pixel values of 1 and a background region with pixel values of 0—apply the predefined binary palette. Refer to the *Display* section of the *NI Vision Concepts Help* for more information about color palettes.



Note At the end of your application, close all open external windows using `imaqCloseWindow()`.

Attach Calibration Information

If you want to attach the calibration information of the current setup to each image you acquire, use `imaqCopyCalibrationInfo2()`. This function takes in a source image containing the calibration information and a destination image that you want to calibrate. The output image is your inspection image with the calibration information attached to it. For detailed information about calibration, refer to Chapter 6, *Calibrating Images*.



Note Because calibration information is part of the image, it is propagated throughout the processing and analysis of the image. Functions that modify the image size, such as geometrical transforms, void the calibration information. Use `imaqWriteVisionFile()` to save the image and all of the attached calibration information to a file.

Analyze an Image

After you acquire and display an image, you may want to analyze the contents of the image for the following reasons:

- To determine whether the image quality is high enough for your inspection task.
- To obtain the values of parameters that you want to use in processing functions during the inspection process.

The histogram and line profile tools can help you analyze the quality of your images.

Use `imaqHistogram()` to analyze the overall grayscale distribution in the image. Use the histogram of the image to analyze two important criteria that define the quality of an image—saturation and contrast. If your image is underexposed, or does not have enough light, the majority of your pixels will have low intensity values, which appear as a concentration of peaks on the left side of your histogram. If your image is overexposed, or has too much light, the majority of your pixels will have high intensity values, which appear as a concentration of peaks on the right side of your histogram. If your image has an appropriate amount of contrast, your histogram will have distinct regions of pixel concentrations. Use the histogram information to decide if the image quality is high enough to separate objects of interest from the background.

If the image quality meets your needs, use the histogram to determine the range of pixel values that correspond to objects in the image. You can use this range in processing functions, such as determining a threshold range during particle analysis.

If the image quality does not meet your needs, try to improve the imaging conditions to get the necessary image quality. You may need to re-evaluate and modify each component of your imaging setup, including lighting equipment and setup, lens tuning, camera operation mode, and acquisition device parameters. If you reach the best possible conditions with your setup but the image quality still does not meet your needs, try to improve the image quality using the image processing techniques described in the *Improve an Image* section of this chapter.

Use `imgLineProfile()` to get the pixel distribution along a line in the image, or use `imgROIProfile()` to get the pixel distribution along a one-dimensional path in the image. By looking at the pixel distribution, you can determine if the image quality is high enough to provide you with sharp edges at object boundaries. Also, you can determine if the image is noisy and identify the characteristics of the noise.

If the image quality meets your needs, use the pixel distribution information to determine some parameters of the inspection functions you want to use. For example, use the information from the line profile to determine the strength of the edge at the boundary of an object. You can input this information into `imgEdgeTool3()` to find the edges of objects along the line.

Improve an Image

Using the information you gathered from analyzing your image, you may want to improve the quality of your image for inspection. You can improve your image with lookup tables, filters, grayscale morphology, and FFTs.

Lookup Tables

Apply *lookup table* (LUT) transformations to highlight image details in areas containing significant information at the expense of other areas. A LUT transformation converts input grayscale values in the source image into other grayscale values in the transformed image. NI Vision provides four functions that directly or indirectly apply lookup tables to images.

- `imgMathTransform()`—Converts the pixel values of an image by replacing them with values from a predefined lookup table. NI Vision has seven predefined lookup tables based on mathematical transformations. Refer to the *Image Processing* section of the *NI Vision Concepts Help* for more information about lookup tables.
- `imgLookup()`—Converts the pixel values of an image by replacing them with values from a user-defined lookup table.

- `imgEqualize()` —Distributes the grayscale values evenly within a given grayscale range. Use `imgEqualize()` to increase the contrast in images containing few grayscale values.
- `imgInverse()` —Inverts the pixel intensities of an image to compute the negative of the image. For example, use `imgInverse()` before applying an automatic threshold to your image if the background pixels are brighter than the object pixels.

Filters

Filter your image when you need to improve the sharpness of transitions in the image or increase the overall signal-to-noise ratio of the image. You can choose either a lowpass or highpass filter depending on your needs.

Lowpass filters remove insignificant details by smoothing the image, removing sharp details, and smoothing the edges between the objects and the background. You can use `imgLowPass()` or define your own lowpass filter with `imgConvolve()` or `imgNthOrderFilter()`.

Highpass filters emphasize details, such as edges, object boundaries, or cracks. These details represent sharp transitions in intensity value. You can define your own highpass filter with `imgConvolve()` or `imgNthOrderFilter()`, or you can use a predefined highpass filter with `imgEdgeFilter()` or `imgCannyEdgeFilter()`. The `imgEdgeFilter()` function allows you to find edges in an image using predefined edge detection kernels, such as the Sobel, Prewitt, and Roberts kernels.

Convolution Filter

The `imgConvolve()` function allows you to use a predefined set of lowpass and highpass filters. Each filter is defined by a kernel of coefficients. Use `imgGetKernel()` to retrieve predefined kernels. If the predefined kernels do not meet your needs, define your own custom filter using a 2D array of floating point numbers.

Nth Order Filter

The `imgNthOrderFilter()` function allows you to define a lowpass or highpass filter depending on the value of N that you choose. One specific N th order filter, the median filter, removes speckle noise, which appears as small black and white dots. Use `imgMedianFilter()` to apply a median filter. Refer to the *Image Processing* section of the *NI Vision Concepts Help* for more information about N th order filters.

Grayscale Morphology

Perform grayscale morphology when you want to filter grayscale features of an image. Grayscale morphology helps you remove or enhance isolated features, such as bright pixels on a dark background. Use these transformations on a grayscale image to enhance non-distinct features before thresholding the image in preparation for particle analysis.

Grayscale morphological transformations compare a pixel to those pixels surrounding it. The transformation keeps the smallest pixel values when performing an erosion or keeps the largest pixel values when performing a dilation.

Refer to the *Image Processing* section of the *NI Vision Concepts Help* for more information about grayscale morphology transformations.

Use `imgaGrayMorphology()` to perform one of the following seven transformations:

- Erosion—Reduces the brightness of pixels that are surrounded by neighbors with a lower intensity.
- Dilation—Increases the brightness of pixels surrounded by neighbors with a higher intensity. A dilation produces the opposite effect of an erosion.
- Opening—Removes bright pixels isolated in dark regions and smooths boundaries.
- Closing—Removes dark pixels isolated in bright regions and smooths boundaries.
- Proper-opening—Removes bright pixels isolated in dark regions and smooths the inner contours of particles.
- Proper-closing—Removes dark pixels isolated in bright regions and smooths the inner contours of particles.
- Auto-median—Generates simpler particles that have fewer details.

FFT

Use the Fast Fourier Transform (FFT) to convert an image into its frequency domain. In an image, details and sharp edges are associated with mid to high spatial frequencies because they introduce significant gray-level variations over short distances. Gradually varying patterns are associated with low spatial frequencies.

An image can have extraneous noise, such as periodic stripes, introduced during the digitization process. In the frequency domain, the periodic

pattern is reduced to a limited set of high spatial frequencies. Also, the imaging setup may produce non-uniform lighting of the field of view, which produces an image with a light drift superimposed on the information you want to analyze. In the frequency domain, the light drift appears as a limited set of low frequencies around the average intensity of the image, the DC component.

You can use algorithms working in the frequency domain to isolate and remove these unwanted frequencies from your image. Complete the following steps to obtain an image in which the unwanted pattern has disappeared but the overall features remain.

1. Use `imgqFFT()` to convert an image from the spatial domain to the frequency domain. This function computes the FFT of the image and results in a complex image representing the frequency information of your image.
2. Improve your image in the frequency domain with a lowpass or highpass frequency filter. Specify which type of filter to use with `imgqAttenuate()` or `imgqTruncate()`. Lowpass filters smooth noise, details, textures, and sharp edges in an image. Highpass filters emphasize details, textures, and sharp edges in images, but they also emphasize noise.
 - Lowpass attenuation—The amount of attenuation is directly proportional to the frequency information. At low frequencies, there is little attenuation. As the frequencies increase, the attenuation increases. This operation preserves all of the zero frequency information. Zero frequency information corresponds to the DC component of the image or the average intensity of the image in the spatial domain.
 - Highpass attenuation—The amount of attenuation is inversely proportional to the frequency information. At high frequencies, there is little attenuation. As the frequencies decrease, the attenuation increases. The zero frequency component is removed entirely.
 - Lowpass truncation—Frequency components above the ideal cutoff frequency are removed, and the frequencies below it remain unaltered.
 - Highpass truncation—Frequency components above the ideal cutoff frequency remain unaltered, and the frequencies below it are removed.
3. To transform your image back to the spatial domain, use `imgqInverseFFT()`.

Complex Image Operations

The `imaqExtractComplexPlane()` and `imaqReplaceComplexPlane()` functions allow you to access, process, and update independently the real and imaginary planes of a complex image. You can also convert planes of a complex image to an array and back with `imaqComplexPlaneToArray()` and `imaqArrayToComplexPlane()`.

Making Grayscale and Color Measurements

This chapter describes how to take measurements from grayscale and color images. You can make inspection decisions based on image statistics, such as the mean intensity level in a region. Based on the image statistics, you can perform many machine vision inspection tasks on grayscale or color images, such as detecting the presence or absence of components, detecting flaws in parts, and comparing a color component with a reference. Figure 3-1 illustrates the basic steps involved in making grayscale and color measurements.

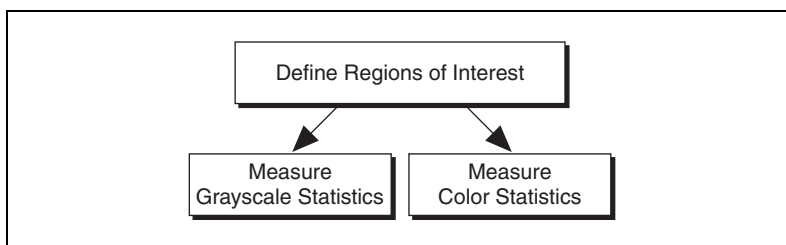


Figure 3-1. Steps to Taking Grayscale and Color Measurements

Define Regions of Interest

A region of interest (ROI) is an area of an image in which you want to focus your image analysis. You can define an ROI interactively, programmatically, or with an image mask.

Defining Regions Interactively

You can interactively define an ROI in a window that displays an image. Use the tools from the NI Vision tools palette to interactively define and manipulate an ROI.

Table 3-1 describes each of the tools and the manner in which you use them.

Table 3-1. Tools Palette Functions














Icon	Tool Name	Function
	Selection Tool	Select an ROI in the image and adjust the position of its control points and contours. Action: Click the desired ROI or control points.
	Point	Select a pixel in the image. Action: Click the desired position.
	Line	Draw a line in the image. Action: Click the initial position, move the cursor to the final position, and click again.
	Rectangle	Draw a rectangle or square in the image. Action: Click one corner and drag to the opposite corner.
	Rotated Rectangle	Draw a rotated rectangle in the image. Action: Click one corner and drag to the opposite corner to create the rectangle. Then, click the lines inside the rectangle and drag to adjust the rotation angle.
	Oval	Draw an oval or circle in the image. Action: Click the center position and drag to the desired size.
	Annulus	Draw an annulus in the image. Action: Click the center position and drag to the desired size. Adjust the inner and outer radii, and adjust the start and end angles.
	Broken Line	Draw a broken line in the image. Action: Click to place a new vertex and double-click to complete the ROI element.
	Polygon	Draw a polygon in the image. Action: Click to place a new vertex and double-click to complete the ROI element.

Table 3-1. Tools Palette Functions (Continued)

Icon	Tool Name	Function
	Freehand Line	Draw a freehand line in the image. Action: Click the initial position, drag to the desired shape, and release the mouse button to complete the shape.
	Freehand Region	Draw a freehand region in the image. Action: Click the initial position, drag to the desired shape, and release the mouse button to complete the shape.
	Zoom	Zoom-in or zoom-out in an image. Action: Click the image to zoom in. Hold down the <Shift> key and click to zoom out.
	Pan	Pan around an image. Action: Click an initial position, drag to the desired position, and release the mouse button to complete the pan.

Hold down the <Shift> key while drawing an ROI to constrain the ROI to the horizontal, vertical, or diagonal axes. Use the Selection Tool to position an ROI by its control points or vertices. ROIs are context sensitive, meaning that the cursor actions differ depending on the ROI with which you interact. For example, if you move your cursor over the side of a rectangle, the cursor changes to indicate that you can click and drag the side to resize the rectangle. If you want to draw more than one ROI in a window, hold down the <Ctrl> key while drawing additional ROIs.

You can display the NI Vision tools palette as part of an ROI constructor window or in a separate, floating window. Follow these steps to invoke an ROI constructor and define an ROI from within the ROI constructor window:

1. Use `imaqConstructROI2()` to display an image and the tools palette in an ROI constructor window, as shown in Figure 3-2.

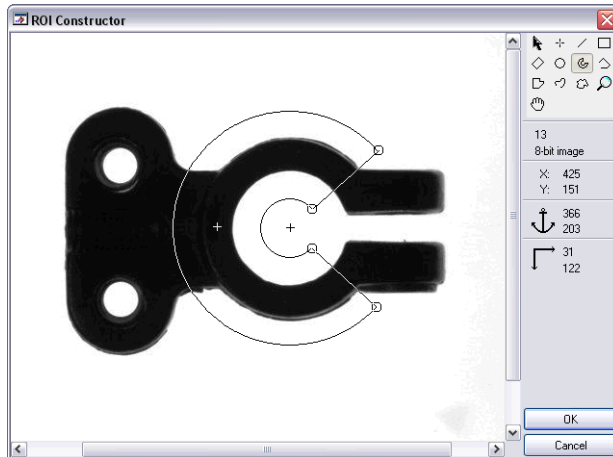


Figure 3-2. ROI Constructor

2. Select an ROI tool from the tools palette.
3. Draw an ROI on your image. Resize and reposition the ROI until it designates the area you want to inspect.
4. Click **OK** to output a descriptor of the region you selected. You can input this ROI descriptor into many analysis and processing functions. You can also convert the ROI descriptor into an image mask, which you can use to process selected regions in the image. Use `imaqROIToMask()` to convert the ROI descriptor into an image mask.

You can also use `imgSelectPoint()`, `imgSelectLine()`, `imgSelectRect()`, and `imgSelectAnnulus()` to define regions of interest. Complete the following steps to use these functions.

1. Call the function to display an image in an ROI Constructor window. Only the tools specific to that function are available for you to use.
2. Draw an ROI on your image. Resize or reposition the ROI until it covers the area you want to process.
3. Click **OK** to populate a structure representing the ROI. You can use this structure as an input to a variety of functions, such as the following functions that measure grayscale intensity.
 - `imgLightMeterPoint()` — Uses the output of `imgSelectPoint()`
 - `imgLightMeterLine()` — Uses the output of `imgSelectLine()`
 - `imgLightMeterRect()` — Uses the output of `imgSelectRect()`

Tools Palette Transformation

The tools palette, shown in Figure 3-3, automatically transforms from the palette on the left to the palette on the right when you manipulate an ROI tool in an image window. The palette on the right displays the characteristics of the ROI you are drawing.

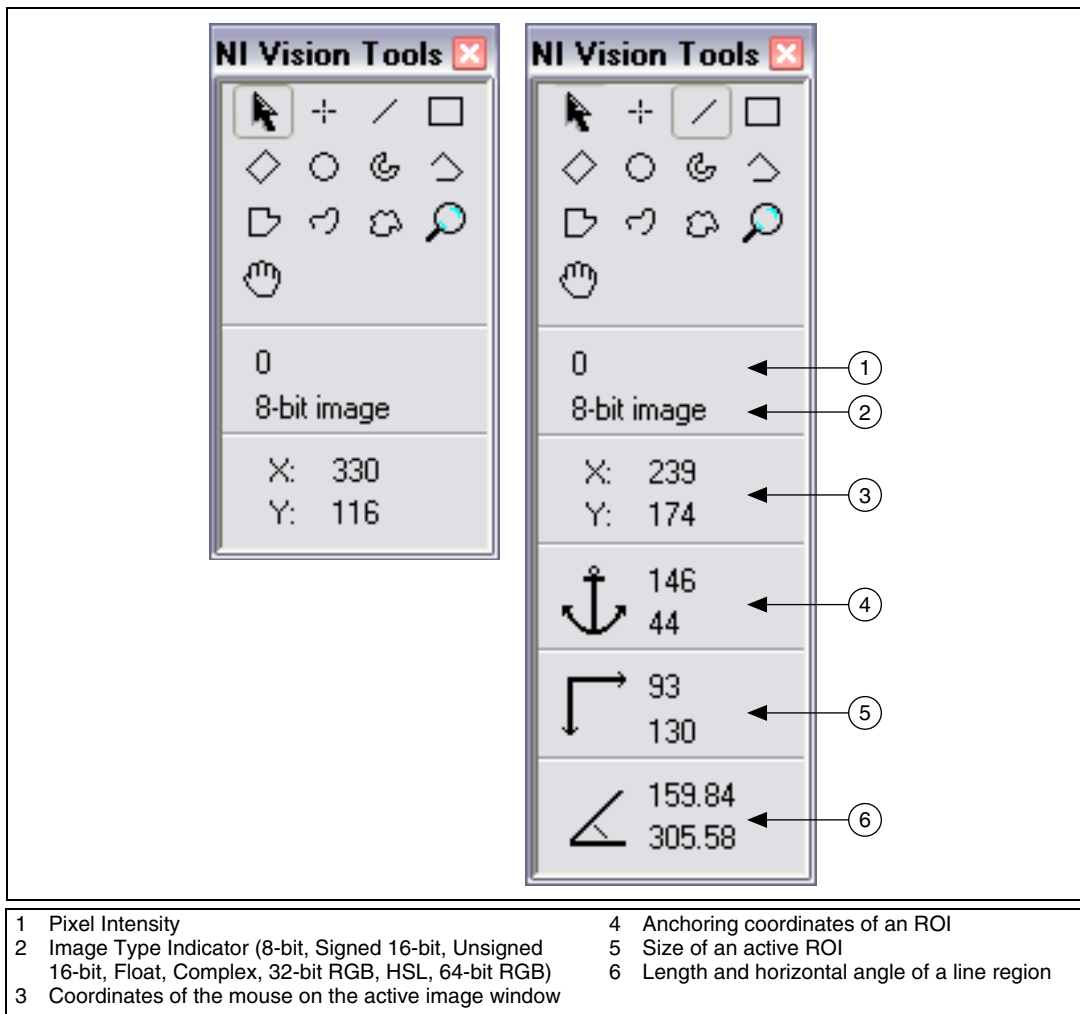


Figure 3-3. Tools Palette Tools and Information

The following list describes how you can display the tools palette in a separate window and manipulate the palette.

- Use `imaqShowToolWindow()` to display the tools palette in a floating window.
- Use `imaqSetupToolWindow()` to configure the appearance of the tools palette.
- Use `imaqMoveToolWindow()` to move the tools palette.
- Use `imaqCloseToolWindow()` to close the tools palette.

If you want to draw an ROI without using an ROI constructor or displaying the tools palette in a separate window, use `imaqSetCurrentTool()`. This function allows you to select a contour from the tools palette without opening the palette.

Defining Regions Programmatically

When you have an automated application, you may need to define regions of interest programmatically. To programmatically define an ROI, create the ROI using `imaqCreateROI()`, and then add the individual contours. A contour is a shape that defines an ROI. You can create contours from points, lines, rectangles, ovals, polygons, and annuli. For example, to add a rectangular contour to an ROI, use `imaqAddRectContour()`.

Specify regions by providing basic parameters that describe the region you want to define. For example, define a point by providing the x-coordinate and y-coordinate. Define a line by specifying the start and end coordinates. Define a rectangle by specifying the coordinates of the top, left point; the width and height; and in the case of a rotated rectangle, the rotation angle.

Defining Regions with Masks

You can define regions to process with image masks. An image mask is an 8-bit image of the same size as or smaller than the image you want to process. Pixels in the mask image determine whether the corresponding pixel in the source image needs to be processed. If a pixel in the image mask has a value different than 0, the corresponding pixel in the source image is processed. If a pixel in the image mask has a value of 0, the corresponding pixel in the source image is left unchanged.

When you need to make intensity measurements on particles in an image, you can use a mask to define the particles. First, threshold your image to make a new binary image. For more information about binary images, refer to Chapter 4, *Performing Particle Analysis*. You can input the binary image or a labeled version of the binary image as a mask image to the intensity measurement function. If you want to make color comparisons, convert the binary image into an ROI descriptor using `imaqMaskToROI()`.

Measure Grayscale Statistics

You can measure grayscale statistics in images using light meters or quantitative analysis functions. You can obtain the center of energy for an image with the centroid function.

Use `imaqLightMeterPoint()` to measure the light intensity at a point in the image. Use `imaqLightMeterLine()` to get pixel value statistics along a line in the image, such as mean intensity, standard deviation, minimum intensity, and maximum intensity. Use `imaqLightMeterRect()` to get the pixel value statistics within a rectangular region in an image.

Use `imaqQuantify()` to obtain the following statistics about the entire image or individual regions in the image: mean intensity, standard deviation, minimum intensity, maximum intensity, area, and the percentage of the image that you analyzed. You can specify regions in the image with a labeled image mask. A labeled image mask is a binary image that has been processed so that each region in the image mask has a unique intensity value. Use `imaqLabel12()` to label your image mask.

Use `imaqCentroid()` to compute the energy center of the image, or of a region within an image.

Measure Color Statistics

Most image processing and analysis functions apply to 8-bit images. However, you can analyze and process individual components of a color image.

Using `imaqExtractColorPlanes()`, you can break down a color image into various sets of primary components, such as RGB (Red, Green, and Blue), HSI (Hue, Saturation, and Intensity), HSL (Hue, Saturation, and Luminance), or HSV (Hue, Saturation, and Value). Each component becomes an 8-bit or 16-bit image that you can process like any other grayscale image. Using `imaqReplaceColorPlanes()`, you can reassemble a color image from a set of three 8-bit or 16-bit images, where each image becomes one of the three primary components. Figures 3-4 and 3-5 illustrate how a color image breaks down into its three primary components.

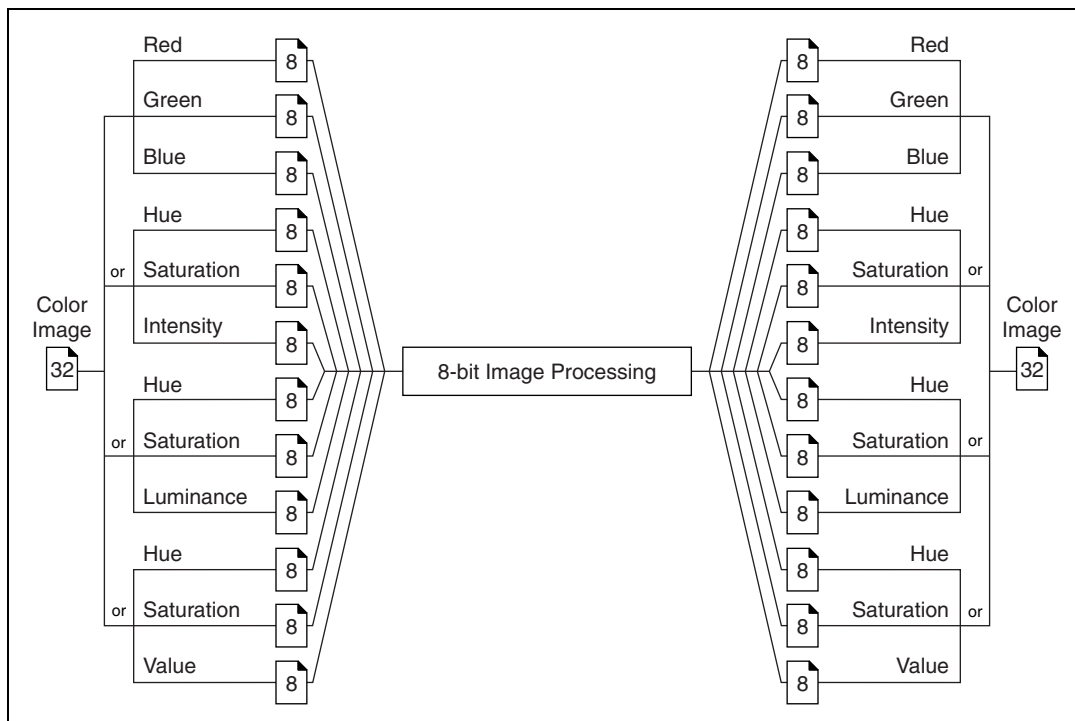


Figure 3-4. Primary Components of a 32-Bit Color Image

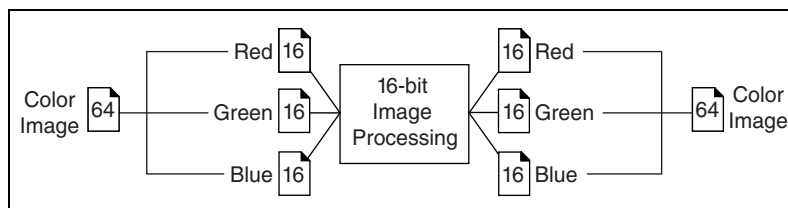


Figure 3-5. Primary Components of a 64-Bit Color Image

Use `imgExtractColorPlanes()` to extract the red, green, blue, hue saturation, intensity, luminance, or value plane of a color image into an 8-bit image.



Note You can also use `imgExtractColorPlanes()` to process the red, green, and blue components of a 64-bit image.

Comparing Colors

You can use the color matching capability of NI Vision to compare or evaluate the color content of an image or regions in an image.

Complete the following steps to compare colors using color matching:

1. Select an image containing the color information that you want to use as a reference. The color information can consist of a single color or multiple dissimilar colors, such as red and blue.
2. Use the entire image or regions in the image to learn the color information using `imaqLearnColor()`, which outputs a color spectrum that contains a compact description of the color information in an image or ROI. Use the color spectrum to represent the learned color information for all subsequent matching operations. Refer to the *Color Inspection* section of the *NI Vision Concepts Help* for more information about color learning.
3. Define an entire image, a region, or multiple regions in an image as the inspection or comparison area.
4. Use `imaqMatchColor()` to compare the learned color information to the color information in the inspection regions. This function returns an array of scores that indicates how close the matches are to the learned color information.
5. Use the color matching score as a measure of similarity between the reference color information and the color information in the image regions being compared.

Learning Color Information

Choose the color information carefully when learning color information.

- Specify an image or regions in an image that contain the color or color information that you want to learn.
- Select the level of detail you want for the learned color information.
- Choose colors that you want to ignore during matching.

Specifying the Color Information to Learn

Because color matching only uses color information to measure similarity, the image or regions in the image representing the object must contain only the significant colors that represent the object, as shown in Figure 3-6a. Figure 3-6b illustrates an unacceptable region containing background colors.

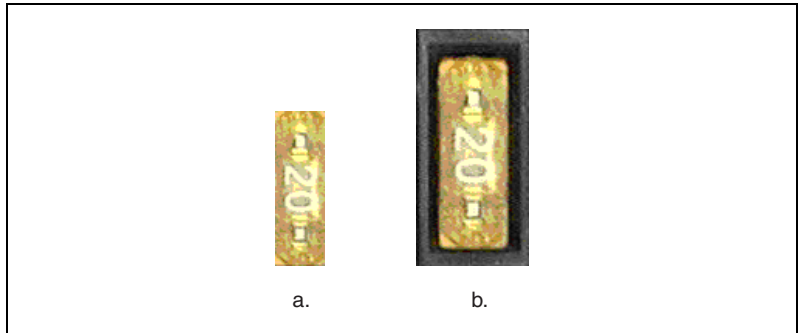


Figure 3-6. Template Color Information

The following sections explain when to learn the color information associated with an entire image, a region in an image, or multiple regions in an image.

Using the Entire Image

You can use an entire image to learn the color spectrum that represents the entire color distribution of the image. In a fabric identification application, for example, an entire image can specify the color information associated with a certain fabric type, as shown in Figure 3-7.

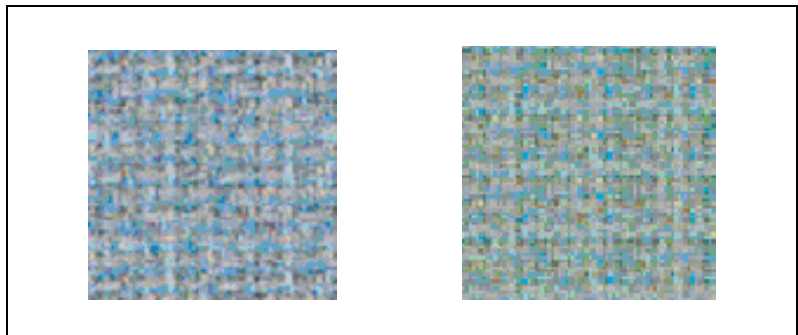


Figure 3-7. Using the Entire Image to Learn Color Distribution

Using a Region in the Image

You can select a region in the image to provide the color information for comparison. A region is helpful for pulling out the useful color information in an image. Figure 3-8 shows an example of using a region that contains the color information that is important for the application.



Figure 3-8. Using a Single Region to Learn Color Distribution

Using Multiple Regions in the Image

The interaction of light with the object surface creates the observed color of that object. The color of a surface depends on the directions of illumination and the direction from which the surface is observed. Two identical objects may have different appearances because of a difference in positioning or a change in the lighting conditions.

Figure 3-9 shows how light reflects differently off of the 3D surfaces of the fuses, resulting in slightly different colors for identical fuses. Compare the 3-amp fuse in the upper row with the 3-amp fuse in the lower row. The difference in light reflection results in different color spectrums for identical fuses.

If you learn the color spectrum by drawing a region of interest around the 3-amp fuse in the upper row, and then do a color matching for the 3-amp fuse in the upper row, you get a very high match score—close to 1000. But the match score for the 3-amp fuse in the lower row is quite low—around 500. This problem could cause a mismatch for the color matching in a fuse box inspection process.

The color learning algorithm of NI Vision uses a clustering process to find the representative colors from the color information specified by one or multiple regions in the image. To create a representative color spectrum for all 3-amp fuses in the learning phase, draw an ROI around the 3-amp fuse in the upper row, hold down the <Shift> key, and draw another ROI around the 3-amp fuse in the lower row. The new color spectrum represents 3-amp

fuses better and results in high match scores (around 800) for both 3-amp fuses. Use as many samples as you want in an image to learn the representative color spectrum for a specified template.

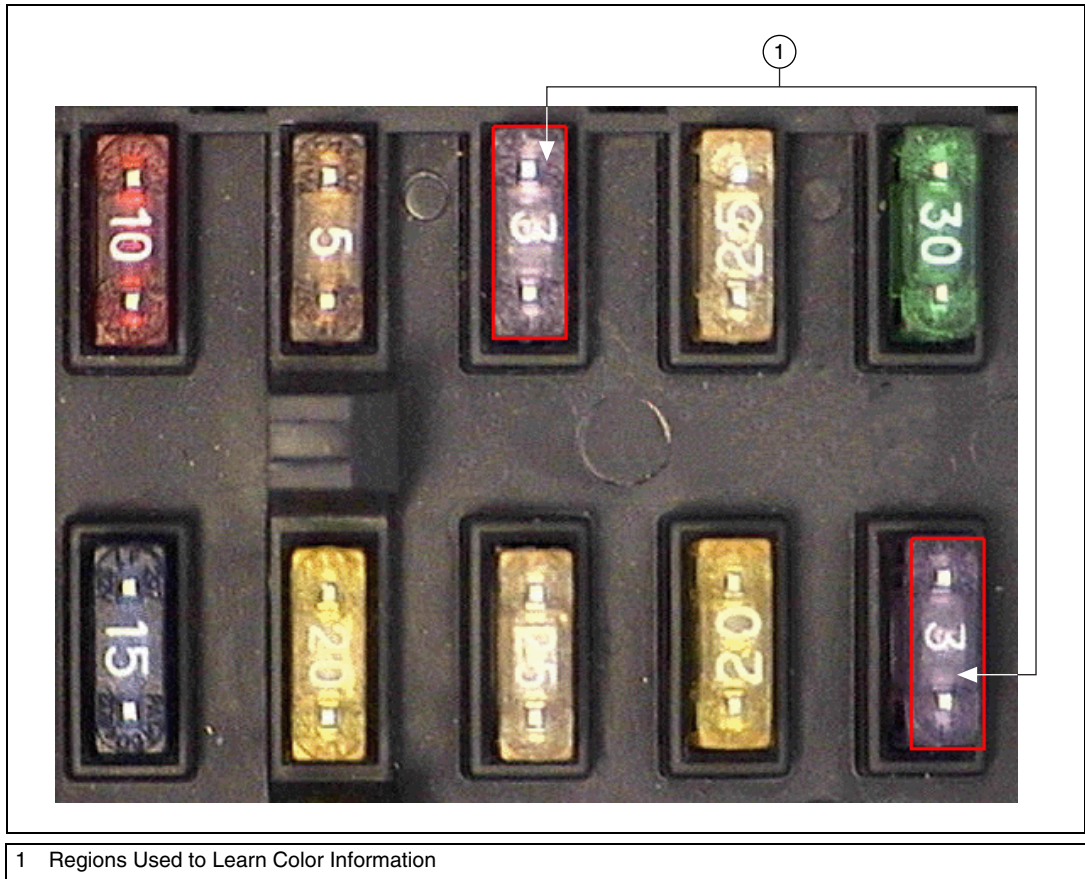


Figure 3-9. Using Multiple Regions to Learn Color Distribution

Choosing a Color Representation Sensitivity

When you learn a color, you need to specify the sensitivity required to specify the color information. An image containing a few, well-separated colors in the color space requires a lower sensitivity to describe the color than an image that contains colors that are close to one another in the color space. Use the **sensitivity** parameter of `imaqLearnColor()` to specify the granularity you want to use to represent the colors. Refer to the [Color Sensitivity](#) section of Chapter 5, *Performing Machine Vision Tasks*, for more information about color sensitivity.

Ignoring Learned Colors

Ignore certain color components in color matching by replacing the corresponding component in the input color spectrum array to -1 . For example, by replacing the last component in the color spectrum with -1 , the white color is ignored during the color matching process. By replacing the second to last component in the color spectrum, the black color is ignored during the color matching process.

To ignore other color components in color matching, determine the index to the color spectrum by locating the corresponding bins in the color wheel, where each bin corresponds to a component in the color spectrum array. Ignoring certain colors such as the background color results in a more accurate color matching score. Ignoring the background color also provides more flexibility when defining the regions of interest in the color matching process. Ignoring other, non-background colors, such as the white color created by glare on a metallic surface, also improves the accuracy of the color matching. Experiment learning the color information about different parts of the images to determine which colors to ignore. Refer to the *Color Inspection* section of the *NI Vision Concepts Help* for more information about the color wheel and color bins.

Performing Particle Analysis

This chapter describes how to perform particle analysis on your images. Use particle analysis to find statistical information about particles—such as the area, location, and presence of particles. With this information, you can perform many machine vision inspection tasks, such as detecting flaws on silicon wafers or detecting soldering defects on electronic boards. Examples of how particle analysis can help you perform inspection tasks include locating structural defects on wood planks or detecting cracks on plastic sheets.

Figure 4-1 illustrates the steps involved in performing particle analysis.

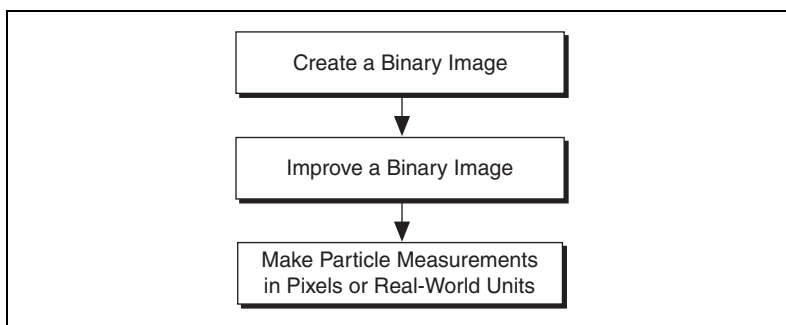


Figure 4-1. Steps to Performing Particle Analysis

Create a Binary Image

Threshold your grayscale or color image to create a binary image. Creating a binary image separates the objects that you want to inspect from the background. The threshold operation sets the background pixels to 0 in the binary image while setting the object pixels to a non-zero value. Object pixels have a value of 1 by default, but you can set the object pixels to any value you choose.

You can use different techniques to threshold your image. If all the objects of interest in your grayscale image fall within a continuous range of intensities and you can specify this threshold range manually, use `imgThreshold()` to threshold your image.

If all the objects in your grayscale image are either brighter or darker than your background, you can use `imAQAutoThreshold2()` to automatically determine the optimal threshold range and threshold your image. Automatic thresholding techniques offer more flexibility than simple thresholds based on fixed ranges. Because automatic thresholding techniques determine the threshold level according to the image histogram, the operation is more independent of changes in the overall brightness and contrast of the image than a fixed threshold. These techniques are more resistant to changes in lighting, which makes them well suited for automated inspection tasks.

If your grayscale image contains objects that have multiple discontinuous grayscale values, use `imAQMultithreshold()` to specify multiple threshold ranges.

If you need to threshold a grayscale image that has nonuniform lighting conditions, such as those resulting from a strong illumination gradient or shadows, use `imAQLocalThreshold()`. You need to define a pixel window that specifies which neighboring pixels are considered in the statistical calculation. The default window size is 32×32 . However, the window size should be approximately the size of the smallest object you want to separate from the background. You also need to specify the local thresholding algorithm to use. The local thresholding algorithm options are the Niblack or background correction algorithm. Refer to the *Image Segmentation* section of the *NI Vision Concepts Help* for more information about local thresholding.

If you need to threshold a color image, use `imAQColorThreshold()`. You must specify threshold ranges for each of the color planes using either the RGB or HSL color model. The binary image resulting from a color threshold is an 8-bit binary image.

Improve the Binary Image

After you threshold your image, you may want to improve the resulting binary image with binary morphology. You can use primary binary morphology or advanced binary morphology to remove unwanted particles, separate connected particles, or improve the shape of particles. Primary morphology functions work on the image as a whole by processing pixels individually. Advanced morphology operations are built upon the primary morphological operators and work on particles as opposed to pixels. Refer to the *Binary Morphology* section of the *NI Vision Concepts Help* for lists of which morphology functions are primary and which are advanced.

The advanced morphology functions require that you specify the type of *connectivity* to use. Use *connectivity-4* when you want NI 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 NI Vision to consider pixels to be part of the same particle even if the pixels touch only at a corner. Refer to the *Binary Morphology* section of the *NI Vision Concepts Help* for more information about connectivity.



Note Use the same type of connectivity throughout your application.

Removing Unwanted Particles

Use `imgRejectBorder()` to remove particles that touch the border of the image. Reject particles on the border of the image when you suspect that the information about those particles is incomplete.

Use `imgSizeFilter()` to remove large or small particles that do not interest you. You can also use the `IMAQ_ERODE`, `IMAQ_OPEN`, and `IMAQ_POPEN` methods in `imgMorphology()` to remove small particles. Unlike `imgSizeFilter()`, these three operations alter the size and shape of the remaining particles.

Use the `IMAQ_HITMISS` method of `imgMorphology()` to locate particular configurations of pixels, which you define with a structuring element. Depending on the configuration of the structuring element, the `IMAQ_HITMISS` method can locate single isolated pixels, cross-shape or longitudinal patterns, right angles along the edges of particles, and other user-specified shapes. Refer to the *Binary Morphology* section of the *NI Vision Concepts Help* for more information about structuring elements.

If you know enough about the shape features of the particles you want to keep, use `imgParticleFilter3()` to filter out particles that do not interest you.

Separating Touching Particles

Use watershed transform and binary morphology to separate touching particles in images.

Using Watershed Transform

Use `imaqDanielssonDistance()` 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. Apply the `imaqWatershedTransform()` function to the distance map to find the watershed separation lines. Use the `imaqMask()` function to superimpose the watershed lines on the original image.

Refer to the *Image Segmentation* section of the *NI Vision Concepts Help* for more information about segmenting images using a watershed transform.

Using Binary Morphology

Use `imaqSeparation()` or apply an erosion or an open operation with `imaqMorphology()` to separate touching objects. The `imaqSeparation()` function is an advanced function that separates particles without modifying their shapes. However, erosion and open operations alter the shape of all the particles.



Note A separation is a time-intensive operation compared to an erosion, watershed transform, or open operation. Consider using one of the methods other than separation if speed is an issue in your application.

Improving Particle Shapes

Use `imaqFillHoles()` to fill holes in the particles. Use `imaqMorphology()` to perform a variety of operations on the particles. You can use the `IMAQ_OPEN`, `IMAQ_POPEN`, `IMAQ_CLOSE`, `IMAQ_PCLOSE`, and `IMAQ_AUTOM` methods to smooth the boundaries of the particles. Open and proper-open smooth the boundaries of the particle by removing small isthmuses while close widens the isthmuses. Close and proper-close fill small holes in the particle. Auto-median removes isthmuses and fills holes. Refer to the *Binary Morphology* section of the *NI Vision Concepts Help* for more information about these methods.

Make Particle Measurements

After you create a binary image and improve it, you can make particle measurements. NI Vision can return the measurements in uncalibrated pixels or calibrated real-world units. With these measurements you can determine the location of particles and their shape features. Use the following functions to perform particle measurements:

- `imaqCountParticles()` —This function returns the number of particles in an image and calculates various measurements for each particle.
- `imaqMeasureParticle()` —This function uses the calculations from `imaqCountParticles()` to return specific measurements of a particle.

Table 4-1 lists all of the measurements that `imaqMeasureParticle()` returns.

Table 4-1. Particle Measurements

Measurement	Description
IMAQ_MT_AREA	Area of the particle
IMAQ_MT_AREA_BY_IMAGE_AREA	Percentage of the particle area covering the image area
IMAQ_MT_AREA_BY_PARTICLE_AND_HOLES_AREA	Percentage of the particle area in relation to the area of its particle and holes
IMAQ_MT_AVERAGE_HORIZ_SEGMENT_LENGTH	Average length of a horizontal segment in the particle
IMAQ_MT_AVERAGE_VERT_SEGMENT_LENGTH	Average length of a vertical segment in the particle
IMAQ_MT_BOUNDING_RECT_BOTTOM	Y-coordinate of the lowest particle point
IMAQ_MT_BOUNDING_RECT_TOP	Y-coordinate of the highest particle point
IMAQ_MT_BOUNDING_RECT_LEFT	X-coordinate of the leftmost particle point
IMAQ_MT_BOUNDING_RECT_RIGHT	X-coordinate of the rightmost particle point

Table 4-1. Particle Measurements (Continued)

Measurement	Description
IMAQ_MT_BOUNDING_RECT_HEIGHT	Distance between the y-coordinate of highest particle point and the y-coordinate of the lowest particle point
IMAQ_MT_BOUNDING_RECT_WIDTH	Distance between the x-coordinate of the leftmost particle point and the x-coordinate of the rightmost particle point
IMAQ_MT_BOUNDING_RECT_DIAGONAL	Distance between opposite corners of the bounding rectangle
IMAQ_MT_CENTER_OF_MASS_X	X-coordinate of the point representing the average position of the total particle mass assuming every point in the particle has a constant density
IMAQ_MT_CENTER_OF_MASS_Y	Y-coordinate of the point representing the average position of the total particle mass assuming every point in the particle has a constant density
IMAQ_MT_COMPACTNESS_FACTOR	Area divided by the product of bounding rectangle width and bounding rectangle height
IMAQ_MT_CONVEX_HULL_AREA	Area of the smallest convex polygon containing all points in the particle
IMAQ_MT_CONVEX_HULL_PERIMETER	Perimeter of the smallest convex polygon containing all points in the particle
IMAQ_MT_EQUIVALENT_ELLIPSE_MAJOR_AXIS	Length of the major axis of the ellipse with the same perimeter and area as the particle
IMAQ_MT_EQUIVALENT_ELLIPSE_MINOR_AXIS	Length of the minor axis of the ellipse with the same perimeter and area as the particle

Table 4-1. Particle Measurements (Continued)

Measurement	Description
IMAQ_MT_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
IMAQ_MT_EQUIVALENT_RECT_DIAGONAL	Distance between opposite corners of the rectangle with the same perimeter and area as the particle
IMAQ_MT_EQUIVALENT_RECT_LONG_SIDE	Longest side of the rectangle with the same perimeter and area as the particle
IMAQ_MT_EQUIVALENT_RECT_SHORT_SIDE	Shortest side of the rectangle with the same perimeter and area as the particle
IMAQ_MT_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
IMAQ_MT_ELONGATION_FACTOR	Max feret diameter divided by equivalent rectangle short side (feret)
IMAQ_MT_FIRST_PIXEL_X	X-coordinate of the highest, leftmost particle pixel
IMAQ_MT_FIRST_PIXEL_Y	Y-coordinate of the highest, leftmost particle pixel
IMAQ_MT_HU_MOMENT_1	First Hu moment
IMAQ_MT_HU_MOMENT_2	Second Hu moment
IMAQ_MT_HU_MOMENT_3	Third Hu moment
IMAQ_MT_HU_MOMENT_4	Fourth Hu moment
IMAQ_MT_HU_MOMENT_5	Fifth Hu moment
IMAQ_MT_HU_MOMENT_6	Sixth Hu moment
IMAQ_MT_HU_MOMENT_7	Seventh Hu moment

Table 4-1. Particle Measurements (Continued)

Measurement	Description
IMAQ_MT_HEYWOOD_CIRCULARITY_FACTOR	Perimeter divided by the circumference of a circle with the same area
IMAQ_MT_HOLES_PERIMETER	Sum of the perimeters of each hole in the particle
IMAQ_MT_HYDRAULIC_RADIUS	Particle area divided by the particle perimeter
IMAQ_MT_HOLES_AREA	Sum of the areas of each hole in the particle
IMAQ_MT_IMAGE_AREA	Area of the image
IMAQ_MT_MAX_FERET_DIAMETER	Distance between the start and end of the line segment connecting the two perimeter points that are the furthest apart
IMAQ_MT_MAX_FERET_DIAMETER_END_X	X-coordinate of the end of the line segment connecting the two perimeter points that are the furthest apart
IMAQ_MT_MAX_FERET_DIAMETER_END_Y	Y-coordinate of the end of the line segment connecting the two perimeter points that are the furthest apart
IMAQ_MT_MAX_FERET_DIAMETER_ORIENTATION	Angle of the line segment connecting the two perimeter points that are the furthest apart
IMAQ_MT_MAX_FERET_DIAMETER_START_X	X-coordinate of the start of the line segment connecting the two perimeter points that are the furthest apart
IMAQ_MT_MAX_FERET_DIAMETER_START_Y	Y-coordinate of the start of the line segment connecting the two perimeter points that are the furthest apart

Table 4-1. Particle Measurements (Continued)

Measurement	Description
IMAQ_MT_MAX_HORIZ_SEGMENT_LENGTH_LEFT	X-coordinate of the leftmost pixel in the longest row of contiguous pixels in the particle
IMAQ_MT_MAX_HORIZ_SEGMENT_LENGTH_RIGHT	X-coordinate of the rightmost pixel in the longest row of contiguous pixels in the particle
IMAQ_MT_MAX_HORIZ_SEGMENT_LENGTH_ROW	Y-coordinate of all of the pixels in the longest row of contiguous pixels in the particle
IMAQ_MT_MOMENT_OF_INERTIA_XX	Moment of inertia in the x direction twice
IMAQ_MT_MOMENT_OF_INERTIA_XY	Moment of inertia in the x and y directions
IMAQ_MT_MOMENT_OF_INERTIA_YY	Moment of inertia in the y direction twice
IMAQ_MT_MOMENT_OF_INERTIA_XXX	Moment of inertia in the x direction three times
IMAQ_MT_MOMENT_OF_INERTIA_XXY	Moment of inertia in the x direction twice and the y direction once
IMAQ_MT_MOMENT_OF_INERTIA_XYY	Moment of inertia in the x direction once and the y direction twice
IMAQ_MT_MOMENT_OF_INERTIA_YYY	Moment of inertia in the y direction three times
IMAQ_MT_NORM_MOMENT_OF_INERTIA_XX	Normalized moment of inertia in the x direction twice
IMAQ_MT_NORM_MOMENT_OF_INERTIA_XY	Normalized moment of inertia in the x and y directions
IMAQ_MT_NORM_MOMENT_OF_INERTIA_YY	Normalized moment of inertia in the y direction twice
IMAQ_MT_NORM_MOMENT_OF_INERTIA_XXX	Normalized moment of inertia in the x direction three times

Table 4-1. Particle Measurements (Continued)

Measurement	Description
IMAQ_MT_NORM_MOMENT_OF_INERTIA_XXY	Normalized moment of inertia in the x direction twice and the y direction once
IMAQ_MT_NORM_MOMENT_OF_INERTIA_XYY	Normalized moment of inertia in the x direction once and the y direction twice
IMAQ_MT_NORM_MOMENT_OF_INERTIA_YYY	Normalized moment of inertia in the y direction three times
IMAQ_MT_NUMBER_OF_HOLES	Number of holes in the particle
IMAQ_MT_NUMBER_OF_HORIZ_SEGMENTS	Number of horizontal segments in the particle
IMAQ_MT_NUMBER_OF_VERT_SEGMENTS	Number of vertical segments in the particle
IMAQ_MT_ORIENTATION	Angle of the line that passes through the particle center of mass about which the particle has the lowest moment of inertia
IMAQ_MT_PERIMETER	Length of the outer boundary of the particle
IMAQ_MT_PARTICLE_AND_HOLES_AREA	Area of a particle that completely covers the image
IMAQ_MT_RATIO_OF_EQUIVALENT_ELLIPSE_AXES	Equivalent ellipse major axis divided by equivalent ellipse minor axis
IMAQ_MT_RATIO_OF_EQUIVALENT_RECT_SIDES	Equivalent rectangle long side divided by equivalent rectangle short side
IMAQ_MT_SUM_X	Sum of all x-coordinates in the particle
IMAQ_MT_SUM_Y	Sum of all y-coordinates in the particle
IMAQ_MT_SUM_XX	Sum of all x-coordinates squared in the particle

Table 4-1. Particle Measurements (Continued)

Measurement	Description
IMAQ_MT_SUM_XY	Sum of all x-coordinates multiplied by y-coordinates in the particle
IMAQ_MT_SUM_YY	Sum of all y-coordinates squared in the particle
IMAQ_MT_SUM_XXX	Sum of all x-coordinates cubed in the particle
IMAQ_MT_SUM_XXY	Sum of all x-coordinates squared multiplied by y-coordinates in the particle
IMAQ_MT_SUM_XYY	Sum of all x-coordinates multiplied by y-coordinates squared in the particle
IMAQ_MT_SUM_YYY	Sum of all y-coordinates cubed in the particle
IMAQ_MT_TYPE_FACTOR	Factor relating area to moment of inertia
IMAQ_MT_WADDEL_DISK_DIAMETER	Diameter of a disk with the same area as the particle

Performing Machine Vision Tasks

This chapter describes how to perform many common machine vision inspection tasks. The most common inspection tasks are detecting the presence or absence of parts in an image and measuring the dimensions of parts to see if they meet specifications.

Measurements are based on characteristic features of the object represented in the image. Image processing algorithms traditionally classify the type of information contained in an image as edges, surfaces and textures, or patterns. Different types of machine vision algorithms leverage and extract one or more types of information.

Edge detectors and derivative techniques—such as rakes, concentric rakes, and spokes—use edges represented in the image. They locate, with high accuracy, the position of the edge of an object in the image. For example, a technique called clamping uses edge locations to measure the width of a part. You can combine multiple edge locations to compute intersection points, projections, circles, or ellipse fits.

Pattern matching algorithms use edges and patterns. Pattern matching can locate, with very high accuracy, the position of fiducials or characteristic features of the part under inspection. Those locations can then be combined to compute lengths, angles, and other object measurements.

The robustness of your measurement relies on the stability of your image acquisition conditions. Sensor resolution, lighting, optics, vibration control, part fixture, and general environment are key components of the imaging setup. All the elements of the image acquisition chain directly affect the accuracy of the measurements.

Figure 5-1 illustrates the basic steps involved in performing machine vision inspection tasks.

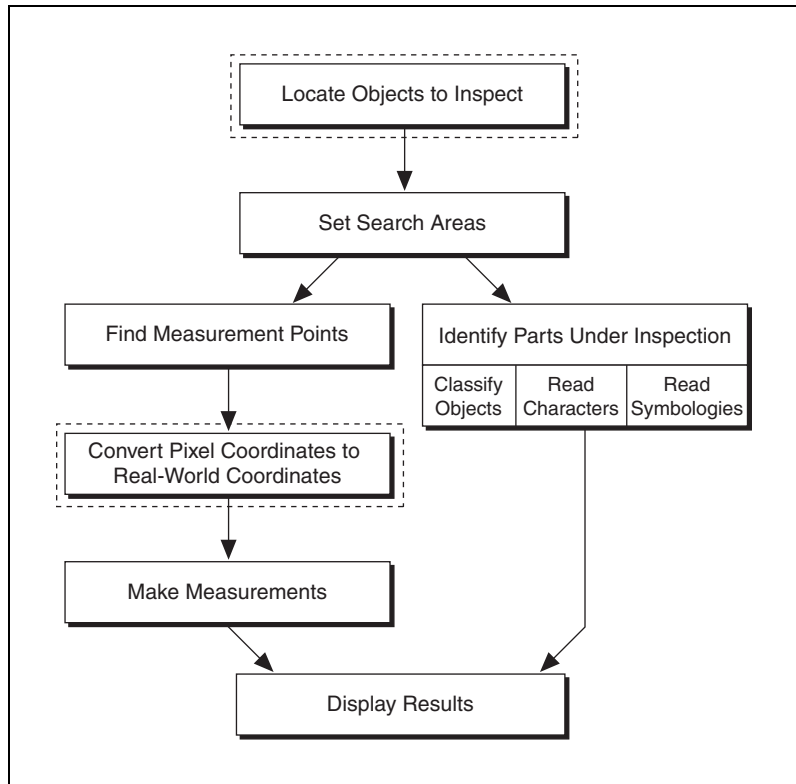


Figure 5-1. Steps to Performing Machine Vision



Note Diagram items enclosed with dashed lines are optional steps.

Locate Objects to Inspect

In a typical machine vision application, you extract measurements from ROIs rather than the entire image. To use this technique, the parts of the object you are interested in must always appear inside the ROI you define.

If the object under inspection is always at the same location and orientation in the images you need to process, defining an ROI is simple. Refer to the [Set Search Areas](#) section of this chapter for information about selecting an ROI.

Often, the object under inspection appears rotated or shifted relative to the reference image in which you located the object. When this occurs, the ROIs need to shift and rotate with the parts of the object in which you are interested. To move the ROIs with the object, you must define a reference coordinate system relative to the object in the reference image. During the measurement process, the coordinate system moves with the object when it appears shifted and rotated in the image you need to process. This coordinate system is referred to as the measurement coordinate system. The measurement methods automatically move the ROIs to the correct position using the position of the measurement coordinate system with respect to the reference coordinate system. Refer to the *Dimensional Measurements* section of the *NI Vision Concepts Help* for information about coordinate systems.

You can build a coordinate transform using edge detection or pattern matching. The output of the edge detection and pattern matching functions that build a coordinate system are the origin, angle, and axes direction of the coordinate system. Some machine vision functions take this output and adjust the regions of inspection automatically. You can also use these outputs to programmatically move the regions of inspection relative to the object.

Using Edge Detection to Build a Coordinate Transform

You can build a coordinate transform using two edge detection techniques. Use `imaqFindTransformRect2()` to define a coordinate system using one rectangular region. Use `imaqFindTransformRects2()` to define a coordinate system using two independent rectangular regions. Follow these steps to build a coordinate transform using edge detection.

1. Specify one or two rectangular ROIs.
 - a. If you use `imaqFindTransformRect2()`, specify one rectangular ROI that includes part of two straight, nonparallel boundaries of the object, as shown in Figure 5-2. This rectangular region must be large enough to include these boundaries in all the images you want to inspect.

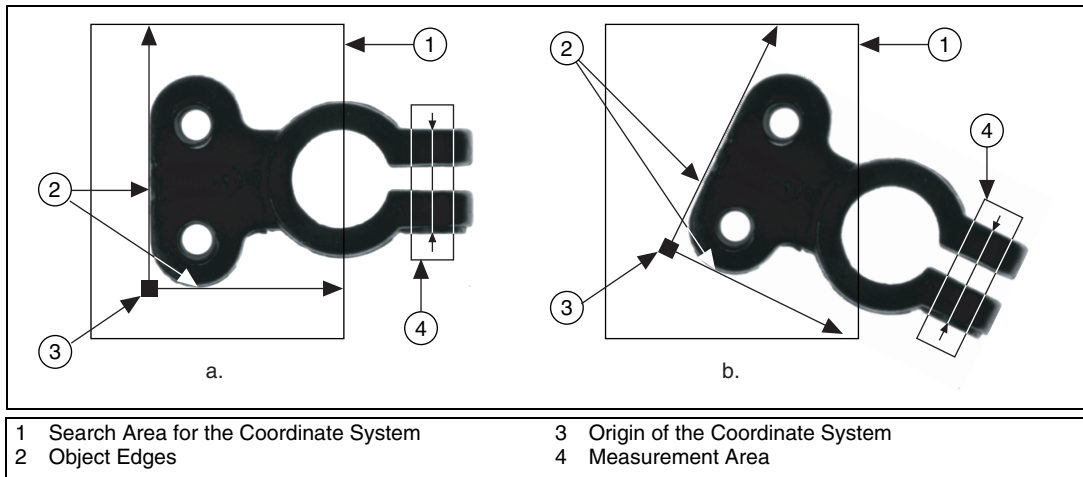


Figure 5-2. Coordinate Systems of a Reference Image and Inspection Image

- b. If you use `imgqFindTransformRects2()`, specify two rectangular objects, each containing one separate, straight boundary of the object, as shown in Figure 5-3. The boundaries cannot be parallel. The regions must be large enough to include the boundaries in all the images you want to inspect.

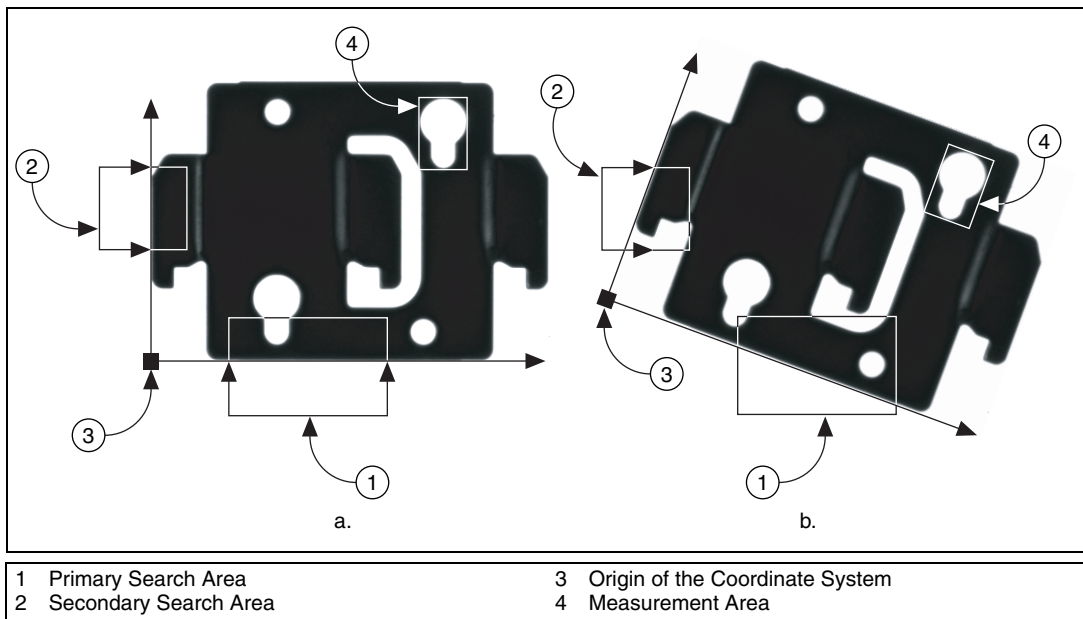


Figure 5-3. Locating Coordinate System Axes with Two Search Areas

2. Use the **findTransformOptions** parameter to choose the options you need to locate the edges on the object, the coordinate system axis direction, and the results that you want to overlay onto the image. Use the **straightEdgeOptions** to determine how to fit a straight line to the found edges. Set either the **findTransformOptions** or **straightEdgeOptions** parameter to `NULL` to use the default options.
3. Choose the mode for the function. To build a coordinate transform for the first time, set the **mode** parameter to `IMAQ_FIND_REFERENCE`. To update the coordinate transform in subsequent images, set this mode to `IMAQ_UPDATE_TRANSFORM`.

Using Pattern Matching to Build a Coordinate Transform

You can build a coordinate transform using pattern matching. Use `imaqFindTransformPattern()` to define a coordinate system based on the location of a reference feature. Use this technique when the object under inspection does not have straight, distinct edges. Complete the following steps to build a coordinate reference system using pattern matching.



Note The object may rotate 360° in the image using this technique if you use rotation-invariant pattern matching.

1. Define a template that represents the part of the object that you want to use as a reference feature. Refer to the [Find Measurement Points](#) section of this chapter for more information about defining a template.
2. Define a rectangular ROI in which you expect to find the template.
3. Use the **options** parameter to select your options for finding the pattern and the results that you want to overlay onto the image. When setting the `Mode` element, select `IMAQ_MATCH_ROTATION_INVARIANT` when you expect your template to appear rotated in the inspection images. Otherwise, select `IMAQ_MATCH_SHIFT_INVARIANT`. Set the **options** parameter to `NULL` to use the default options.
4. Choose the mode for the function. To build a coordinate transform for the first time, set the **mode** parameter to `IMAQ_FIND_REFERENCE`. To update the coordinate system in subsequent images, set the **mode** parameter to `IMAQ_UPDATE_TRANSFORM`.

Choosing a Method to Build the Coordinate Transform

Figure 5-4 guides you through choosing the best method for building a coordinate transform for your application.

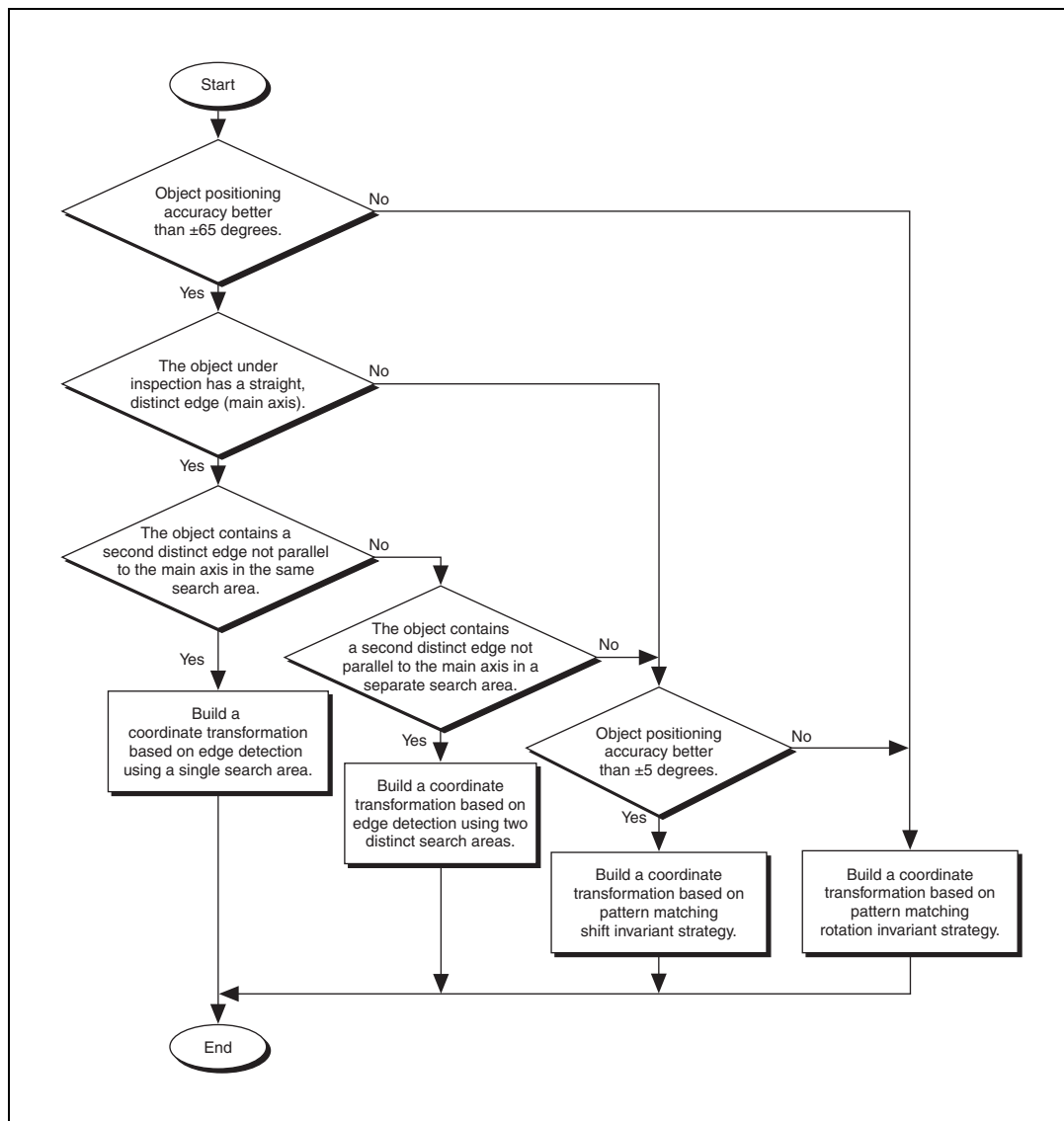


Figure 5-4. Building a Coordinate Transform

Set Search Areas

You use ROIs to define search areas in your images and limit the areas in which you perform your processing and inspection. You can define ROIs interactively or programmatically.

Defining Regions Interactively

Complete the following steps to interactively define an ROI:

1. Use `imaqConstructROI2()` to display an image and the tools palette in a window.
2. Select an ROI tool from the tools palette.
3. Draw an ROI on your image. Resize and reposition the ROI until it specifies the area you want to process.
4. Click **OK** to output a descriptor of the region you selected. You can input the ROI descriptor into many analysis and processing functions.

You can also use `imaqSelectRect()` and `imaqSelectAnnulus()` to define ROIs. Complete the following steps to use these functions:

1. Call the function to display an image in a window. Only the tools specific to that function are available for you to use.
2. Select an ROI tool from the tools palette.
3. Draw an ROI on your image. Resize or reposition the ROI until it specifies the area you want to process.

Click **OK** to output a description of the ROI. You can use this description as an input for the following functions.

ROI Selection Function	Measurement Function
<code>imaqSelectRect()</code>	<code>imaqFindPattern()</code> <code>imaqClampMax()</code> <code>imaqClampMin()</code> <code>imaqFindEdge2()</code>
<code>imaqSelectAnnulus()</code>	<code>imaqFindCircularEdge()</code> <code>imaqFindConcentricEdge()</code>

Defining Regions Programmatically

When you have an automated application, you need to define ROIs programmatically. You can programmatically define regions in two ways:

- Specify the contours of the ROI.
- Specify individual structures by providing basic parameters that describe the region you want to define. You can specify a rotated rectangle by providing the coordinates of the center, the width, the height, and the rotation angle. You can specify an annulus by providing the coordinates of the center, inner radius, outer radius, start angle, and end angle. You can specify a point by setting its x-coordinates and y-coordinates. You can specify a line by setting the coordinates of the start and end points.

Refer to Chapter 3, [Making Grayscale and Color Measurements](#), for more information about defining ROIs.

Find Measurement Points

After you set regions of inspection, locate points within those regions on which you can base measurements. You can locate measurement points using edge detection, pattern matching, color pattern matching, and color location.

Finding Features Using Edge Detection

Use the edge detection tools to identify and locate sharp discontinuities in an image. Discontinuities typically represent abrupt changes in pixel intensity values, which characterize the boundaries of objects.

Finding Lines or Circles

If you want to find points along the edge of an object and find a line describing the edge, use `imaqFindEdge2()`, `imaqFindConcentricEdges()`, or `imaqStraightEdge()`. The `imaqFindEdge2()` and `imaqStraightEdge()` functions find edges based on rectangular search areas, as shown in Figure 5-5. The `imaqFindConcentricEdge()` function finds edges based on annular search areas.

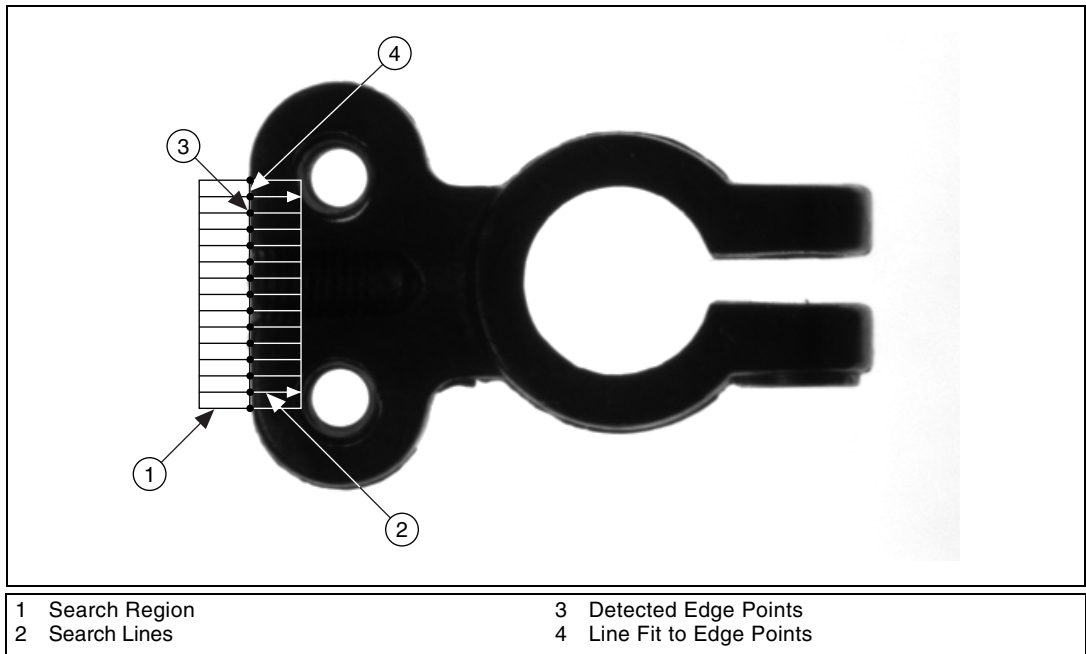


Figure 5-5. Finding a Straight Feature

If you want to find points along a circular edge and find the circle that best fits the edge, as shown in Figure 5-6, use `imaqFindCircularEdge()`.

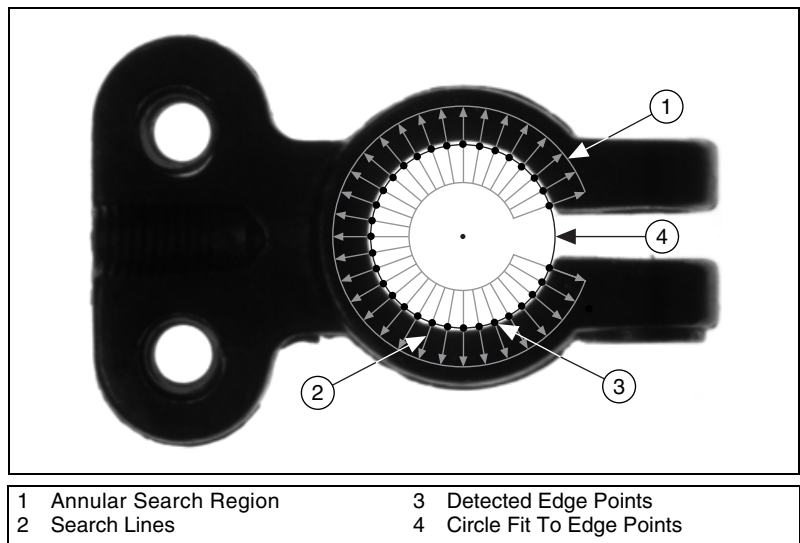


Figure 5-6. Finding a Circular Feature

Use `imaqFindEdge2()` and `imaqFindConcentricEdge()` to locate the intersection points between a set of search lines within the search region and the edge of an object. You can specify the search region using `imaqSelectRect()` or `imaqSelectAnnulus()`. Specify the separation between the lines that the functions use to detect edges. The functions determine the intersection points based on their contrast, width, and steepness. The software calculates a best-fit line with outliers rejected or a best-fit circle through the points it found. The functions return the coordinates of the edges found.

Finding Edge Points Along One Search Contour

Use `imaqSimpleEdge()` or `imaqEdgeTool3()` to find edge points along a contour. Using `imaqSimpleEdge()`, you can find the first edge, last edge, or all edges along the contour. Use `imaqSimpleEdge()` when your image contains little noise and the object and background are clearly differentiated. Otherwise, use `imaqEdgeTool3()`.

These functions require you to input the coordinates of the points along the search contour. Use `imaqROIProfile()` to obtain the coordinates along the edge of each contour in an ROI. If you have a straight line, use `imaqGetPointsOnLine()` to obtain the points along the line instead of using an ROI.

These functions determine the edge points based on their contrast and slope. You can specify whether you want to find the edge points using subpixel accuracy.

Finding Edge Points Along Multiple Search Contours

Use `imaqRake2()`, `imaqSpoke2()`, and `imaqConcentricRake2()` to find edge points along multiple search contours. Pass in an ROI to define the search region for these functions.

The `imaqRake2()` function works on a rectangular search region. The search lines are drawn parallel to the orientation of the rectangle. Control the number of search lines in the region by specifying the distance, in pixels, between each line. Specify 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 `imaqSpoke2()` function works on an annular search region, scanning the 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, in degrees, between each line. Specify the search direction as either going from the center outward or from the outer boundary to the center.

The `imaqConcentricRake2()` function works on an annular search region. The concentric rake is an adaptation of the Rake to an annular region. NI Vision does edge detection 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 counterclockwise.

Finding Points Using Pattern Matching

The pattern matching algorithms in NI Vision measure the similarity between an idealized representation of a feature, called a template, and the feature that may be present in an image. A feature is defined as a specific pattern of pixels in an image. Pattern matching returns the location of the center of the template and the template orientation. Complete the following generalized steps to find features in an image using pattern matching.

1. Define a template image in the form of a reference or fiducial pattern.
2. Use the reference pattern to train the pattern matching algorithm with `imaqLearnPattern3()`.
3. Define an image or an area of an image as the search area. A small search area reduces the time to find the features.
4. Set the tolerances and parameters to specify how the algorithm operates at run time using the **options** parameter of `imaqMatchPattern2()`.
5. Test the search algorithm on test images using `imaqMatchPattern2()`.
6. Verify the results using a ranking method.

Defining and Creating Effective Template Images

The selection of an effective template image plays a critical part in obtaining good results. Because the template image represents the pattern that you want to find, make sure that all the important and unique characteristics of the pattern are well defined in the image.

These factors are critical in creating a template image: symmetry, feature detail, positional information, and background information.

Symmetry

A rotationally symmetric template, shown in Figure 5-7a, is less sensitive to changes in rotation than one that is rotationally asymmetric, shown in Figure 5-7b. A rotationally symmetric template provides good positioning information but no orientation information.

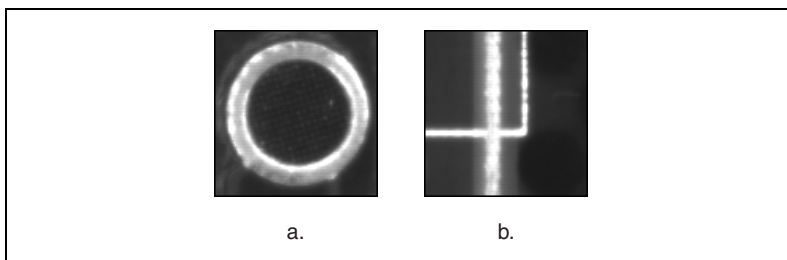


Figure 5-7. Symmetry

Feature Detail

A template with relatively coarse features, shown in Figure 5-8a, is less sensitive to variations in size and rotation than a model with fine features, Figure 5-8b. However, the model must contain enough detail to identify the feature.

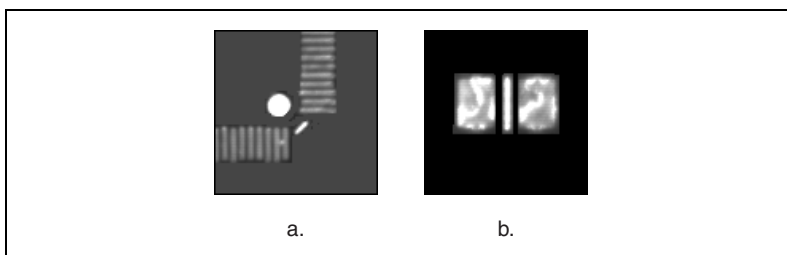


Figure 5-8. Feature Detail

Positional Information

A template with strong edges in both the x and y directions is easier to locate. Figure 5-9a shows good positional information in both the x and y directions, while Figure 5-9b shows insufficient positional information in the y direction.

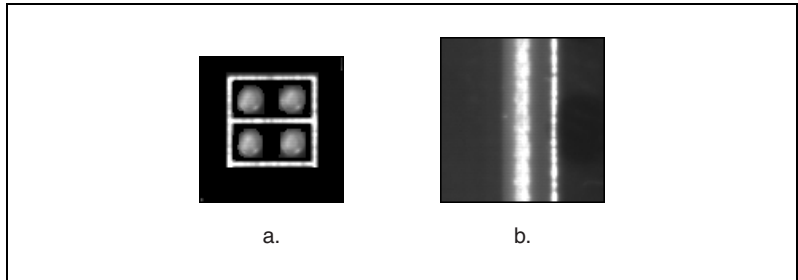


Figure 5-9. Positional Information

Background Information

Unique background information in a template improves search performance and accuracy. Figure 5-10a shows a pattern with insufficient background information. Figure 5-10b illustrates a pattern with sufficient background information.

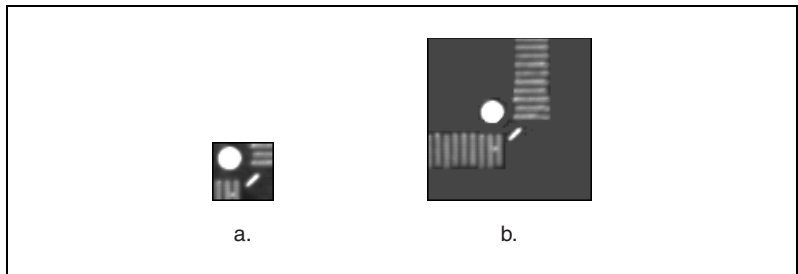


Figure 5-10. Background Information

Training the Pattern Matching Algorithm

After you create a good template image, the pattern matching algorithm has to learn the important features of the template. Use `imgLearnPattern3()` to learn the template. The learning process depends on the type of matching that you expect to perform. If you do not expect the instance of the template in the image to rotate or change its size, then the pattern matching algorithm has to learn only those features from the template that are necessary for shift-invariant matching. However, if

you want to match the template at any orientation, use rotation-invariant matching. Use the **learningMode** parameter of `imaqLearnPattern3()` to specify which type of learning mode to use. If it is not necessary to learn the pattern of the entire template, you can provide a mask to `imaqLearnPattern3()`. The mask must be an image the same size as the template image. If you provide a mask to `imaqLearnPattern3()`, the algorithm only learns the pixels in the template that have corresponding pixels in the mask with a value of 0. Mask pixels with a value other than 0 are ignored. Provide a NULL mask to learn the entire template.

The learning process is usually time intensive because the algorithm attempts to find the optimum features of the template for the particular matching process. You can also save time by training the pattern matching algorithm offline, and then saving the template image with `imaqWriteVisionFile()`.

Defining a Search Area

Two equally important factors define the success of a pattern matching algorithm: accuracy and speed. You can define a search area to reduce ambiguity in the search process. For example, if your image has multiple instances of a pattern and only one of them is required for the inspection task, the presence of additional instances of the pattern can produce incorrect results. To avoid this, reduce the search area so that only the desired pattern lies within the search area.

The time required to locate a pattern in an image depends on both the template size and the search area. By reducing the search area or increasing the template size, you can reduce the required search time. Increasing the template size can improve the search time, but doing so reduces match accuracy if the larger template includes an excess of background information.

In many inspection applications, you have general information about the location of the fiducial. Use this information to define a search area. For example, in a typical component placement application, each printed circuit board (PCB) being tested may not be placed in the same location with the same orientation. The location of the PCB in various images can move and rotate within a known range of values, as illustrated in Figure 5-11. Figure 5-11a shows the template used to locate the PCB in the image. Figure 5-11b shows an image containing a PCB with a fiducial you want to locate. Notice the search area around the fiducial. If you know before the matching process begins that the PCB can shift or rotate in the image within a fixed range, then you can limit the search for the fiducial to a small region of the image. Figure 5-11c and Figure 5-11d show examples of a shifted fiducial and a rotated fiducial respectively.

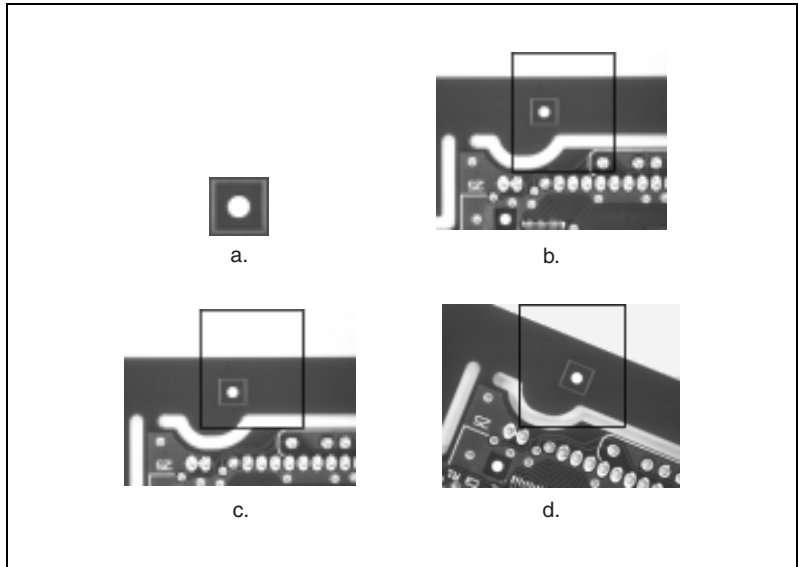


Figure 5-11. Selecting a Search Area for Grayscale Pattern Matching

Setting Matching Parameters and Tolerances

Every pattern matching algorithm makes assumptions about the images and pattern matching parameters used in machine vision applications. These assumptions work for a high percentage of the applications. However, there may be applications in which the assumptions used in the algorithm are not optimal. Knowing your particular application and the images you want to process is useful in selecting the pattern matching parameters. Use the `imaqMatchPattern2()` function to set the following elements that influence the NI Vision pattern matching algorithm: match mode, minimum contrast, and rotation angle ranges.

Match Mode

You can set the match mode to control how the pattern matching algorithm handles the template at different orientations. If you expect the orientation of valid matches to vary less than $\pm 5^\circ$ from the template, set the `mode` element of the **options** parameter to `IMAQ_MATCH_SHIFT_INVARIANT`. Otherwise, set the `mode` element to `IMAQ_MATCH_ROTATION_INVARIANT`.



Note Shift-invariant matching is faster than rotation-invariant matching.

Minimum Contrast

The pattern matching algorithm ignores all image regions in which contrast values fall below a set minimum contrast value. Contrast is the difference between the smallest and largest pixel values in a region. Set the `minContrast` element of the `imgMatchPattern2()` **options** parameter to slightly below the contrast value of the search area with the lowest contrast.

You can set the minimum contrast to potentially increase the speed of the pattern matching algorithm. If the search image has high contrast overall but contains some low contrast regions, set a high minimum contrast value to exclude all areas of the image with low contrast. Excluding these areas significantly reduces the area in which the pattern matching algorithm must search. However, if the search image has low contrast throughout, set a low minimum contrast to ensure that the pattern matching algorithm looks for the template in all regions of the image.

Rotation Angle Ranges

If you know that the pattern rotation is restricted to a certain range, such as between -15° to 15° , provide this restriction information to the pattern matching algorithm in the **angleRanges** element of the `imgMatchPattern2()` **options** parameter. This information improves your search time because the pattern matching algorithm looks for the pattern at fewer angles. Refer to the *Pattern Matching* section of the *NI Vision Concepts Help* for information about pattern matching.

Testing the Search Algorithm on Test Images

To determine if your selected template or reference pattern is appropriate for your machine vision application, test the template on a few test images by using `imgMatchPattern2()`. These test images should reflect the images generated by your machine vision application during true operating conditions. If the pattern matching algorithm locates the reference pattern in all cases, you have selected a good template. Otherwise, refine the current template, or select a better template until both training and testing are successful.

Using a Ranking Method to Verify Results

The manner in which you interpret the pattern matching algorithm depends on your application. For typical alignment applications, such as finding a fiducial on a wafer, the most important information is the position and location of the best match. Use the **position** and **corner** elements of the `PatternMatch` structure to get the position and the bounding rectangle of a match.

In inspection applications, such as optical character verification, the score of the best match is more useful. The score of a match returned by the pattern matching algorithm is an indicator of the closeness between the original pattern and the match found in the image. A high score indicates a very close match, while a low score indicates a poor match. The score can be used as a gauge to determine whether a printed character is acceptable. Use the **score** element of the **PatternMatch** structure to get the score corresponding to a match.

Finding Points Using Geometric Matching

The geometric matching algorithm in NI Vision 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. Geometric matching finds template matches regardless of lighting variation, blur, noise, occlusion, and geometric transformations such as shifting, rotation, or scaling of the template. Geometric matching returns the location of the center of the template, the template orientation, and the scale of the template.

NI Vision supports two types of geometric matching. Use `imaqMatchGeometricPattern3()` to perform edge-based geometric matching. Use `imaqMatchGeometricPattern2()` to perform feature-based geometric matching. Always start with edge-based geometric matching. If you cannot reach the performance or memory requirements of your application using edge-based geometric matching, and the object you need to match contains geometric features that can be reliably extracted, use the feature-based geometric matching algorithm. Refer to the *Geometric Matching* section of the *NI Vision Concepts Help* for more information about geometric matching algorithms.

Complete the following generalized steps to find features in an image using geometric matching.

1. Define a reference or fiducial pattern to use a template image.
2. Use the reference pattern to train the geometric matching algorithm with the NI Vision Template Editor. Go to **Start»All Programs»National Instruments»Vision»Template Editor** to launch the NI Vision Template Editor.
3. Define an image or an area of an image as the search area. A small search area can reduce the time to find the features.
4. Set the tolerances and parameters to specify how the algorithm operates at run time, and test the search algorithm on test images using `imaqMatchGeometricPattern3()`¹.

Defining and Creating Effective Template Images

The selection of an effective template image plays a critical part in obtaining good results. Because the template image represents the pattern that you want to find, make sure that all the important and unique characteristics of the pattern are well defined in the image.

The geometric matching algorithm is optimized for locating objects with good geometric information. Figure 5-12 shows examples of template images with good geometric or shape information.

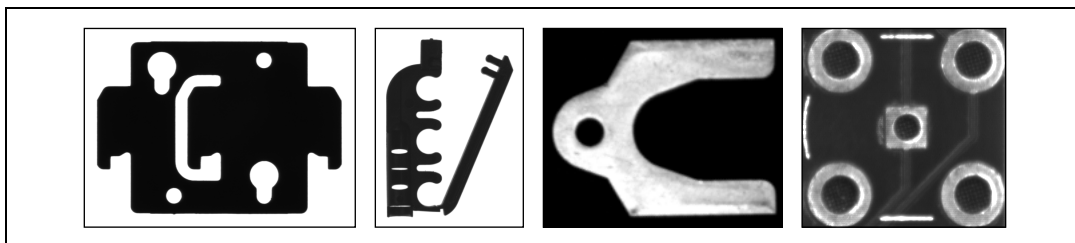


Figure 5-12. Examples of Objects on which Geometric Matching is Designed to Work

Geometric matching is not suited for template images that are predominantly defined by grayscale or texture information. Figure 5-13 shows examples of template images that do not have good geometric information. The template image in Figure 5-13a is characterized by the grayscale or texture information in the image. Figure 5-13b contains too many edges and will dramatically increase the time geometric matching takes to locate the template in an image.

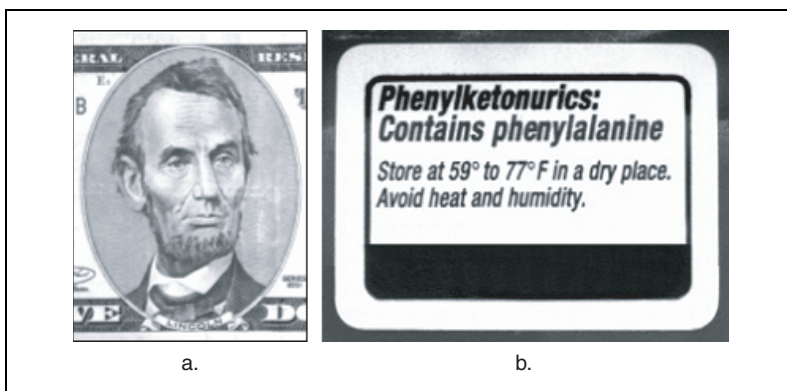


Figure 5-13. Examples of Objects for which Geometric Matching is Not Suited

¹ Use `imgMatchGeometricPattern2()` to test a feature-based template.

Training the Geometric Matching Algorithm

After you create a good template image, the geometric matching algorithm has to learn the important features of the template. Use the NI Vision Template Editor to select the important features in the template. Geometric matching uses the curves found in the template image as the basis for the features that are used for matching. Use the NI Vision Template Editor to remove noisy or unimportant curves from the template image and keep only those curves that are important to locate the template in the image.

The template image of a pacemaker is shown in Figure 5-14a. Figure 5-14b shows all the curves that are found in the template image. These images are comprised of curves that are good for matching, such as those along the boundary of the pacemaker, as well as noisy or unimportant curves, such as those found around the barcode and areas of reflection in the image. If you learn the template with the unimportant curves, the geometric matching algorithm will treat these curves as important features that must be found within a valid match region. You can make the matching process more reliable by removing these curves before learning the template. Figure 5-14c shows the curves that are used to learn the template after the unimportant curves have been removed using the NI Vision Template Editor.

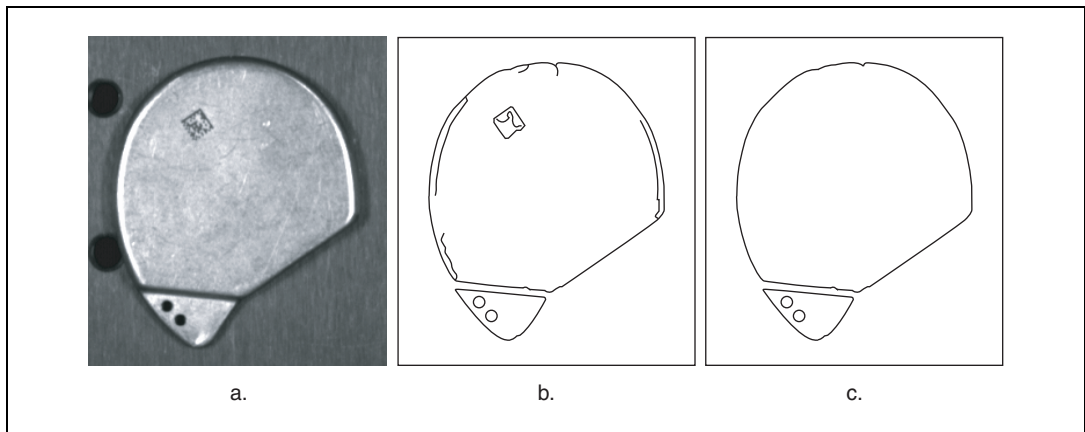


Figure 5-14. Selecting Good Features for Training a Template

Once the template image is trained, you can save the template to a file from within the NI Vision Template Editor. Use `imgqReadVisionFile()` to read the template file within your application.

Refer to the *NI Vision Template Editor Help* for more information about training the Geometric Matching algorithm.

Setting Matching Parameters and Tolerances

The geometric matching algorithm makes assumptions about the images and geometric matching parameters used in machine vision applications. These assumptions work for a high percentage of the applications.

However, there may be applications in which the assumptions used in the algorithm are not optimal. Knowing your particular application and the images you want to process is useful in selecting the pattern matching parameters. Use the **matchOptions** parameter to set the following parameters that influence the NI Vision geometric matching algorithm: match mode, rotation angle ranges, scale ranges, and occlusion ranges.

Match Mode

Set the match mode to control the conditions under which the geometric matching algorithm finds the template matches. If you expect the orientation of the valid matches to vary more than $\pm 5^\circ$ from the template, apply a bitwise OR operation to the **mode** element of the **matchOptions** parameter with `IMAQ_GEOMETRIC_MATCHING_ROTATION_INVARIANT`. If you expect the size of the valid matches to change more than $\pm 5\%$, apply a bitwise OR operation to the **mode** element of the **matchOptions** parameter with `IMAQ_GEOMETRIC_MATCHING_SCALE_INVARIANT`. If you expect the valid matches to be partially covered or missing, apply a bitwise OR operation to the **mode** element of the **matchOptions** parameter with `IMAQ_GEOMETRIC_MATCHING_OCCLUSION_INVARIANT`. Disabling any of these options or limiting their ranges decreases the search time.

Rotation Angle Ranges

If you know the rotation of a pattern is restricted to a certain range—for example, between -15° and 15° —provide this restriction information to the geometric matching algorithm in the **angleRanges** element of the **matchOptions** parameter. This information improves your search time because the geometric matching algorithm looks for the pattern at fewer angles.

Scale Factor Ranges

When the **mode** element contains

`IMAQ_GEOMETRIC_MATCHING_SCALE_INVARIANT`, geometric matching searches for occurrences of the template in the image regardless of whether valid matches are of a different size. The default scale range is 75% to 125%. If you know that the scale range of the valid matches is restricted to a certain range—for example, between 90% and 110%—provide this restriction information to the geometric matching algorithm by setting the **scaleRange** element of the **matchOptions** parameter.

Occlusion Ranges

When the **mode** element contains

`IMAQ_GEOMETRIC_MATCHING_OCCLUSION_INVARIANT`, geometric matching searches for occurrences of the template in the image, allowing for a specified percentage of the template to be occluded. The default occlusion range is 0% to 25%. If you know that the occlusion range of the valid matches is restricted to a certain range—for example, between 0% and 10%—provide this restriction information to the geometric matching algorithm by setting the **occlusionRange** element of the **matchOptions** parameter.

Refer to the *Geometric Matching* section of the *NI Vision Concepts Help* for more information about geometric matching.

Testing the Search Algorithm on Test Images

To determine if your selected template or reference pattern is appropriate for your machine vision application, test the template on a few test images by using `imaqMatchGeometricPattern3()`¹. These test images should reflect the images generated by your machine vision application during true operating conditions. If the geometric matching algorithm locates the reference pattern in all cases, you have selected a good template. Otherwise, refine the current template, or select a better template until both training and testing are successful.

Using Multiple Template Images

Complete the following steps if your application requires locating multiple template images in a target image.

1. Use the methods described in *Defining and Creating Effective Template Images* and *Training the Geometric Matching Algorithm* to create and train each template image.
2. Use the `imaqLearnMultipleGeometricPatterns()` function to combine the templates you trained in the previous step into a **MultipleGeometricPattern**.



Note Use `imaqWriteMultipleGeometricPatternFile()` to save the multiple geometric templates created by this step if you want to perform the learning process off-line. Use `imaqReadMultipleGeometricPatternFile()` to read the multiple geometric template file within your application during the matching process.

¹ Use `imaqMatchGeometricPattern2()` to test a feature-based template.

3. Use the methods described in [Setting Matching Parameters and Tolerances](#) to define how each template matches the target image.



Note When setting matching parameters and tolerances, pass the options to `imaqSetMultipleGeometricPatternsOptions()` instead of `imaqMatchGeometricPattern3()`.



Note If you save the multiple geometric template to file after you set the options, these options will be loaded within your application when you load the multiple geometric template file.

4. Test the multiple template search algorithm on test images using `imaqMatchMultipleGeometricPatterns()` to match all of the templates. The importance of this step is for the same reason explained in [Testing the Search Algorithm on Test Images](#).

Finding Points Using Color Pattern Matching

Color pattern matching algorithms provide a quick way to locate objects when color is present. Use color pattern matching if your images have the following qualities:

- The object you want to locate has color information that is very different from the background, and you want to find a very precise location of the object in the image.
- The object to locate has grayscale properties that are very difficult to characterize or that are very similar to other objects in the search image. In such cases, grayscale pattern matching can give inaccurate results. If the object has 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.

Color pattern matching returns the location of the center of the template and the template orientation. Complete the following general steps to find features in an image using color pattern matching:

1. Define a template image that contains a reference or fiducial pattern.
2. Use the reference pattern to train the color pattern matching algorithm with `imaqLearnColorPattern()`.
3. Define an image or an area of an image as the search area. A small search area reduces the time to find the features.
4. Set the **featureMode** element of the `imaqMatchColorPattern()` **options** parameter to `IMAQ_COLOR_AND_SHAPE_FEATURES`.

5. Set the tolerances and parameters to specify how the algorithm operates at run time using the **options** parameter of `imaqMatchColorPattern()`.
6. Test the search algorithm on test images using `imaqMatchColorPattern()`.
7. Verify the results using a ranking method.

Defining and Creating Good Color Template Images

The selection of a good template image plays a critical part in obtaining accurate results with the color pattern matching algorithm. Because the template image represents the color and the pattern that you want to find, make sure that all the important and unique characteristics of the pattern are well defined in the image.

Several factors are critical in creating a template image. These critical factors include color information, symmetry, feature detail, positional information, and background information. Refer to the *Defining and Creating Effective Template Images* section of this chapter for more information about some of these factors.

Color Information

A template with colors that are unique to the pattern provides better results than a template that contains many colors, especially colors found in the background or other objects in the image.

Symmetry

A rotationally symmetric template in the luminance plane is less sensitive to changes in rotation than a template that is rotationally asymmetric.

Feature Detail

A template with relatively coarse features is less sensitive to variations in size and rotation than a template with fine features. However, the template must contain enough detail to identify it.

Positional Information

A template with strong edges in both the x and y directions is easier to locate.

Background Information

Unique background information in a template improves search performance and accuracy during the grayscale pattern matching phase. This requirement could conflict with the color information requirement because background colors may not be desirable during the color location phase. Avoid this problem by choosing a template with sufficient background information for grayscale pattern matching while specifying the exclusion of the background color during the color location phase. Refer to the *Training the Color Pattern Matching Algorithm* section of this chapter for more information about how to ignore colors.

Training the Color Pattern Matching Algorithm

After you have created a good template image, the color pattern matching algorithm learns the important features of the template. Use `imaqLearnColorPattern()` to learn the template. The learning process depends on the type of matching that you expect to perform. By default, the color pattern matching algorithm learns only those features from the template that are necessary for shift-invariant matching. However, if you want to match the template at any orientation, the learning process must consider the possibility of arbitrary orientations. Use the **learnMode** element of the `imaqLearnColorPattern()` **options** parameter to specify which type of learning mode to use.

Exclude colors in the template that you are not interested in using during the search phase. Typically, you should ignore colors that either belong to the background of the object or are not unique to the template, to reduce the potential for incorrect matches during the color location phase. You can ignore certain predefined colors using the **ignoreMode** element of the **options** parameter. To ignore other colors, first learn the colors to ignore using `imaqLearnColor()`. Then set the **colorsToIgnore** element of the **options** parameter to the resulting `ColorInformation` structure from `imaqLearnColor()`.

The learning process is time-intensive because the algorithm attempts to find unique features of the template that allow for fast, accurate matching. However, you can train the pattern matching algorithm offline, and save the template image using `imaqWriteVisionFile()`.

Defining a Search Area

Two equally important factors define the success of a color pattern matching algorithm—accuracy and speed. You can define a search area to reduce ambiguity in the search process. For example, if your image has multiple instances of a pattern and only one instance is required for the inspection task, the presence of additional instances of the pattern can produce incorrect results. To avoid this, reduce the search area so that only the desired pattern lies within the search area. For example, in the fuse box inspection example, use the location of the fuses to be inspected to define the search area. Because the inspected fuse box may not be in the exact location or have the same orientation in the image as the previous one, the search area you define must be large enough to accommodate these variations in the position of the box. Figure 5-15 shows how you can select search areas for different objects.

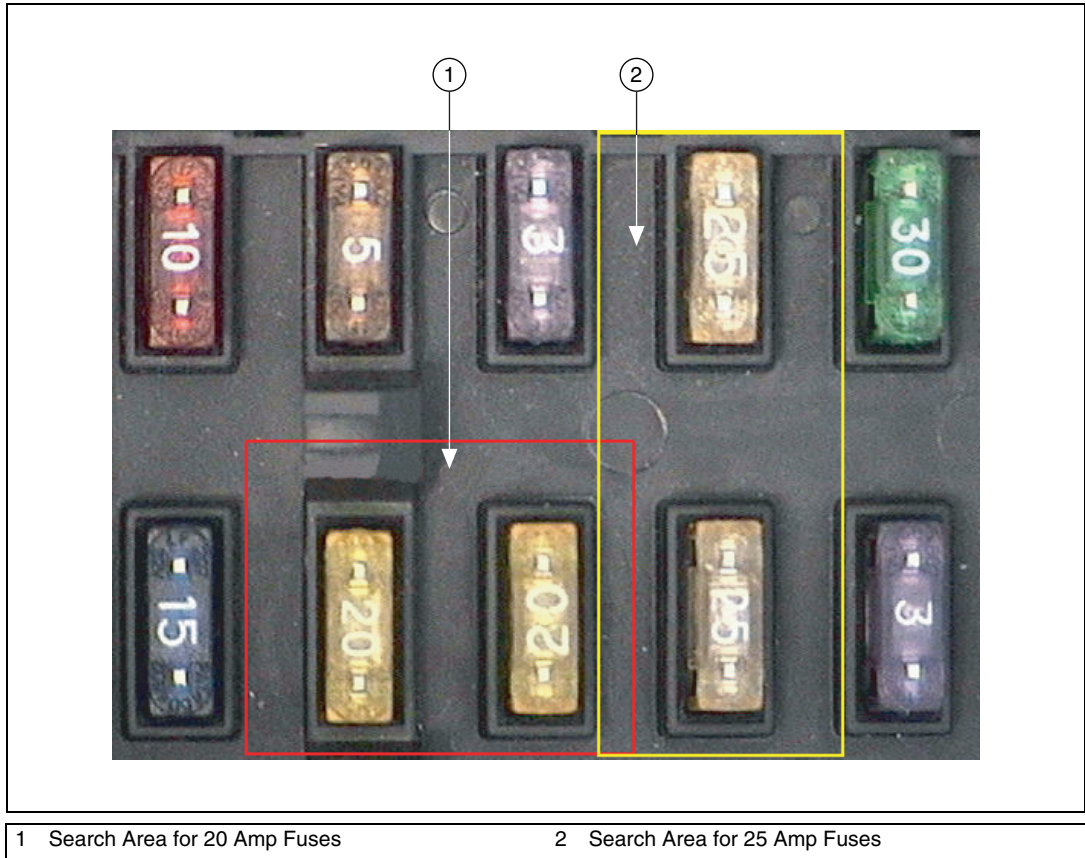


Figure 5-15. Selecting a Search Area for Color Pattern Matching

The time required to locate a pattern in an image depends on both the template size and the search area. By reducing the search area or increasing the template size, you can reduce the required search time. Increasing the template size can improve the search time, but doing so reduces match accuracy if the larger template includes an excess of background information.

Setting Matching Parameters and Tolerances

Every color pattern matching algorithm makes assumptions about the images and color pattern matching parameters used in machine vision applications. These assumptions work for a high percentage of the applications. However, there may be applications in which the assumptions used in the algorithm are not optimal. In such cases, you must modify the color pattern matching parameters. Knowing your particular application and the images you want to process is useful in selecting the pattern matching parameters. Use the **options** parameter of `imaqMatchColorPattern()` to set these elements.

The following are elements of the NI Vision pattern matching algorithm that influence color pattern matching: color sensitivity, search strategy, color score weight, ignore background colors, minimum contrast, and rotation angle ranges. These elements are discussed in the following sections.

Color Sensitivity

Use the **sensitivity** element to control the granularity of the color information in the template image. If the background and objects in the image contain colors that are very close to colors in the template image, use a higher color sensitivity setting. Increase the color sensitivity settings as the color differences decrease. Three color sensitivity settings are available in NI Vision: `IMAQ_SENSITIVITY_LOW`, `IMAQ_SENSITIVITY_MED`, and `IMAQ_SENSITIVITY_HIGH`. Refer to the *Color Inspection* section of the *NI Vision Concepts Help* for more information about color sensitivity.

Search Strategy

Use the **strategy** element to optimize the speed of the color pattern matching algorithm. The search strategy controls the step size, sub-sampling factor, and percentage of color information used from the template.

Choose from the following search strategies:

- `IMAQ_CONSERVATIVE`—Uses a very small step size, the least amount of subsampling, and all the color information present in the template. The conservative strategy is the most reliable method to look for a template in any image at potentially reduced speed.



Note Use the `IMAQ_CONSERVATIVE` strategy if you have multiple targets located very close to each other in the image.

- `IMAQ_BALANCED`—Uses values in between the `IMAQ_AGGRESSIVE` and `IMAQ_CONSERVATIVE` strategies.
- `IMAQ_AGGRESSIVE`—Uses a large step size, a lot of sub-sampling, and all of the color information from the template.
- `IMAQ_VERY_AGGRESSIVE`—Uses the largest step size, the most subsampling, and only the dominant color from the template to search for the template. Use this strategy when the color in the template is almost uniform, the template is well contrasted from the background, and there is a good amount of separation between different occurrences of the template in the image. This strategy is the fastest way to find templates in an image.

Color Score Weight

When you search for a template using both color and shape information, the color and shape scores generated during the match process are combined to generate the final color pattern matching score. The color score weight determines the contribution of the color score to the final color pattern matching score. If the color information of the templates is superior to its shape information, set the weight higher. For example, if you set `colorWeight` to 1000, the algorithm finds each match by using both color and shape information, and then ranks the matches based entirely on their color scores. If you set `colorWeight` to 0, the matches are still found using color and shape information, but they are ranked based entirely on their shape scores.

Minimum Contrast

Use the `minContrast` element to increase the speed of the color pattern matching algorithm. The color pattern matching algorithm ignores all image regions where grayscale contrast values fall beneath a set minimum contrast value. Refer to the [Setting Matching Parameters and Tolerances](#) section of this chapter for more information about minimum contrast.

Rotation Angle Ranges

Refer to the [Setting Matching Parameters and Tolerances](#) section of this chapter for information about rotation angle ranges.

Testing the Search Algorithm on Test Images

To determine if your selected template or reference pattern is appropriate for your machine vision application, test the template on a few test images by using `imaqMatchColorPattern()`. These test images reflect the images generated by your machine vision application during true operating conditions. If the pattern matching algorithm locates the reference pattern in all cases, you have selected a good template. Otherwise, refine the current template, or select a better template until both training and testing are successful.

Finding Points Using Color Location

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 the color arrangement in the region
- Does not require the orientation of the region
- Does not always require the location with sub-pixel accuracy
- Does not require shape information for the region

Complete the following general steps to find features in an image using color location.

1. Define a reference pattern in the form of a template image.
2. Use the reference pattern to train the color location algorithm with `imaqLearnColorPattern()`.
3. Define an image or an area of an image as the search area. A small search area reduces the time to find the features.
4. Set the `featureMode` element of the `imaqMatchColorPattern()` **options** parameter to `IMAQ_COLOR_FEATURES`.
5. Set the tolerances and parameters to specify how the algorithm operates at run time using the **options** parameter of `imaqMatchColorPattern()`.

6. Test the color location algorithm on test images using `imaqMatchColorPattern()`.
7. Verify the results using a ranking method.

You can save the template image using `imaqWriteVisionFile()`.

Convert Pixel Coordinates to Real-World Coordinates

The measurement points you located with edge detection and pattern matching are in pixel coordinates. If you need to make measurements using real-world units, use `imaqTransformPixelToRealWorld()` to convert the pixel coordinates into real-world units.

Make Measurements

You can make different types of measurements either directly from the image or from points that you detect in the image.

Distance Measurements

Use the following functions to make distance measurements for your inspection application.

Clamp functions measure the separation between two edges in a rectangular search region. First, clamp functions detect points along the two edges using the Rake function. Then, they compute the distance between the points detected on the edges along each search line of the rake and return the largest or smallest distance. The `imaqSelectRect()` function generates a valid input search region for these functions. You also need to specify the parameters for edge detection and the separation between the search lines that you want to use within the search region to find the edges. These functions work directly on the image under inspection, and they output the coordinates of all the edge points that they find. The following list describes the available clamp functions:

- `imaqClampMax()`—Measures the largest separation between two edges in a rectangular search region.
- `imaqClampMin()`—Finds the smallest separation between two edges.

Use `imaqGetDistance()` to compute the distances between two points, such as consecutive pairs of points in an array of points. You can obtain these points from the image using any one of the feature detection methods described in the [Find Measurement Points](#) section of this chapter.

Analytic Geometry Measurements

Use the following functions to make geometrical measurements from the points you detect in the image:

- `imaqFitLine()` — Fits a line to a set of points and computes the equation of the line.
- `imaqFitCircle2()` — Fits a circle to a set of at least three points and computes its area, perimeter, and radius.
- `imaqFitEllipse2()` — Fits an ellipse to a set of at least six points and computes its area, perimeter, and the lengths of its major and minor axis.
- `imaqGetIntersection()` — Finds the intersection point of two lines specified by their start and end points.
- `imaqGetAngle()` — Finds the smaller angle between two lines.
- `imaqGetPerpendicularLine()` — Finds the perpendicular line from a point to a line and computes the perpendicular distance between the point and the line.
- `imaqGetBisectingLine()` — Finds the line that bisects the angle formed by two lines.
- `imaqGetMidLine()` — Finds the line that is midway between a point and a line and is parallel to the line.
- `imaqGetPolygonArea()` — Calculates the area of a polygon specified by its vertex points.

Instrument Reader Measurements

You can make measurements based on the values obtained by meter and LCD readers.

Use `imaqGetMeterArc()` to calibrate a meter or gauge that you want to read. The `imaqGetMeterArc()` function calibrates the meter using one of two modes. The `IMAQ_METER_ARC_ROI` mode uses the initial position and the full-scale position of the needle. When using this mode, the function calculates the position of the base of the needle and the arc traced by the tip of the needle. The `IMAQ_METER_ARC_POINTS` mode calibrates the meter using three points on the meter: the base of the needle, the tip of the needle at its initial position, and the tip of the needle at its full-scale position. When using this mode, the function calculates the position of the points along the arc covered by the tip of the needle. Use `imaqReadMeter()` to read the position of the needle using the base of the needle and the array of points on the arc traced by the tip of the needle.

Use `imgFindLCDSegments()` to calculate the ROI around each digit in an LCD or LED. To find the area of each digit, all the segments of the indicator must be activated. Use `imgReadLCD()` to read multiple digits of an LCD or LED.

Identify Parts Under Inspection

In addition to making measurements after you set regions of inspection, you can also identify parts using classification, optical character recognition (OCR), and barcode reading.

Classifying Samples

Use classification to identify unknown samples based on their color or shape by comparing a set of significant characteristics to a set of characteristics that conceptually represent a class of known samples.

Color Classification

Typical applications involving color classification include the following:

- **Sorting**—For example, using the color of the liquid in a bottle to sort bottles on a conveyor belt into different boxes.
- **Inspection**—For example, using the color of fruit to determine if it is ripe.

Before you classify colors, you must create a classifier file with samples of the colors using the NI Color Classification Training Interface.

Go to **Start»All Programs»National Instruments»Color Classification Training** to launch the NI Color Classification Training Interface. Refer to the *NI Color Classification Training Interface Help* for more information about the NI Color Classification Training Interface.

Particle Classification

Use particle classification to identify an unknown object by comparing a set of its significant features to a set of features that conceptually represent a class of known objects. Typical applications involving particle classification include the following:

- **Sorting**—Sorts objects of varied shapes. For example, sorting different mechanical parts on a conveyor belt into different bins.
- **Inspection**—Inspects objects by assigning each object an identification score and then rejecting objects that do not closely match members of the training set.

Before you classify objects, you must train the particle classifier session with samples of the objects using the NI Particle Classification Training Interface. Go to **Start»All Programs»National Instruments»Particle Classification Training** to launch the NI Particle Classification Training Interface. Refer to the *NI Particle Classification Training Interface Help* for more information about the NI Particle Classification Training Interface.

Performing Classification

After you have trained samples of the objects or colors that you want to classify, use the following functions to perform classification:

1. Use `imaqReadClassifierFile()` to read in a classifier session that you created using the NI Particle Classification Training Interface or the NI Color Classification Training Interface.
2. Use `imaqClassify()` to classify the sample inside the ROI of the image under inspection into one of the classes you created using the NI Particle Classification Training Interface.
3. Use `imaqDispose()` to free the resources that the classifier session used.

The following code sample provides an example of a typical classification application.

```
ClassifierSession* session;
Image* image;
ROI* roi;
char* fileName; // The classifier file to use.
ClassifierReport* report;

session = imaqReadClassifierFile(NULL, fileName,
IMAQ_CLASSIFIER_READ_ALL, NULL, NULL, NULL);
while (stillClassifying)
{
    // Acquire and process an image and store it in the
    //image variable.
    // Locate the object to classify, and store an ROI
    //containing that object in the roi variable.
    report = imaqClassify(image, session, roi, NULL, 0);
    // Take action based on the report.
    imaqDispose(report);
}
imaqDispose(session);
```

Reading Characters

Use OCR to read text and/or characters in an image. Typical uses for OCR in an inspection application include identifying or classifying components.

Before you read text and/or characters in an image, you must train the OCR Session with samples of the characters using the NI OCR Training Interface. Go to **Start»All Programs»National Instruments»Vision»OCR Training** to launch the NI OCR Training Interface.

Refer to the *NI OCR Training Interface Help* for more information about the NI OCR Training Interface.

After you have trained samples of the characters you want to read, use the following functions to read the characters:

1. Use `imaqReadOCRFile()` to read in a session that you created using the NI OCR Training Interface.
2. Use `imaqReadText3()` to read the characters inside the ROI of the image under inspection.
3. Use `imaqDispose()` to free the resources that the OCR Session used.

Reading Barcodes

Use barcode reading functions to read values encoded into 1D barcodes, Data Matrix codes, QR codes and PDF417 codes.

Reading 1D Barcodes

To read a 1D barcode, locate the barcode in the image using one of the techniques described in this chapter. Then pass the ROI Descriptor of the location into `imaqReadBarcode()`.

Use `imaqReadBarcode()` to read values encoded in the 1D barcode. Specify the type of 1D barcode in the application using the **type** parameter. NI Vision supports the following 1D barcode types: Codabar, Code 39, Code 93, Code 128, EAN 8, EAN 13, Interleaved 2 of 5, MSI, UPCA, GS1 DataBar Limited (previously referred to as RSS-14 Limited), and Pharmacode.

Reading Data Matrix Codes

Use `imaqReadDataMatrixBarcode2()` to read values encoded in a Data Matrix code. The function can determine automatically the appropriate search options for your application. However, you can improve the performance of the application by specifying parameter values specific to your application.

`imaqReadDataMatrixBarcode2()` can locate automatically the Data Matrix code in an image. However, you can improve the inspection performance by locating the code using one of the techniques described in this chapter. Then pass the ROI indicating the location into `imaqReadDataMatrixBarcode2()`.

Reading QR Codes

Use `imaqReadQRCode()` to read values encoded in a QR or micro-QR code. The function can determine automatically the appropriate search options for your application. However, you can improve the performance of the application by specifying parameter values specific to your application.

`imaqReadQRCode()` can locate automatically the QR code in an image. However, you can improve the inspection performance by locating the code using one of the techniques described in this chapter. Then pass the ROI indicating the location into `imaqReadQRCode()`.

Reading PDF417 Codes

Use `imaqReadPDF417Barcode()` to read values encoded in a PDF417 code.

`imaqReadPDF417Barcode()` can locate automatically one or multiple PDF417 codes in an image. However, you can improve the inspection performance by locating the codes using one of the techniques described in this chapter. Then pass in the ROI location to `imaqReadPDF417Barcode()`.



Tip If you need to read only one PDF417 code per image, set the **searchMode** parameter to `IMAQ_SEARCH_SINGLE_CONSERVATIVE` to increase the speed of your application.

Inspect Image for Defects

In addition to identifying parts under inspection, you can also compare images to a golden template to inspect your image based on differences in intensity and use character verification to verify characters in your image.

Compare to Golden Template

Use `imaqCompareGoldenTemplate()` to inspect your image based on differences in intensity. Use the NI Vision Template Editor to learn golden templates for inspection. Go to **Start»All Programs»National Instruments»Vision»Template Editor** to launch the NI Vision Template Editor. Use the methods described in Chapter 4, *Performing Particle Analysis*, to analyze the resulting binary image.

Verify Characters

Use character verification functions to verify characters in your image.

Before you verify text, you must train the OCR Session with samples of the characters using the NI OCR Training Interface. Go to **Start»All Programs»National Instruments»Vision»OCR Training** to launch the NI OCR Training Interface. You must then designate reference characters for each of the character classes in your character set. The characters you want to verify are compared against these reference characters and are assigned a score based on this comparison.

Refer to the *NI OCR Training Interface Help* for more information about training reference characters.

After you have trained the samples of the characters and assigned the reference character, use the following functions to verify the characters.

1. Use `imaqReadOCRFile()` to read in a session that you created using the NI OCR Training Interface.
2. Use `imaqVerifyText()` to verify the characters inside the ROI(s) of the image under inspection.
3. Use `imaqDispose()` to free the resources that the OCR Session used.

Display Results

You can overlay the results obtained at various stages of your inspection process on the window that displays your inspection image. The software attaches the information that you want to overlay to the image, but it does not modify the image. You also can group similar types of overlay information together to allow multiple overlays to act as a single overlay. The overlay appears every time you display the image in an external window.

Use the following functions to set or view the properties of image overlays.

- `imaqSetOverlayProperties()`—Configures the overlay properties for an image.
- `imaqGetOverlayProperties()`—Retrieves the overlay properties for an image.

Use the following functions to overlay search regions, inspection results, and other information, such as text and bitmaps.

- `imaqOverlayPoints()`—Overlays points on an image. Specify a point by its x-coordinate and y-coordinate.
- `imaqOverlayLine()`—Overlays a line on an image. Specify a line by its start and end points.
- `imaqOverlayRect()`—Overlays a rectangle on an image.
- `imaqOverlayOval()`—Overlays an oval or a circle on the image.
- `imaqOverlayArc()`—Overlays an arc on the image.
- `imaqOverlayMetafile()`—Overlays a metafile on the image.
- `imaqOverlayText()`—Overlays text on an image.
- `imaqOverlayROI()`—Overlays an ROI on an image.
- `imaqOverlayClosedContour()`—Overlays a closed contour on an image.
- `imaqOverlayOpenContour()`—Overlays an open contour on an image.

To use these functions, pass in the image on which you want to overlay information and the information that you want to overlay.



Tip You can select the color of overlays with these functions.

You can configure the following processing functions to overlay different types of information on the inspection image:

- `imaqFindEdge2()`
- `imaqFindCircularEdge()`
- `imaqFindConcentricEdge()`
- `imaqClampMax()`
- `imaqClampMin()`
- `imaqFindPattern()`
- `imaqCountObjects()`
- `imaqFindTransformRect2()`
- `imaqFindTransformRects2()`
- `imaqFindTransformPattern()`

The following list contains the kinds of information you can overlay with the previous functions except `imaqFindPattern()`, `imaqCountObjects()`, and `imaqFindTransformPattern()`.

- The search area input into the function
- The search lines used for edge detection
- The edges detected along the search lines
- The result of the function

With `imaqFindPattern()`, `imaqCountObjects()`, and `imaqFindTransformPattern()`, you can overlay the search area and the result. Select the information you want to overlay by setting the element that corresponds to the information type to `TRUE` in the **options** input parameter.

Use `imaqClearOverlay()` to clear any previous overlay information from the image. Use `imaqWriteVisionFile()` to save an image with its overlay information to a file. You can read the information from the file into an image using `imaqReadVisionFile()`.



Note Use `imaqSetOverlayProperties()` to specify the behavior of overlay information when the image size or orientation changes. By default, calibration and overlay information is removed from an image when the image size or orientation changes.

Calibrating Images

This chapter describes how to calibrate your imaging system, save calibration information, and attach calibration information to an image.

After you set up your imaging system, you may want to calibrate your system. If your imaging setup is such that the camera axis is perpendicular or nearly perpendicular to the object under inspection and your lens has no distortion, use simple calibration. With simple calibration, you do not need to learn a template. Instead, you define the distance between pixels in the horizontal and vertical directions using real-world units.

If your camera axis is not perpendicular to the object under inspection, use perspective calibration to calibrate your system. If your lens is distorted, use nonlinear distortion calibration.

Perspective and Nonlinear Distortion Calibration

Perspective errors and lens aberrations cause images to appear distorted. This distortion misplaces information in an image, but it does not necessarily destroy the information in the image. Calibrate your imaging system if you need to compensate for perspective errors or nonlinear lens distortion.

Complete the following general steps to calibrate your imaging system:

1. Define a calibration template.
2. Define a reference coordinate system.
3. Learn the calibration information.

After you calibrate your imaging system, you can attach the calibration information to an image. Refer to the [Attach Calibration Information](#) section of this chapter for more information. Then, depending on your needs, you can do one of the following:

- Use the calibration information to convert pixel coordinates to real-world coordinates without correcting the image.
- Create a distortion-free image by correcting the image for perspective errors and lens aberrations.

Refer to Chapter 5, *Performing Machine Vision Tasks*, for more information about applying calibration information before making measurements.

Defining a Calibration Template

You can define a calibration template by supplying an image of a grid or providing a list of pixel coordinates and their corresponding real-world coordinates. This section discusses the grid method in detail.

A calibration template is a user-defined grid of circular dots. As shown in Figure 6-1, the grid has constant spacings in the x and y directions. You can use any grid, but follow these guidelines for best results:

- The displacement in the x and y directions should be equal ($dx = dy$).
- The dots should cover the entire desired working area.
- The radius of the dots in the acquired image should be 6–10 pixels.
- The center-to-center distance between dots in the acquired image should range from 18 to 32 pixels, as shown in Figure 6-1.
- The minimum distance between the edges of the dots in the acquired image should be 6 pixels, as shown in Figure 6-1.

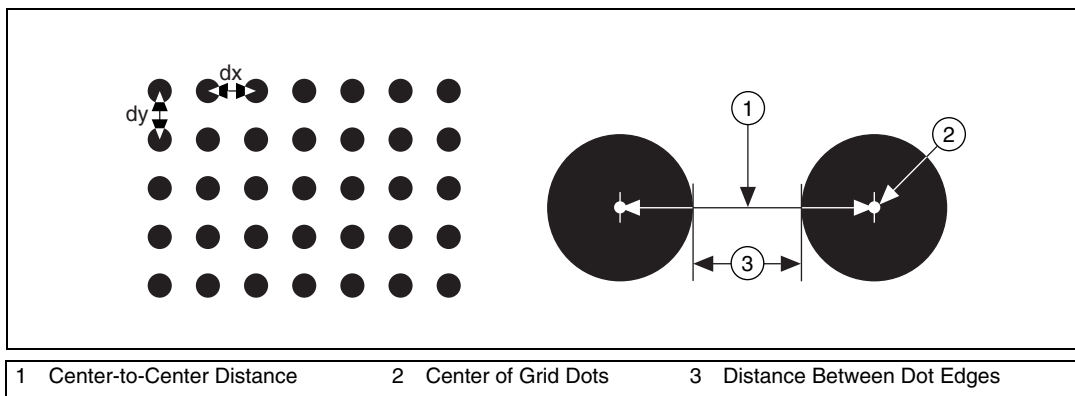


Figure 6-1. Defining a Calibration Grid



Note Click **Start»All Programs»National Instruments»Vision»Documentation»Calibration Grid** to use the calibration grid installed with NI Vision. The dots have radii of 2 mm and center-to-center distances of 1 cm. Depending on your printer, these measurements may change by a fraction of a millimeter. You can purchase highly accurate calibration grids from optics suppliers, such as Edmund Industrial Optics.

Defining a Reference Coordinate System

To express measurements in real-world units, you need to define a coordinate system in the image of the grid. Use the `CoordinateSystem` structure to define a coordinate system by its origin, angle, and axis direction.

The origin, expressed in pixels, defines the center of your coordinate system. The angle specifies the orientation of your coordinate system with respect to the angle of the topmost row of dots in the grid image. The calibration procedure automatically determines the direction of the horizontal axis in the real world. The vertical axis direction can either be indirect, as shown in Figure 6-2a, or direct, as shown in Figure 6-2b.

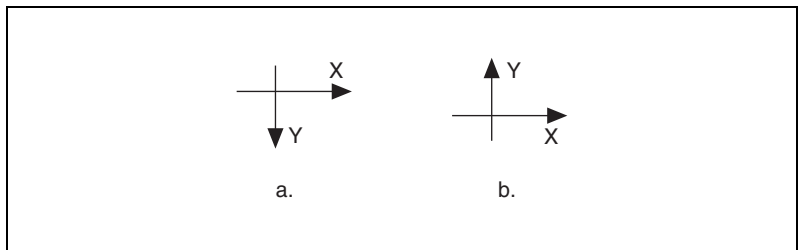


Figure 6-2. Axis Direction in the Image Plane

If you do not specify a coordinate system, the calibration process defines a default coordinate system. If you specify a grid for the calibration process, the software defines the following default coordinate system, as shown in Figure 6-3:

1. The origin is placed at the center of the left, topmost dot in the calibration grid.
2. The angle is set to 0° . This aligns the x-axis with the first row of dots in the grid, as shown in Figure 6-3a.
3. The axis direction is set to indirect. This aligns the y-axis to the first column of the dots in the grid, as shown in Figure 6-3b.

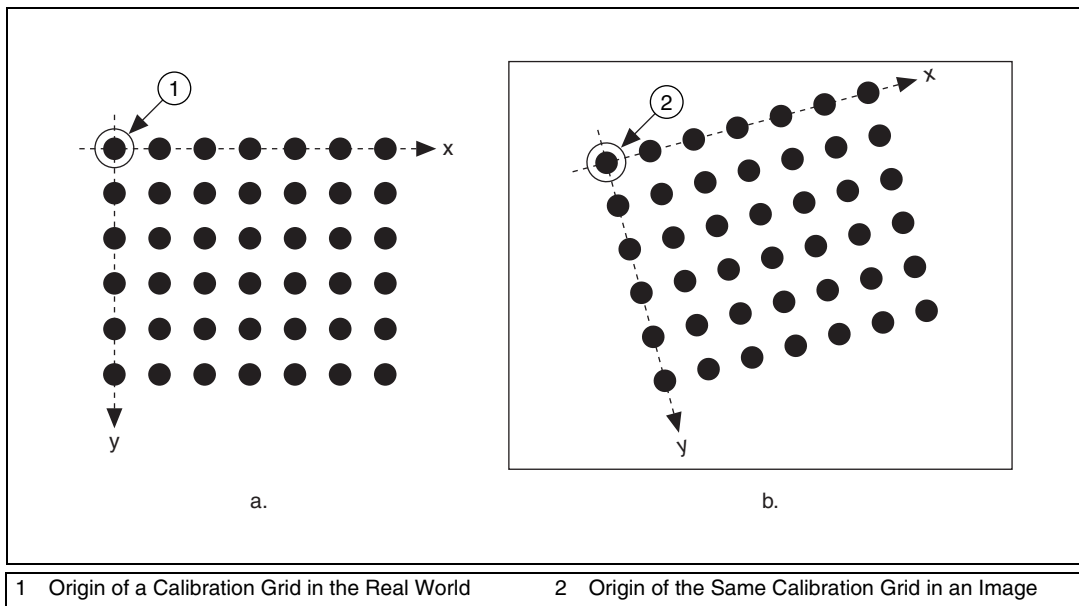


Figure 6-3. A Calibration Grid and an Image of the Grid



Note If you specify a list of points instead of a grid for the calibration process, the software defines a default coordinate system, as follows:

1. The origin is placed at the point in the list with the lowest x-coordinate value and then the lowest y-coordinate value.
2. The angle is set to 0° .
3. The axis direction is set to indirect.

If you define a coordinate system yourself, carefully consider the needs of your application.

- Express the origin in pixels. Always choose an origin location that lies within the calibration grid so that you can convert the location to real-world units.
- Specify the angle as the angle between the x-axis of the new coordinate system (x') and the top row of dots (x), as shown in Figure 6-4. If your imaging system exhibits nonlinear distortion, you cannot visualize the angle as you can in Figure 6-4 because the dots do not appear in straight lines.

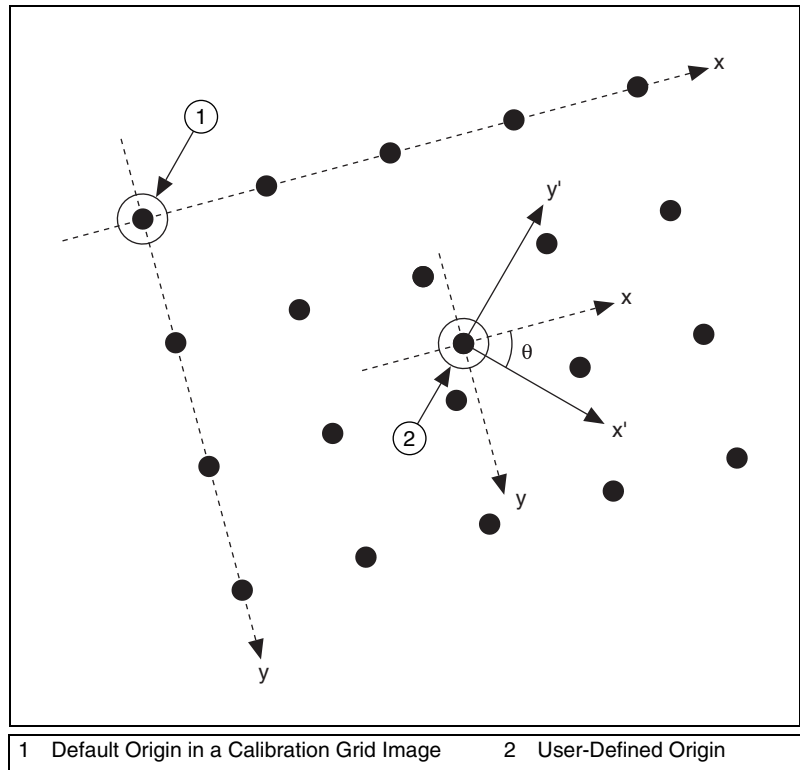


Figure 6-4. Defining a Coordinate System

Learning Calibration Information

After you define a calibration grid and reference axis, acquire an image of the grid using the current imaging setup. For information about acquiring images, refer to the [Acquire or Read an Image](#) section of Chapter 2, [Getting Measurement-Ready Images](#). The grid does not need to occupy the entire image. You can choose a region within the image that contains the grid. After you acquire an image of the grid, learn the calibration information by inputting the image of the grid into `imaqLearnCalibrationGrid()`.



Note If you want to specify a list of points instead of a grid, use `imaqLearnCalibrationPoints()` to learn the calibration information. Use the `CalibrationPoints` structure to specify the pixel to real-world mapping.

Specifying Scaling Factors

Scaling factors are the real-world distances between the dots in the calibration grid in the x and y directions and the units in which the distances are measured. Use the `GridDescriptor` structure to specify the scaling factors.

Choosing a Region of Interest

Define a learning ROI during the learning process to define a region of the calibration grid you want to learn. The software ignores dot centers outside this region when it estimates the transformation. Depending on the other calibration options selected, this is an effective way to increase correction speeds. Set the user-defined ROI using the **roi** parameter of either `imgLearnCalibrationGrid()` or `imgLearnCalibrationPoints()`.



Note The user-defined ROI represents the area in which you are interested. The learning ROI is different from the calibration ROI generated by the calibration algorithm. Refer to Figure 6-6 for an illustration of calibration ROIs.

Choosing a Learning Algorithm

Select a method in which to learn the calibration information: perspective projection or nonlinear. Figure 6-5 illustrates the types of errors your image can exhibit. Figure 6-5a shows an image of a calibration grid with no errors. Figure 6-5b shows an image of a calibration grid with perspective projection. Figure 6-5c shows an image of a calibration grid with nonlinear distortion.

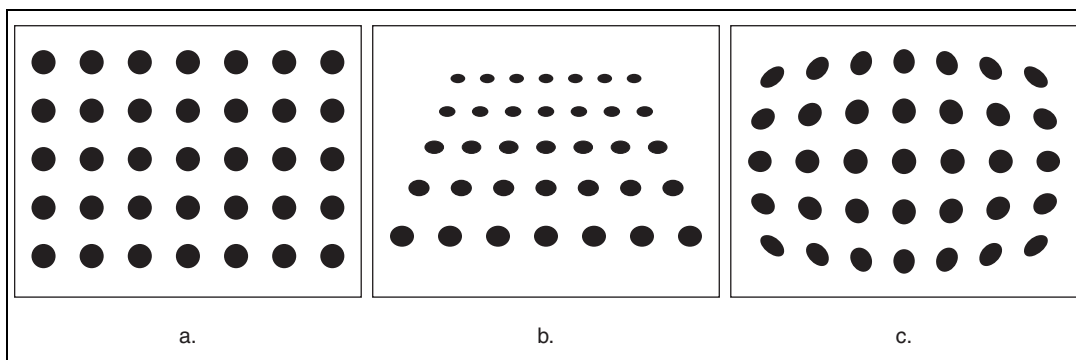


Figure 6-5. Types of Image Distortion

Choose the perspective projection algorithm when your system exhibits perspective errors only. A perspective projection calibration has an accurate transformation even in areas not covered by the calibration grid, as shown in Figure 6-6. Set the `mode` element of the **options** parameter to `IMAQ_PERSPECTIVE` to choose the perspective calibration algorithm. Learning and applying perspective projection is less computationally intensive than the nonlinear method. However, perspective projection cannot handle nonlinear distortions.

If your imaging setup exhibits nonlinear distortion, use the nonlinear method. The nonlinear method guarantees accurate results only in the area that the calibration grid covers, as shown in Figure 6-6. If your system exhibits both perspective and nonlinear distortion, use the nonlinear method to correct for both. Set the `mode` element of the **options** parameter to `IMAQ_NONLINEAR` to choose the nonlinear calibration algorithm.

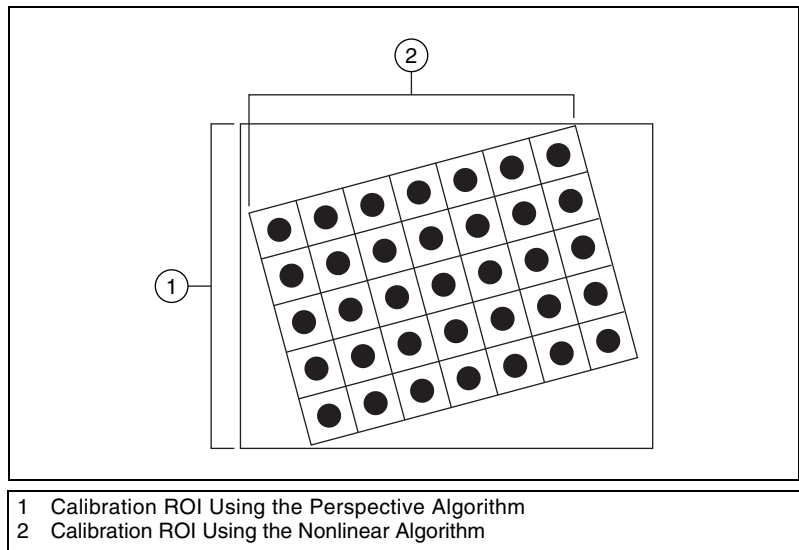


Figure 6-6. Calibration ROIs

Using the Learning Score

The learning process returns a score that reflects how well the software learned the input image. A learning score above 800 indicates that you chose the appropriate learning algorithm, that the grid image complies with the guideline, and that your vision system setup is adequate.



Note A high score does not reflect the accuracy of your system.

If the learning process returns a learning score below 600, try the following:

1. Make sure your grid complies with the guidelines listed in the [Defining a Calibration Template](#) section of this chapter.
2. Check the lighting conditions. If you have too much or too little lighting, the software may estimate the center of the dots incorrectly. Also, adjust the **range** parameter to distinguish the dots from the background.
3. Select another learning algorithm. When nonlinear lens distortion is present, using perspective projection sometimes results in a low learning score.

Learning the Error Map

An error map helps you gauge the quality of your complete system. The error map returns an estimated error range to expect when a pixel coordinate is transformed into a real-world coordinate. The transformation accuracy may be higher than the value the error range indicates. Set the `learnMap` element of the **options** parameter to `TRUE` to learn the error map.

Learning the Correction Table

If the speed of image correction is a critical factor for your application, use 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 extra memory requirements for this option are based on the size of the image. Use this option when you want to correct several images at a time in your vision application. Set the `learnTable` element of the **options** parameter to `TRUE` to learn the correction table.

Setting the Scaling Method

Use the `method` element of the **options** parameter to choose the appearance of the corrected image. Select either `IMAQ_SCALE_TO_FIT` or `IMAQ_SCALE_TO_PRESERVE_AREA`. Refer to the *System Setup and Calibration* section of the *NI Vision Concepts Help* for more information about the scaling methods.

Calibration Invalidation

Any image processing operation that changes the image size or orientation voids the calibration information in a calibrated image. Examples of functions that void calibration information include `imaqResample()`, `imaqScale()`, `imaqArrayToImage()`, and `imaqUnwrap()`.

Simple Calibration

When the axis of your camera is perpendicular to the image plane and lens distortion is negligible, use simple calibration. In simple calibration, a pixel coordinate is transformed to a real-world coordinate through scaling in the horizontal and vertical directions.

Use simple calibration to map pixel coordinates to real-world coordinates directly without a calibration grid. The software rotates and scales a pixel coordinate according to predefined coordinate reference and scaling factors. You can assign the calibration to an arbitrary image using `imaqSetSimpleCalibration()`.

To perform a simple calibration, set a coordinate reference (angle, center, and axis direction) and scaling factors on the defined axis, as shown in Figure 6-7. Express the angle between the x-axis and the horizontal axis of the image in degrees. Express the center as the position, in pixels, where you want the coordinate reference origin. Set the axis direction to direct or indirect. Simple calibration also offers a correction table option and a scaling mode option.

Use the **system** parameter to define the coordinate system. Use the **grid** parameter to specify the scaling factors. Use the **method** parameter to set the scaling method. Set the **learnTable** parameter to `TRUE` to learn the correction table.

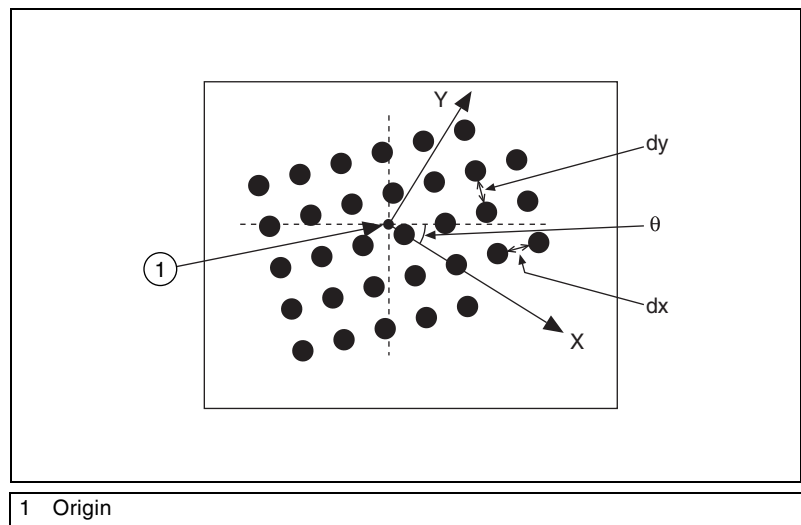


Figure 6-7. Defining a Simple Calibration

Save Calibration Information

After you learn the calibration information, you can save it so that you do not have to relearn the information for subsequent processing. Use `imaqWriteVisionFile()` to save the image of the grid and its associated calibration information to a file. To read the file containing the calibration information use `imaqReadVisionFile()`. Refer to the *Attach Calibration Information* section of this chapter for more information about attaching the calibration information you read from another image.

Attach Calibration Information

Now that you have calibrated your setup correctly, you can apply the calibration settings to images that you acquire. Use `imaqCopyCalibrationInfo2()` to attach the calibration information of the current setup to each image you acquire. This function takes in a source image containing the calibration information and a destination image that you want to calibrate. The destination image is your inspection image with the calibration information attached to it.

Using the calibration information attached to the image, you can accurately convert pixel coordinates to real-world coordinates to make any of the analytic geometry measurements with `imaqTransformPixelToRealWorld()`. If your application requires that you make shape measurements, correct the image by removing distortion with `imaqCorrectImage()`.



Note Correcting images is a time-intensive operation.

A calibrated image is not the same as a corrected image. Because calibration information is part of the image, it is propagated throughout the processing and analysis of the image. Functions that modify the image size, such as an image rotation function, void the calibration information. Use `imaqWriteVisionFile()` to save the image and all of the attached calibration information to a file.

Technical Support and Professional Services

Visit the following sections of the award-winning National Instruments Web site at ni.com for technical support and professional services:

- **Support**—Technical support at ni.com/support includes the following resources:
 - **Self-Help Technical Resources**—For answers and solutions, visit ni.com/support for software drivers and updates, a searchable KnowledgeBase, product manuals, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on. Registered users also receive access to the NI Discussion Forums at ni.com/forums. NI Applications Engineers make sure every question submitted online receives an answer.
 - **Standard Service Program Membership**—This program entitles members to direct access to NI Applications Engineers via phone and email for one-to-one technical support as well as exclusive access to on demand training modules via the Services Resource Center. NI offers complementary membership for a full year after purchase, after which you may renew to continue your benefits.

For information about other technical support options in your area, visit ni.com/services, or contact your local office at ni.com/contact.

- **Training and Certification**—Visit ni.com/training for self-paced training, eLearning virtual classrooms, interactive CDs, and Certification program information. You also can register for instructor-led, hands-on courses at locations around the world.
- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, National Instruments Alliance Partner members can help. To learn more, call your local NI office or visit ni.com/alliance.

If you searched ni.com and could not find the answers you need, contact your local office or NI corporate headquarters. Phone numbers for our worldwide offices are listed at the front of this manual. You also can visit the Worldwide Offices section of ni.com/niglobal to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

Glossary

Numbers

1D	One-dimensional.
2D	Two-dimensional.
3D	Three-dimensional.

A

AIPD	The National Instruments internal image file format used for saving complex images and calibration information associated with an image (extension APD).
alignment	The process by which a machine vision application determines the location, orientation, and scale of a part being inspected.
alpha channel	The channel used to code extra information, such as gamma correction, about a color image. The alpha channel is stored as the first byte in the four-byte representation of an RGB pixel.
area	(1) A rectangular portion of an acquisition window or frame that is controlled and defined by software. (2) The size of an object in pixels or user-defined units.
arithmetic operators	The image operations multiply, divide, add, subtract, and modulo.
array	An ordered, indexed set of data elements of the same type.
auto-median function	A function that uses dual combinations of opening and closing operations to smooth the boundaries of objects.

B

b	Bit. One binary digit, either 0 or 1.
B	Byte. Eight related bits of data, an eight-bit binary number. Also denotes the amount of memory required to store one byte of data.
barycenter	The grayscale value representing the centroid of the range of an image's grayscale values in the image histogram.
binary image	An image in which the objects usually have a pixel intensity of 1 (or 255) and the background has a pixel intensity of 0.
binary morphology	Functions that perform morphological operations on a binary image.
binary threshold	The separation of an image into objects of interest (assigned a pixel value of 1) and background (assigned pixel values of 0) based on the intensities of the image pixels.
bit depth	The number of bits (n) used to encode the value of a pixel. For a given n , a pixel can take 2^n different values. For example, if n equals 8, a pixel can take 256 different values ranging from 0 to 255. If n equals 16, a pixel can take 65,536 different values ranging from 0 to 65,535 or $-32,768$ to $32,767$.
blurring	Reduces the amount of detail in an image. Blurring commonly occurs because the camera is out of focus. You can blur an image intentionally by applying a lowpass frequency filter.
BMP	Bitmap. An image file format commonly used for 8-bit and color images. BMP images have the file extension <i>BMP</i> .
border function	Removes objects (or particles) in a binary image that touch the image border.
brightness	(1) A constant added to the red, green, and blue components of a color pixel during the color decoding process. (2) The perception by which white objects are distinguished from gray and light objects from dark objects.
buffer	Temporary storage for acquired data.

C

caliper	<p>(1) A function in the NI Vision Assistant and in NI Vision Builder for Automated Inspection that calculates distances, angles, circular fits, and the center of mass based on positions given by edge detection, particle analysis, centroid, and search functions.</p> <p>(2) A measurement function that finds edge pairs along a specified path in the image. This function performs an edge extraction and then finds edge pairs based on specified criteria such as the distance between the leading and trailing edges, edge contrasts, and so forth.</p>
center of mass	The point on an object where all the mass of the object could be concentrated without changing the first moment of the object about any axis.
chroma	The color information in a video signal.
chromaticity	The combination of hue and saturation. The relationship between chromaticity and brightness characterizes a color.
closing	A dilation followed by an erosion. A closing fills small holes in objects and smooths the boundaries of objects.
clustering	A technique where the image is sorted within a discrete number of classes corresponding to the number of phases perceived in an image. The gray values and a barycenter are determined for each class. This process is repeated until a value is obtained that represents the center of mass for each phase or class.
CLUT	Color lookup table. A table for converting the value of a pixel in an image into a red, green, and blue (RGB) intensity.
color image	An image containing color information, usually encoded in the RGB form.
color space	The mathematical representation for a color. For example, color can be described in terms of red, green, and blue; hue, saturation, and luminance; or hue, saturation, and intensity.
complex image	Stores information obtained from the FFT of an image. The complex numbers that compose the FFT plane are encoded in 64-bit floating-point values: 32 bits for the real part and 32 bits for the imaginary part.
connectivity	Defines which of the surrounding pixels of a given pixel constitute its neighborhood.

connectivity-4	Only pixels adjacent in the horizontal and vertical directions are considered neighbors.
connectivity-8	All adjacent pixels are considered neighbors.
contrast	A constant multiplication factor applied to the luma and chroma components of a color pixel in the color decoding process.
convex hull	The smallest convex polygon that can encapsulate a particle.
convex hull function	Computes the convex hull of objects in a binary image.
convolution	See linear filter .
convolution kernel	2D matrices, or templates, used to represent the filter in the filtering process. The contents of these kernels are a discrete two-dimensional representation of the impulse response of the filter that they represent.
curve extraction	The process of finding curves, or connected edge points, in a grayscale image. Curves usually represent the boundaries of objects in the image.

D

Danielsson function	Similar to the distance functions, but with more accurate results.
determinism	A characteristic of a system that describes how consistently it can respond to external events or perform operations within a given time limit.
digital image	An image $f(x, y)$ that has been converted into a discrete number of pixels. Both spatial coordinates and brightness are specified.
dilation	Increases the size of an object along its boundary and removes tiny holes in the object.
driver	Software that controls a specific hardware device, such as an NI Vision or DAQ device.

E

edge	Defined by a sharp transition in the pixel intensities in an image or along an array of pixels.
edge contrast	The difference between the average pixel intensity before and the average pixel intensity after the edge.
edge detection	Any of several techniques to identify the edges of objects in an image.
edge steepness	The number of pixels that corresponds to the slope or transition area of an edge.
energy center	The center of mass of a grayscale image. <i>See also</i> center of mass .
equalize function	<i>See</i> histogram equalization .
erosion	Reduces the size of an object along its boundary and eliminates isolated points in the image.
exponential and gamma corrections	Expand the high gray-level information in an image while suppressing low gray-level information.
exponential function	Decreases brightness and increases contrast in bright regions of an image, and decreases contrast in dark regions of an image.

F

FFT	Fast Fourier Transform. A method used to compute the Fourier transform of an image.
fiducial	A reference pattern on a part that helps a machine vision application find the part's location and orientation in an image.
Fourier transform	Transforms an image from the spatial domain to the frequency domain.
frequency filters	The counterparts of spatial filters in the frequency domain. For images, frequency information is in the form of spatial frequency.
ft	Feet.
function	A set of software instructions executed by a single line of code that may have input and/or output parameters and returns a value when executed.

G

gamma	The nonlinear change in the difference between the video signal's brightness level and the voltage level needed to produce that brightness.
geometric features	Information extracted from a grayscale template that are used to locate the template in the target image. Geometric features in an image range from low-level features such as edges or curves detected in the image to higher-level features such as the geometric shapes made by the curves in the image.
geometric matching	A technique used to locate quickly a grayscale template that is characterized by distinct geometric or shape information within a grayscale image.
Golden Template comparison	A comparison of 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.
gradient convolution filter	<i>See</i> gradient filter.
gradient filter	An edge detection algorithm that extracts the contours in gray-level values. Gradient filters include the Prewitt and Sobel filters.
gray level	The brightness of a pixel in an image.
gray-level dilation	Increases the brightness of pixels in an image that are surrounded by other pixels with a higher intensity.
gray-level erosion	Reduces the brightness of pixels in an image that are surrounded by other pixels with a lower intensity.
grayscale image	An image with monochrome information.
grayscale morphology	Functions that perform morphological operations on a gray-level image.

H

h	Hour.
highpass attenuation	The inverse of lowpass attenuation.
highpass filter	Emphasizes the intensity variations in an image, detects edges or object boundaries, and enhances fine details in an image.
highpass frequency filter	Removes or attenuates low frequencies present in the frequency domain of the image. A highpass frequency filter suppresses information related to slow variations of light intensities in the spatial image.
highpass truncation	The inverse of lowpass truncation.
histogram	Indicates the quantitative distribution of the pixels of an image per gray-level value.
histogram equalization	Transforms the gray-level values of the pixels of an image to occupy the entire range of the histogram, thus increasing the contrast of the image. The histogram range in an 8-bit image is 0 to 255.
histogram inversion	Finds the photometric negative of an image. The histogram of a reversed image is equal to the original histogram flipped horizontally around the center of the histogram.
histograph	In LabVIEW, a histogram that can be wired directly into a graph.
hit-miss function	Locates objects in the image similar to the pattern defined in the structuring element.
HSI	A color encoding scheme in hue, saturation, and intensity.
HSL	A color encoding scheme using hue, saturation, and luminance information where each image in the pixel is encoded using 32 bits: 8 bits for hue, 8 bits for saturation, 8 bits for luminance, and 8 unused bits.
HSV	A color encoding scheme in hue, saturation, and value.
hue	Represents the dominant color of a pixel. The hue function is a continuous function that covers all the possible colors generated using the R, G, and B primaries. <i>See also</i> RGB .
Hz	Hertz. Frequency in units of 1/second.

I

I/O	Input/output. The transfer of data to/from a computer system involving communications channels, operator interface devices, and/or data acquisition and control interfaces.
image	A two-dimensional light intensity function $f(x, y)$ where x and y denote spatial coordinates and the value f at any point (x, y) is proportional to the brightness at that point.
image border	A user-defined region of pixels surrounding an image. Functions that process pixels based on the value of the pixel neighbors require image borders.
Image Browser	An image that contains thumbnails of images to analyze or process in a vision application.
image buffer	A memory location used to store images.
image definition	The number of values a pixel can take on, which is the number of colors or shades that you can see in the image.
image display environment	A window or control that displays an image.
image enhancement	The process of improving the quality of an image that you acquire from a sensor in terms of signal-to-noise ratio, image contrast, edge definition, and so on.
image file	A file containing pixel data and additional information about the image.
image format	Defines how an image is stored in a file. Usually composed of a header followed by the pixel data.
image mask	A binary image that isolates parts of a source image for further processing. A pixel in the source image is processed if its corresponding mask pixel has a non-zero value. A source pixel whose corresponding mask pixel has a value of 0 is left unchanged.
image palette	The gradation of colors used to display an image on screen, usually defined by a CLUT.
image processing	Encompasses various processes and analysis functions that you can apply to an image.

image source	The original input image.
imaging	Any process of acquiring and displaying images and analyzing image data.
inner gradient	Finds the inner boundary of objects.
inspection	The process by which parts are tested for simple defects such as missing parts or cracks on part surfaces.
inspection function	Analyzes groups of pixels within an image and returns information about the size, shape, position, and pixel connectivity. Typical applications include quality of parts, analyzing defects, locating objects, and sorting objects.
instrument driver	A set of high-level software functions, such as NI-IMAQ, that control specific plug-in computer boards. Instrument drivers are available in several forms, ranging from a function callable from a programming language to a VI in LabVIEW.
intensity	The sum of the Red, Green, and Blue primary colors divided by three, $(Red + Green + Blue)/3$.
intensity calibration	Assigns user-defined quantities such as optical densities or concentrations to the gray-level values in an image.
intensity profile	The gray-level distribution of the pixels along an ROI in an image.
intensity range	Defines the range of gray-level values in an object of an image.
intensity threshold	Characterizes an object based on the range of gray-level values in the object. If the intensity range of the object falls within the user-specified range, it is considered an object. Otherwise it is considered part of the background.

J

jitter	The maximum amount of time that the execution of an algorithm varies from one execution to the next.
JPEG	Joint Photographic Experts Group. An image file format for storing 8-bit and color images with lossy compression. JPEG images have the file extension <i>JPG</i> .

JPEG2000 An Image file format for storing 8-bit, 16-bit, or color images with either lossy or lossless compression. JPEG2000 images have the file extension *JP2*.

K

kernel A structure that represents a pixel and its relationship to its neighbors. The relationship is specified by weighted coefficients of each neighbor.

L

labeling A morphology operation that identifies each object in a binary image and assigns a unique pixel value to all the pixels in an object. This process is useful for identifying the number of objects in the image and giving each object a unique pixel intensity.

LabVIEW Laboratory Virtual Instrument Engineering Workbench. A program development environment application based on the programming language G used commonly for test and measurement applications.

LabWindows/CVI Windows/Sun Product.

line gauge Measures the distance between selected edges with high-precision subpixel accuracy along a line in an image. For example, this function can be used to measure distances between points and edges. This function also can step and repeat its measurements across the image.

line profile Represents the gray-level distribution along a line of pixels in an image.

linear filter A special algorithm that calculates the value of a pixel based on its own pixel value as well as the pixel values of its neighbors. The sum of this calculation is divided by the sum of the elements in the matrix to obtain a new pixel value.

local threshold Creates a binary image by segmenting a grayscale image into a particle region and a background region.

logarithmic function Increases the brightness and contrast in dark regions of an image and decreases the contrast in bright regions of the image.

logic operators The image operations AND, NAND, OR, XOR, NOR, XNOR, difference, mask, mean, max, and min.

lossless compression	Compression in which the decompressed image is identical to the original image.
lossy compression	Compression in which the decompressed image is visually similar but not identical to the original image.
lowpass attenuation	Applies a linear attenuation to the frequencies in an image, with no attenuation at the lowest frequency and full attenuation at the highest frequency.
lowpass FFT filter	Removes or attenuates high frequencies present in the FFT domain of an image.
lowpass filter	Attenuates intensity variations in an image. You can use these filters to smooth an image by eliminating fine details and blurring edges.
lowpass frequency filter	Attenuates high frequencies present in the frequency domain of the image. A lowpass frequency filter suppresses information related to fast variations of light intensities in the spatial image.
lowpass truncation	Removes all frequency information above a certain frequency.
L-skeleton function	Uses an L-shaped structuring element in the skeleton function.
luma	The brightness information in the video picture. The luma signal amplitude varies in proportion to the brightness of the video signal and corresponds exactly to the monochrome picture.
luminance	<i>See</i> luma.
LUT	Lookup table. A table containing values used to transform the gray-level values of an image. For each gray-level value in the image, the corresponding new value is obtained from the lookup table.

M

M	<p>(1) Mega, the standard metric prefix for 1 million or 10^6, when used with units of measure such as volts and hertz.</p> <p>(2) Mega, the prefix for 1,048,576, or 220, when used with B to quantify data or computer memory.</p>
machine vision	An automated application that performs a set of visual inspection tasks.

mask FFT filter	Removes frequencies contained in a mask (range) specified by the user.
match score	A number ranging from 0 to 1000 that indicates how closely an acquired image matches the template image. A match score of 1000 indicates a perfect match. A match score of 0 indicates no match.
MB	Megabyte of memory.
median filter	A lowpass filter that assigns to each pixel the median value of its neighbors. This filter effectively removes isolated pixels without blurring the contours of objects.
memory buffer	See buffer .
MMX	Multimedia Extensions. An Intel chip-based technology that allows parallel operations on integers, which results in accelerated processing of 8-bit images.
morphological transformations	Extract and alter the structure of objects in an image. You can use these transformations for expanding (dilating) or reducing (eroding) objects, filling holes, closing inclusions, or smoothing borders. They are used primarily to delineate objects and prepare them for quantitative inspection analysis.
M-skeleton function	Uses an M-shaped structuring element in the skeleton function.
multiple template matching	The technique used to simultaneously locate multiple grayscale templates within a grayscale image.

N

neighbor	A pixel whose value affects the value of a nearby pixel when an image is processed. The neighbors of a pixel are usually defined by a kernel or a structuring element.
neighborhood operations	Operations on a point in an image that take into consideration the values of the pixels neighboring that point.
NI-IMAQ	The driver software for National Instruments image acquisition hardware.
NI-IMAQdx	The National Instruments driver software for IEEE 1394 or Gigabit Ethernet (GigE) cameras.

nonlinear filter	Replaces each pixel value with a nonlinear function of its surrounding pixels.
nonlinear gradient filter	A highpass edge-extraction filter that favors vertical edges.
nonlinear Prewitt filter	A highpass, edge-extraction filter based on two-dimensional gradient information.
nonlinear Sobel filter	A highpass, edge-extraction filter based on two-dimensional gradient information. The filter has a smoothing effect that reduces noise enhancements caused by gradient operators.
Nth order filter	Filters an image using a nonlinear filter. This filter orders (or classifies) the pixel values surrounding the pixel being processed. The pixel being processed is set to the Nth pixel value, where N is the order of the filter.
number of planes (in an image)	The number of arrays of pixels that compose the image. A gray-level or pseudo-color image is composed of one plane, while an RGB image is composed of three planes (one for the red component, one for the blue, and one for the green).

0

occlusion invariant matching	A geometric matching technique in which the reference pattern can be partially obscured in the target image.
OCR	Optical Character Recognition. The process of analyzing an image to detect and recognize characters/text in the image.
OCV	Optical Character Verification. A machine vision application that inspects the quality of printed characters.
offset	The coordinate position in an image where you want to place the origin of another image. Setting an offset is useful when performing mask operations.
opening	An erosion followed by a dilation. An opening removes small objects and smooths boundaries of objects in the image.
operators	Allow masking, combination, and comparison of images. You can use arithmetic and logic operators in NI Vision.

optical representation	Contains the low-frequency information at the center and the high-frequency information at the corners of an FFT-transformed image.
outer gradient	Finds the outer boundary of objects.

P

palette	The gradation of colors used to display an image on screen, usually defined by a CLUT.
particle	A connected region or grouping of non-zero pixels in a binary image.
particle analysis	A series of processing operations and analysis functions that produce some information about the particles in an image.
pattern matching	The technique used to locate quickly a grayscale template within a grayscale image
picture element	An element of a digital image. Also called pixel.
pixel	Picture element. The smallest division that makes up the video scan line. For display on a computer monitor, a pixel's optimum dimension is square (aspect ratio of 1:1, or the width equal to the height).
pixel aspect ratio	The ratio between the physical horizontal size and the vertical size of the region covered by the pixel. An acquired pixel should optimally be square, thus the optimal value is 1.0, but typically it falls between 0.95 and 1.05, depending on camera quality.
pixel calibration	Directly calibrates the physical dimensions of a pixel in an image.
pixel depth	The number of bits used to represent the gray level of a pixel.
PNG	Portable Network Graphic. An image file format for storing 8-bit, 16-bit, and color images with lossless compression. PNG images have the file extension <i>PNG</i> .
Prewitt filter	An edge detection algorithm that extracts the contours in gray-level values using a 3×3 filter kernel.
proper-closing	A finite combination of successive closing and opening operations that you can use to fill small holes and smooth the boundaries of objects.

proper-opening A finite combination of successive opening and closing operations that you can use to remove small particles and smooth the boundaries of objects.

Q

quantitative analysis Obtaining various measurements of objects in an image.

R

real time A property of an event or system in which data is processed as it is acquired instead of being accumulated and processed at a later time.

resolution The number of rows and columns of pixels. An image composed of m rows and n columns has a resolution of $m \times n$.

reverse function Inverts the pixel values in an image, producing a photometric negative of the image.

RGB A color encoding scheme using red, green, and blue (RGB) color information where each pixel in the color image is encoded using 32 bits: 8 bits for red, 8 bits for green, 8 bits for blue, and 8 bits for the alpha value (unused).

RGB U64 A color encoding scheme using red, green, and blue (RGB) color information where each pixel in the color image is encoded using 64 bits: 16 bits for red, 16 bits for green, 16 bits for blue, and 16 bits for the alpha value (unused).

Roberts filter An edge detection algorithm that extracts the contours in gray level, favoring diagonal edges.

ROI Region of interest.

- (1) An area of the image that is graphically selected from a window displaying the image. This area can be used focus further processing.
- (2) A hardware-programmable rectangular portion of the acquisition window.

ROI tools A collection of tools that enable you to select a region of interest from an image. These tools let you select points, lines, annuli, polygons, rectangles, rotated rectangles, ovals, and freehand open and closed contours.

rotational shift	The amount by which one image is rotated relative to a reference image. This rotation is computed relative to the center of the image.
rotation-invariant matching	A pattern matching technique in which the reference pattern can be located at any orientation in the test image as well as rotated at any degree.

S

saturation	The amount of white added to a pure color. Saturation relates to the richness of a color. A saturation of zero corresponds to a pure color with no white added. Pink is a red with low saturation.
scale-invariant matching	A pattern matching technique in which the reference pattern can be any size in the test image.
segmentation function	Fully partitions a labeled binary image into non-overlapping segments, with each segment containing a unique object.
separation function	Separates objects that touch each other by narrow isthmuses.
shape detection	Detects rectangles, lines, ellipses, and circles within images.
shift-invariant matching	A pattern matching technique in which the reference pattern can be located anywhere in the test image but cannot be rotated or scaled.
skeleton function	Applies a succession of thinning operations to an object until its width becomes one pixel.
smoothing filter	Blurs an image by attenuating variations of light intensity in the neighborhood of a pixel.
Sobel filter	An edge detection algorithm that extracts the contours in gray-level values using a 3×3 filter kernel.
spatial calibration	Assigns physical dimensions to the area of a pixel in an image.
spatial filters	Alter the intensity of a pixel relative to variations in intensities of its neighboring pixels. You can use these filters for edge detection, image enhancement, noise reduction, smoothing, and so forth.
spatial resolution	The number of pixels in an image, in terms of the number of rows and columns in the image.
square function	See exponential function .

square root function	See logarithmic function .
standard representation	Contains the low-frequency information at the corners and high-frequency information at the center of an FFT-transformed image.
structuring element	A binary mask used in most morphological operations. A structuring element is used to determine which neighboring pixels contribute in the operation.
subpixel analysis	Finds the location of the edge coordinates in terms of fractions of a pixel.

T

template	A color, shape, or pattern that you are trying to match in an image using the color matching, shape matching, or pattern matching functions. A template can be a region selected from an image or it can be an entire image.
threshold	Separates objects from the background by assigning all pixels with intensities within a specified range to the object and the rest of the pixels to the background. In the resulting binary image, objects are represented with a pixel intensity of 255 and the background is set to 0.
threshold interval	Two parameters, the lower threshold gray-level value and the upper threshold gray-level value.
TIFF	Tagged Image File Format. An image format commonly used for encoding 8-bit, 16-bit, and color images. TIFF images have the file extension <i>TIF</i> .
time-bounded	Describes algorithms that are designed to support a lower and upper bound on execution time.
tools palette	A collection of tools that enable you to select regions of interest, zoom in and out, and change the image palette.

V

value	The grayscale intensity of a color pixel computed as the average of the maximum and minimum red, green, and blue values of that pixel.
-------	--

VI

Virtual Instrument.

(1) A combination of hardware and/or software elements, typically used with a PC, that has the functionality of a classic stand-alone instrument.

(2) A LabVIEW software module (VI), which consists of a front panel user interface and a block diagram program.

W

Watershed Transform

Partitions an image based on the topographic surface of the image. The image is separated into non-overlapping segments with each segment containing a unique particle.

Index

Numerics

- 1D barcodes
 - reading, 5-33
- 2D arrays
 - converting to images, 2-5, 2-7

A

- acquiring
 - images, 2-5
- analyzing
 - components of color images, 3-8
 - images, 2-8
 - particles, 4-1
- applications
 - creating Vision, 1-4
- applying
 - LUTs, 2-9
- applying to images, 2-9
- arrays
 - converting planes of complex images to arrays, 2-13
- attaching
 - calibration information, 6-10
 - calibration information to images, 2-8
- attenuation
 - highpass, 2-12
 - lowpass, 2-12
- automatic thresholding, 4-1

B

- barcodes
 - reading 1D barcodes, 5-33
 - reading data matrix barcodes, 5-34
 - reading PDF417 barcodes, 5-34

- binary images
 - creating, 4-1
 - improving, 4-2
- binary morphology, 4-4
- BMP, 2-6

C

- calibrating
 - images, 6-1
 - imaging systems, 2-2
- calibration
 - defining templates, 6-2
 - saving calibration information, 6-10
 - using simple calibration, 6-9
- calibration information
 - attaching, 6-10
 - attaching to images, 2-8
 - learning, 6-5
 - saving, 6-10
- calibration templates
 - defining, 6-2
- calibrations
 - invalidating, 6-8
- characters
 - reading, 5-33
- choosing
 - learning algorithms, 6-6
- classifying
 - colors, 5-31
 - objects, 5-31
- closing
 - particles, 4-4
 - tools palette, 3-6

- color
 - classification, 5-31
 - defining color template images, 5-23
 - pattern matching
 - minimum contrast, 5-27
 - setting sensitivity, 5-26
 - using pattern matching to find points, 5-22
- color distribution
 - learning in images, 3-11, 3-12
 - learning with a single ROI, 3-12
 - learning with multiple ROIs, 3-12
- color images
 - analyzing components, 3-8
 - extracting planes, 3-9
 - ignoring learned colors, 3-14
 - processing, 3-8
- color information
 - learning, 3-10, 3-11
 - specifying, 3-11
- color location
 - using to find points, 5-28
- color representation sensitivity
 - specifying, 3-13
- color score weight, 5-27
- comparing
 - color content in images, 3-10
- computing
 - energy center of an image, 3-8
 - energy center of an ROI in an image, 3-8
- configuring
 - tools palette, 3-6
- conventions used in the manual, ix
- converting
 - 2D arrays to images, 2-5, 2-7
 - images to frequency domains, 2-11
 - pixel coordinates to real-world coordinates, 5-29
 - planes of complex images to arrays, 2-13
- convolution filter, 2-10

- coordinate systems
 - defining using edge detection, 5-3
 - defining using pattern matching, 5-5
- correction tables
 - learning, 6-8
- counting
 - particles, 4-5
- creating
 - binary images, 4-1
 - images, 2-2
 - Vision applications, 1-4

D

- data matrix barcodes
 - reading, 5-34
- Declaration of Conformity (NI resources), A-1
- defects
 - inspect, 5-35
- defining
 - calibration templates, 6-2
 - color template images, 5-23
 - coordinate systems using edge detection, 5-3
 - coordinate systems using pattern matching, 5-5
 - image masks, 3-7
 - pattern matching tolerances, 5-15
 - reference coordinate systems, 6-3
 - regions of interest, 3-1
 - ROIs, 6-6
 - ROIs interactively, 3-1, 5-7
 - ROIs programmatically, 3-7, 5-8
 - search areas, 5-7, 5-14, 5-25
 - template images, 5-11
- destination images, 2-4
- detecting
 - circular edges, 5-8
 - edges, 5-8
 - edges along a contour, 5-10
 - edges along multiple contours, 5-10
 - rectangular edges, 5-8

- determining, 2-8
 - image quality, 2-8
- diagnostic tools (NI resources), A-1
- displaying
 - images, 2-7
 - results, 5-36
 - tools palette, 3-6
- documentation
 - conventions used in manual, *ix*
 - NI resources, A-1
 - related documentation, *x*
- drivers (NI resources), A-1

E

- edge detection, 5-8
- edges
 - detecting, 5-8
 - detecting along a contour, 5-10
 - detecting along multiple contours, 5-10
 - detecting circular edges, 5-8
- eroding, particles, 4-4
- erosion, 4-4
- error maps
 - learning, 6-8
- examples (NI resources), A-1
- external windows
 - displaying images, 2-7
- extracting
 - planes of color images, 3-9

F

- Fast Fourier Transform, 2-11
- features, finding in images, 5-11
- FFT, 2-11
- filtering
 - grayscale features of an image, 2-11
 - images, 2-10

- filters
 - convolution, 2-10
 - highpass, 2-10
 - lowpass, 2-10
 - Nth order, 2-10
- finding
 - image features, 5-11
 - points using color location, 5-28
 - points using color pattern matching, 5-22
- frequency domains
 - converting images, 2-11
- function tree
 - NI Vision, 1-2
 - NI Vision Machine Vision, 1-4
- function types (table)
 - IMAQ Machine Vision, 1-4
 - IMAQ Vision, 1-2

G

- geometric matching, 5-17
 - match mode, 5-20
 - occlusion ranges, 5-21
 - rotation angle ranges, 5-20
 - scale factor ranges, 5-20
 - training, 5-19
- getting
 - center of energy for an image, 3-8
 - image statistics, 3-8
- golden template comparison, 5-35
- grayscale morphology, 2-11
- grayscale statistics
 - measuring, 3-8

H

- help, technical support, A-1
- highpass
 - attenuation, 2-12
 - filters, 2-10
 - truncation, 2-12
- holes, filling in particles, 4-4

I

- identifying
 - parts, 5-31
- ignoring
 - learned colors, 3-14
- image masks
 - defining, 3-7
- image quality, 2-8
- images
 - acquiring, 2-5
 - analyzing, 2-8
 - analyzing components, 3-8
 - applying LUTs, 2-9
 - attaching calibration information, 2-8
 - calibrating, 6-1
 - comparing color content, 3-10
 - computing the energy center, 3-8
 - computing the energy center of an ROI in an image, 3-8
 - converting 2D arrays to images, 2-5, 2-7
 - creating, 2-2
 - creating binary images, 4-1
 - defining template images, 5-11
 - destination, 2-4
 - detecting edges, 5-8
 - displaying, 2-7
 - extracting planes, 3-9
 - filtering, 2-10
 - filtering grayscale features, 2-11
 - finding features, 5-11
 - getting statistics, 3-8
 - getting the center of energy, 3-8
 - ignoring learned colors, 3-14
 - improving, 2-9
 - improving binary images, 4-2
 - improving sharpness of transitions, 2-10
 - inspecting, 2-8
 - learning color information, 3-10
 - learning the color distribution, 3-11, 3-12
 - loading from file, 2-5
 - measuring light intensity, 3-8
 - modifying complex images, 2-13
 - processing components, 3-8
 - reading, 2-5
 - reading from file, 2-6
 - setting color sensitivity, 5-26
 - source, 2-4
 - taking color measurements, 3-1
 - taking grayscale measurements, 3-1
 - thresholding, 4-1
- imaging systems
 - calibrating, 2-2
 - setting up, 2-1
- improving
 - binary images, 4-2
 - images, 2-9
 - sharpness of transitions, 2-10
- inspecting
 - images, 2-8
 - objects, 5-2
- inspection
 - defects, 5-35
- inspection tasks
 - performing, 5-1
- instrument drivers (NI resources), A-1
- interpreting
 - pattern matching results, 5-16
- invalidating
 - calibrations, 6-8

J

- JPEG, 2-6
- JPEG2000, 2-6

K

- KnowledgeBase, A-1

L

learning

- calibration information, 6-5
- color information, 3-10, 3-11
- color spectrum, 3-11, 3-12
- correction tables, 6-8
- error maps, 6-8

learning algorithms

- choosing, 6-6

learning scores

- using, 6-7

light intensity

- measuring in images, 3-8

local thresholding, 4-1

locating, 5-2

- measurement points, 5-8

lowpass

- attenuation, 2-12
- filters, 2-10
- truncation, 2-12

LUTs, 2-9

M

machine vision tasks, 5-1

measurement points

- locating, 5-8

measurements

- analytic geometry, 5-30
- distance measurements, 5-29
- instrument reader, 5-30

measuring

- grayscale statistics, 3-8
- light intensity in images, 3-8
- particles, 4-5

modifying

- complex images, 2-13

moving

- tools palette, 3-6

multiple ROIs

- learning color distribution, 3-12

multiple template images, 5-21

multiple thresholding, 4-1

N

National Instruments support and services, A-1

NI Machine Vision

- function tree, 1-4
- function types (table), 1-4

NI support and services, A-1

NI Vision

- function tree, 1-2
- function types (table), 1-2

NI Vision Assistant, x

NI Vision Template Editor, 5-17, 5-19

Nth order filter, 2-10

O

objects

- inspecting, 5-2
- locating, 5-2

open operation, 4-4

opening

- particles, 4-4

P

particle analysis

- performing, 4-1

particles

- analyzing, 4-1
- classifying, 5-31
- closing, 4-4
- counting, 4-5
- eroding, 4-4
- filling holes, 4-4
- measuring, 4-5
- opening, 4-4
- removing unwanted particles, 4-3

- separating touching particles, 4-4
- smoothing boundaries, 4-4
- parts
 - identifying, 5-31
- pattern matching
 - color score weight, 5-27
 - defining tolerances, 5-15
 - interpreting results, 5-16
 - minimum contrast, 5-16, 5-27
 - mode, 5-15
 - orientation, 5-15
 - selecting search strategies, 5-26
 - training color pattern matching
 - algorithm, 5-24
 - training the pattern matching
 - algorithm, 5-13
 - using color pattern matching, 5-22
- PDF417 barcodes
 - reading, 5-34
- performing
 - machine vision inspection tasks, 5-1
 - particle analysis, 4-1
- pixel coordinates
 - converting to real-world
 - coordinates, 5-29
- PNG, 2-6
- points
 - finding using color location, 5-28
- processing
 - components of color images, 3-8
- programming examples (NI resources), A-1

R

- read image file
 - BMP, 2-6
 - JPEG, 2-6
 - JPEG2000, 2-6
 - PNG, 2-6
 - TIFF, 2-6

- reading
 - 1D barcodes, 5-33
 - characters, 5-33
 - data matrix barcodes, 5-34
 - images, 2-5
 - images from file, 2-6
 - PDF417 barcodes, 5-34
- reference coordinate systems
 - defining, 6-3
- regions of interest (ROIs). *See* ROIs
- related documentation, *x*
- removing
 - unwanted particles, 4-3
- results
 - displaying, 5-36
- ROIs
 - defining, 3-1, 6-6
 - defining interactively, 3-1, 5-7
 - defining programmatically, 3-7, 5-8
 - rotating, 5-3
 - shifting, 5-3
- rotating
 - ROIs, 5-3

S

- saving
 - calibration information, 6-10
- scaling factors
 - specifying, 6-6
- scaling methods
 - setting, 6-8
- search algorithm
 - testing, 5-16, 5-21, 5-28
- search areas, 5-8
 - defining, 5-14, 5-25
 - ROIs
 - defining search areas, 5-7
- search strategies
 - selecting for pattern matching, 5-26
- selecting
 - pattern matching search strategies, 5-26

- separating
 - touching particles, 4-4
- setting
 - color sensitivity, 5-26
 - pattern matching tolerances, 5-26
 - scaling methods, 6-8
- setting up
 - imaging systems, 2-1
- shifting
 - ROIs, 5-3
- single ROI
 - learning color distribution, 3-12
- smoothing
 - boundaries of particles, 4-4
- software (NI resources), A-1
- source images, 2-4
- specifying
 - color information to learn, 3-11
 - color representation sensitivity, 3-13
 - scaling factors, 6-6
- support
 - technical, A-1

T

- taking
 - color measurements, 3-1
 - grayscale measurements, 3-1
- technical support, A-1
- template images
 - background information, 5-13, 5-24
 - color information, 5-23
 - creating, 5-18
 - defining color template images, 5-23
 - feature detail, 5-12, 5-23
 - multiple, 5-21
 - positional information, 5-13
 - symmetry, 5-12, 5-23
- templates
 - defining template images, 5-11

- testing
 - search algorithm, 5-16, 5-28
- thresholding, 4-1
 - images, 4-1
- TIFF, 2-6
- tolerances
 - defining for pattern matching, 5-15
 - setting for pattern matching, 5-26
- tools palette
 - closing, 3-6
 - configuring, 3-6
 - displaying, 3-6
 - moving, 3-6
- Tools palette functions, 3-2
- training
 - color pattern matching algorithm, 5-24
 - pattern matching algorithm, 5-13
- training and certification (NI resources), A-1
- troubleshooting (NI resources), A-1
- truncation
 - highpass, 2-12
 - lowpass, 2-12

U

- using
 - learning scores, 6-7
 - simple calibration, 6-9

V

- Vision applications
 - creating, 1-4

W

- watershed transform, 4-4
- Web resources, A-1
- windows
 - displaying images in external windows, 2-7