

- [Table of Contents](#)
- [Examples](#)

Image Processing with LabVIEW™ and IMAQ™ Vision

By [Thomas Klinger](#)

[Start Reading ►](#)

Publisher: Prentice Hall PTR

Pub Date: June 11, 2003

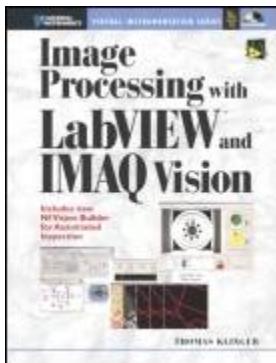
ISBN: 0-13-047415-0

Pages: 368

This book brings together everything you need to achieve superior results with PC-based image processing and analysis. Expert Thomas Klinger combines a highly accessible overview of the field's key concepts, tools, and techniques; the first expert introduction to NI's breakthrough IMAQ Vision software; and several start-to-finish application case studies. You also get an extensive library of code and image samples, as well as a complete trial version of IMAQ Vision for Windows. Coverage includes:

- Defining what to measure and how to measure it
- Acquiring images: working with CCDs, cameras, frame grabber cards, and leading medical image sources, including ultrasound, CT, and MRI
- Distributing images: compression techniques, image format standards, and DICOM medical imaging
- Processing images: gray-scale operations, spatial image filtering, frequency filtering, and morphology functions
- Analyzing images: pixel value and quantitative analyses, shape and pattern matching, bar codes, and more

With 300+ figures and 50+ exercises-all listed up front for easy access-this is the definitive image processing tutorial for every professional.



- [Table of Contents](#)
- [Examples](#)

Image Processing with LabVIEW™ and IMAQ™ Vision

By [Thomas Klinger](#)

[Start Reading ▶](#)

Publisher: Prentice Hall PTR

Pub Date: June 11, 2003

ISBN: 0-13-047415-0

Pages: 368

[Copyright](#)

[Virtual Instrumentation Series](#)

[About Prentice Hall Professional Technical Reference](#)

[List of Figures](#)

[List of Tables](#)

[List of Exercises](#)

[Preface](#)

[Acknowledgements](#)

[Disclaimer](#)

[Warning Regarding Medical and Clinical Use of National Instruments Products](#)

[Chapter 1. Introduction and Definitions](#)

[Introduction](#)

[Some Definitions](#)

[Introduction to IMAQ Vision Builder](#)

[NI Vision Builder for Automated Inspection](#)

[Chapter 2. Image Acquisition](#)

[Charge-Coupled Devices](#)

[Line-Scan Cameras](#)

[CMOS Image Sensors](#)

[Video Standards](#)

[Color Images](#)

[Other Image Sources](#)

[Chapter 3. Image Distribution](#)

[Frame Grabbing](#)

[Camera Interfaces and Protocols](#)

[Compression Techniques](#)

[Image Standards](#)

[Digital Imaging and Communication in Medicine \(DICOM\)](#)

[Chapter 4. Image Processing](#)

[Gray-Scale Operations](#)

[Spatial Image Filtering](#)

[Frequency Filtering](#)

[Morphology Functions](#)

[Chapter 5. Image Analysis](#)

[Pixel Value Analysis](#)

[Quantitative Analysis](#)

[Shape Matching](#)

[Pattern Matching](#)

[Reading Instrument Displays](#)

[Character Recognition](#)

[Image Focus Quality](#)

[Application Examples](#)

[Bibliography](#)

[Chapter 1:](#)

[Chapter 2:](#)

[Chapter 3:](#)

[Chapters 4 and 5:](#)

[Application Papers:](#)

[About the Author](#)

[About the CD-ROM](#)

[License Agreement](#)

[Technical Support](#)

[\[Team LiB \]](#)

PREVIOUS NEXT

Copyright

Library of Congress Cataloging-in-Publication Data

Klinger, Thomas, Ph.D.

Image processing with LabVIEW and IMAQ vision / Thomas Klinger.

p. cm.—(National Instruments virtual instrumentation series)

Includes bibliographical references and index.

ISBN 0-13-047415-0

1. Image processing—Digital techniques. 2. Engineering instruments—Data processing. 3.

LabVIEW. I. Title. II. Series.

TA1632.K58 2003

621.36'7—dc21

2003045952

Editorial/production supervision: *Jane Bonnell*

Cover design director: *Jerry Votta*

Cover design: *Nina Scuderi*

Manufacturing buyer: *Maura Zaldivar*

Publisher: *Bernard M. Goodwin*

Editorial assistant: *Michelle Vincenti*

Marketing manager: *Dan DePasquale*

© 2003 Pearson Education, Inc.

Publishing as Prentice Hall Professional Technical Reference

Upper Saddle River, New Jersey 07458

Prentice Hall PTR offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact: U.S. Corporate and Government Sales, 1-800-382-3419, corpsales@pearsontechgroup.com. For sales outside of the U.S., please contact: International Sales, 1-317-581-3793, international@pearsontechgroup.com.

Company and product names mentioned herein are the trademarks or registered trademarks of their respective owners.

All rights reserved. No part of this book may be reproduced, in any form or by any means, without permission in writing from the publisher.

Printed in the United States of America

First Printing

Pearson Education LTD.

Pearson Education Australia PTY, Limited

Pearson Education Singapore, Pte. Ltd.

Pearson Education North Asia Ltd.

Pearson Education Canada, Ltd.
Pearson Educaciòn de Mexico, S.A. de C.V.
Pearson Education—Japan
Pearson Education Malaysia, Pte. Ltd.

To Uschi, Peter, and Judith

[[Team LiB](#)]

[ PREVIOUS] [[NEXT](#) ]

Virtual Instrumentation Series

Kenneth L. Ashley

- Analog Electronics with LabVIEW

Jeffrey Y. Beyon

- Hands-On Exercise Manual for LabVIEW Programming, Data Acquisition, and Analysis

Jeffrey Y. Beyon

- LabVIEW Programming, Data Acquisition, and Analysis

Mahesh L. Chugani • Abhay R. Samant • Michael Cerra

- LabVIEW Signal Processing

Jon Conway • Steve Watts

- A Software Engineering Approach to LabVIEW

Nesimi Ertugrul

- LabVIEW for Electric Circuits, Machines, Drives, and Laboratories

Rahman Jamal • Herbert Pichlik

- LabVIEW Applications and Solutions

Shahid F. Khalid

- Advanced Topics in LabWindows/CVI

Shahid F. Khalid

- LabWindows/CVI Programming for Beginners

Thomas Klinger

- Image Processing with LabVIEW and IMAQ Vision

Hall T. Martin • Meg L. Martin

- LabVIEW for Automotive, Telecommunications, Semiconductor, Biomedical, and Other Applications

Bruce Mihura

- LabVIEW for Data Acquisition

Jon B. Olansen • Eric Rosow

- Virtual Bio-Instrumentation: Biomedical, Clinical, and Healthcare Applications in LabVIEW

Barry Paton

- Sensors, Transducers, and LabVIEW

Jeffrey Travis

- LabVIEW for Everyone, second edition

About Prentice Hall Professional Technical Reference

With origins reaching back to the industry's first computer science publishing program in the 1960s, and formally launched as its own imprint in 1986, Prentice Hall Professional Technical Reference (PH PTR) has developed into the leading provider of technical books in the world today. Our editors now publish over 200 books annually, authored by leaders in the fields of computing, engineering, and business.

Our roots are firmly planted in the soil that gave rise to the technical revolution. Our bookshelf contains many of the industry's computing and engineering classics: Kernighan and Ritchie's *C Programming Language*, Nemeth's *UNIX System Administration Handbook*, Horstmann's *Core Java*, and Johnson's *High-Speed Digital Design*.



PH PTR acknowledges its auspicious beginnings while it looks to the future for inspiration. We continue to evolve and break new ground in publishing by providing today's professionals with tomorrow's solutions.

List of Figures

1.1	Ultrasound Imager and Refractometer	7
1.2	Scientific Microscope and Visual Presenter	7
1.3	Network Structure with Simultaneous Use of Ethernet and IEEE1394	8
1.4	Definition of an Image as a Rectangular Matrix	10
1.5	Definition of a Color Image as Multiplane Image	11
1.6	Definition of an RGB-Color Image	12
1.7	Image Processing Example	13
1.8	Image Analysis Example	13
1.9	IMAQ Vision Builder Environment	16
1.10	IMAQ Vision Builder Image Browser	17
1.11	Acquiring Images into IMAQ Vision Builder	18
1.12	Edge Detection with a Line Profile	20
1.13	Using the Caliper Tool	21
1.14	Creating a Script File for Blob Analysis	23
1.15	LabVIEW VI Creation Wizard	26
1.16	<code>metal.vi</code> Created by the VI Creation Wizard	27
1.17	NI Vision Builder AI Configuration Interface	29
1.18	NI Vision Builder AI Inspection Interface	30
1.19	Finding a Straight Edge in NI Vision Builder AI	30
1.20	Measuring the Distance Between Two Edges	31
1.21	Pattern Matching in a Limited Search Region	31
2.1	Questions Regarding Pixel Transfer	34
2.2	Principle of a CCD Sensor	34
2.3	CCD Transfer Mechanism	35
2.4	Charge Transfer Efficiency (CTE) as a Function of Pulse Length	36
2.5	Impact of Charge Transfer Efficiency (CTE) on Pixel Brightness	37
2.6	Charge Transfer Efficiency (CTE) Far Too Low	38
2.7	Structure of Surface Channel and Buried Channel CCDs	39
2.8	Visualization of the Modulation Transfer Function (MTF)	41
2.9	Hot Pixels of a 2.1 Megapixel CCD Camera	42
2.10	MediaChance Hot Pixel Eliminator	43
2.11	Blooming Effect Caused by a Laser Pointer	44

2.12	Blooming Effect (Exercise)	45
2.13	Test Image for Measuring the Smear Effect	46
2.14	Structure of a Linear CCD Sensor	46
2.15	CCD Sensor Chip with Interline Structure	47
2.16	Comparison of Interline and Frame Transfer Structures	48
2.17	CCD Sensor Chip with Frame Transfer Structure	48
2.18	Principle of a Line-Scan Camera	49
2.19	Line-Scan Sensor Used in a Flat-Bed Scanner	50
2.20	Principle of a CMOS Image Sensor	51
2.21	CMOS (Color) Sensor Chip	52
2.22	Blooming Effect in CCD and CMOS Cameras	53
2.23	Smear Effect in CCD and CMOS Cameras	54
2.24	Video Frames (European CCIR Standard)	55
2.25	CCIR Standard Timing Diagram	56
2.26	RS 170 Standard Timing Diagram	56
2.27	Interlaced Mode of a CCD Video Sensor	57
2.28	Noninterlaced Mode and Progressive Scan Sensor	58
2.29	Image Reduction with IMAQ's Interlace Function	58
2.30	Possibilities for Color CCD Sensors	59
2.31	RGB Color Cube	60
2.32	HSI Color Triangle and Color Solid	62
2.33	Front Panel of the VI Created in Exercise 2.4	63
2.34	Diagram of the VI Created in Exercise 2.4	64
2.35	Color Subsampling in Digital Video	67
2.36	Principle of Ultrasound A and M Mode	68
2.37	Curved Array Ultrasound Head and Corresponding Image	68
2.38	CT Device Generations 1 to 4	71
2.39	Simple Calculation of a CT Image	72
2.40	Tomography Simulation	73
2.41	Iterative Calculation of a CT Image	74
2.42	Separation of Energy Levels According to Spin Directions	75
2.43	Relaxation Times τ_1 and τ_2	76
2.44	MRI Images: Based on τ_1 and τ_2 of a Knee Joint	77
3.1	Getting an Image into a PC	79
3.2	Typical Block Diagram of a PCI Frame Grabber	81
3.3	Typical 1394 Bus Structure with Single- and Multiport Devices	83
3.4	Windows Device Manager Listing 1394 Devices	84

3.5	1394 Zip100 Drive and 1394 Hard Drive	85
3.6	1394 Video Camera	85
3.7	1394 Repeater	86
3.8	1394 PC104 Boards	86
3.9	Isochronous and Asynchronous Transactions	87
3.10	1394 6-Pin Connector (Plug and Socket)	89
3.11	1394 4-Pin Connector (Plug and Socket)	89
3.12	Cross Sections of 4-Conductor and 6-Conductor Cables	90
3.13	Data Strobe Encoding	91
3.14	Example of a 1394 Bus Topology	92
3.15	Isochronous Stream Packet (Video Data YUV 4:4:4)	94
3.16	1394 Camera Image and Properties in IMAQ Vision Builder	95
3.17	Setting 1394 Camera Properties in LabVIEW	96
3.18	Windows Device Manager Listing USB Devices	97
3.19	USB Hub Types	98
3.20	USB Hub Performing Downstream and Upstream Connectivity	98
3.21	USB Hub with Four Ports	99
3.22	USB Mass Storage Device and USB Camera	101
3.23	Cross Sections of Low-Speed and High-Speed USB Cables	101
3.24	USB Cable with A- and B-Plug	102
3.25	USB Cables Using NRZI Encoding and Differential Signaling	102
3.26	NRZI Encoding	103
3.27	Importing USB Camera Images in LabVIEW	106
3.28	Camera Link Block Diagram (Base Configuration)	107
3.29	Camera Link Block Diagram (Medium and Full Configuration)	108
3.30	Compression Techniques and Algorithms	111
3.31	Example of Huffman Coding	112
3.32	Lempel-Ziv Coding Example (LZ77 Algorithm)	113
3.33	Arithmetic Coding Example	115
3.34	Arithmetic Decoding Example	115
3.35	8 x 8 DCT Coefficients	117
3.36	Calculating 8 x 8 DCT Coefficients with LabVIEW	118
3.37	Diagram of Exercise 3.3	118
3.38	DCT and Inverse DCT Calculation	119
3.39	DCT and Inverse DCT Calculation with JPEG Quantization	120
3.40	JPEG Quantization Table and Coefficient Reading	121
3.41	2D Wavelet Transform Example	122

3.42	JPEG2000 Generation Tool	123
3.43	Comparison of JPEG and JPEG2000 Image Areas	123
3.44	Simple DICOM Communication	141
3.45	Structure of a DICOM Data Element	143
3.46	ActiveX Control Import List	147
3.47	Imported ActiveX Control	147
3.48	Import of Accusoft DICOM Comm SDK in LabVIEW	148
3.49	Loading DICOM Images into LabVIEW and IMAQ Vision	148
3.50	Frames 0 and 1 of Exercise 3.5	149
4.1	Histogram Function in IMAQ Vision Builder	152
4.2	Histogram and Histogram of an Image	153
4.3	Histogram Exported in MS Excel	154
4.4	Color Histogram Function in IMAQ Vision Builder	154
4.5	Exercise 4.2 : Creating User LuTs	155
4.6	Processing Look-up Tables (LuTs)	156
4.7	Creating a Logarithmic Look-up Table	157
4.8	Creating an Exponential Look-up Table	158
4.9	Creating a Square Look-up Table	158
4.10	Creating a Square Root Look-up Table	159
4.11	Creating a Power x Look-up Table	159
4.12	Creating a Power $1/x$ Look-up Table	160
4.13	Image and Histogram Resulting from Equalizing	161
4.14	Diagram of Exercise 4.3	162
4.15	Inverting the Bear Image	163
4.16	Diagram of Exercise 4.4	164
4.17	Using Special LuTs for Modifying Brightness and Contrast	165
4.18	Diagram of Exercise 4.5	166
4.19	Moving the Filter Kernel	167
4.20	Diagram of Exercise 4.6	168
4.21	Visualizing Effects of Various Filter Kernels	168
4.22	Diagram of Exercise 4.7	169
4.23	Filter Example: Smoothing (#5)	170
4.24	Filter Example: Gaussian (#4)	171
4.25	Filter Example: Gradient (#0)	172
4.26	Filter Example: Gradient (#1)	173
4.27	Filter Example: Gradient (#4)	173
4.28	Filter Example: Laplace (#0)	175

4.29	Filter Example: Laplace (#1)	175
4.30	Filter Example: Laplace (#6)	176
4.31	Filter Example: Laplace (#7)	176
4.32	Waveform Spectrum	177
4.33	FFT Spectrum of an Image	178
4.34	FFT Spectrum of an Image	179
4.35	FFT Spectrum (Standard and Optical Display)	179
4.36	FFT Low-Pass Filter	180
4.37	Diagram of Exercise 4.9	181
4.38	FFT High-Pass Filter	181
4.39	FFT Low-Pass Attenuation Filter	183
4.40	FFT High-Pass Attenuation Result	183
4.41	Morphology Functions in IMAQ Vision Builder	184
4.42	Thresholding with IMAQ Vision Builder	184
4.43	Result of Thresholding Operation	185
4.44	Thresholding with IMAQ Vision	186
4.45	Diagram of Exercise 4.11	186
4.46	Predefined Thresholding Functions	187
4.47	Diagram of Exercise 4.12	188
4.48	Examples of Structuring Elements	189
4.49	Configuring the Structuring Element in IMAQ Vision Builder	190
4.50	Morphology Functions: Erosion	191
4.51	Diagram of Exercise 4.13	192
4.52	Morphology: Dilation Result	193
4.53	Morphology: Opening Result	194
4.54	Morphology: Closing Result	194
4.55	Morphology: Proper Opening Result	195
4.56	Morphology: Proper Closing Result	196
4.57	Hit-Miss Result with Structuring Element That Is All 1s	197
4.58	Hit-Miss Result with Structuring Element That Is All 0s	198
4.59	Inner Gradient (Internal Edge) Result	198
4.60	Outer Gradient (External Edge) Result	199
4.61	Morphology: Gradient Result	200
4.62	Morphology: Thinning Result	200
4.63	Morphology: Thickening Result	201
4.64	Morphology: Auto-Median Result	202
4.65	Remove Particle: Low Pass	203

4.66	Diagram of Exercise 4.14	204
4.67	Remove Particle: High Pass	204
4.68	Particles Touching the Border Are Removed	205
4.69	Diagram of Exercise 4.15	205
4.70	Particle Filtering by x Coordinate	206
4.71	Diagram of Exercise 4.16	207
4.72	Filling Holes in Particles	210
4.73	Diagram of Exercise 4.17	210
4.74	IMAQ Convex Function	211
4.75	Diagram of Exercise 4.18	211
4.76	Separation of Particles	212
4.77	Diagram of Exercise 4.19	213
4.78	IMAQ Magic Wand: Separating Objects from the Background	214
4.79	L-Skeleton Function	215
4.80	Diagram of Exercise 4.20	215
4.81	M-Skeleton Function	216
4.82	Skiz Function	216
4.83	Gray Morphology: Erosion	217
4.84	Diagram of Exercise 4.21	218
4.85	Gray Morphology: Dilation	219
4.86	Comparison of Square and Hexagon Connectivity	219
4.87	Gray Morphology: Opening	220
4.88	Gray Morphology: Closing	220
4.89	Gray Morphology: Proper Opening	221
4.90	Gray Morphology: Proper Closing	221
4.91	Gray Morphology: Auto-Median Function	222
5.1	Line Profile Function in IMAQ Vision Builder	224
5.2	Line Profile of an Image	225
5.3	Diagram of Exercise 5.1	226
5.4	Menu Palette Containing Overlay Functions	226
5.5	Quantifying Image Areas with IMAQ Vision Builder	227
5.6	LabVIEW Quantify VI Generated with IMAQ Vision Builder	228
5.7	IVB (IMAQ Vision Builder) Functions	228
5.8	Centroid Function (Center of Energy)	229
5.9	Linear Average of Pixel Values in x and y Direction	230
5.10	Diagram of Exercise 5.2	230
5.11	Simple Edge Detector	232

5.12	Diagram of Exercise 5.3	232
5.13	Menu Palette Containing ROI Functions	233
5.14	Edge Detection Tool	234
5.15	Diagram of Exercise 5.4	234
5.16	Detecting Peaks and Valleys of a Line Profile	235
5.17	Diagram of Exercise 5.5	236
5.18	Menu Palette Containing Pixel Manipulation Functions	236
5.19	Locating Edges in Images	237
5.20	Locating Horizontal Edges	238
5.21	Locating Horizontal and Circular Edges	238
5.22	Diagram of Exercise 5.7	239
5.23	Edge-Locating Result (Motor)	239
5.24	Distance Indication in Binary Images	241
5.25	Diagram of Exercise 5.8	241
5.26	Distance Function Applied to a Binary Motor Image	242
5.27	Danielsson Function Applied to a Binary Motor Image	242
5.28	Labeling of Binary Images	243
5.29	Diagram of Exercise 5.9	244
5.30	Segmentation of Labeled Binary Images	244
5.31	Diagram of Exercise 5.10	245
5.32	Segmentation Result of the Motor Image	245
5.33	Circle Detection Exercise	246
5.34	Circle Detection Result of the Modified Motor Image	248
5.35	Diagram of Exercise 5.11	248
5.36	Counting Objects in Gray-Scaled Images	251
5.37	Diagram of Exercise 5.12	252
5.38	Measuring Vertical Maximum Distances	253
5.39	Measuring Vertical Minimum Distances	254
5.40	Measuring Horizontal Maximum Distances	254
5.41	Measuring Horizontal Minimum Distances	255
5.42	Basic Particle Analysis of a Filtered Image	256
5.43	Diagram of Exercise 5.14	257
5.44	Complex Particle Analysis of a Filtered Image	258
5.45	Diagram of Exercise 5.15	258
5.46	Calculating Other Complex Particle Parameters	259
5.47	Diagram of Exercise 5.16	260
5.48	Intercept and Chord Particle Measurements	263

5.49	Calibrating the Motor Image	265
5.50	Setting a Simple Pixel Calibration	266
5.51	Diagram of Exercise 5.17	267
5.52	Grid Calibration with IMAQ Vision Builder	267
5.53	Shape Matching with IMAQ Vision Builder	269
5.54	Pattern Matching with IMAQ Vision Builder	271
5.55	Pattern Matching: Decreasing Matching Score	272
5.56	Pattern Matching: Single Match	273
5.57	Part of the Pattern Matching Diagram	274
5.58	Pattern Matching: Multiple Match	275
5.59	Reading an Analog Needle Instrument	276
5.60	Part of the Analog Meter Reading Diagram	277
5.61	Reading a Digital LCD Instrument	278
5.62	Part of the Digital LCD Reading Diagram	279
5.63	Characters to "Unwrap" on a Compact Disc	280
5.64	Bar Code Reading Example (Code 39 Setting)	281
5.65	Bar Code Reading Example (EAN 13 Setting)	282
5.66	Focus Quality Rating with Edge Detection	284
5.67	Focus Quality Diagram (Edge Detection)	285
5.68	Focus Quality Rating with Histogram Analysis	286
5.69	Focus Quality Diagram (Histogram Analysis)	286
5.70	Focus Quality Rating with FFT	287
5.71	Focus Quality Diagram (FFT)	288
5.72	Block Diagram of the Moving Camera System	290
5.73	Prototype of the Moving Camera Unit	291
5.74	Screenshot of a Typical Image Processing Application	293
5.75	LabVIEW Diagram and Front Panel Details	294
5.76	Villach City Hall Square with Interactive Fountain	296
5.77	Fountain Control and NI 6B Modules	297
5.78	Principle of the Object Detection Algorithm	298
5.79	User Interface of the Fountain Control Software	299
5.80	Diagram Window of the Layer Extraction Algorithm	302
5.81	Visualization of NETQUEST Results in 2D and 3D View	303
5.82	Feedback Form Prepared for Automatic Reading	305
5.83	Results of the Form Reader Compared with Original Values	307
5.84	Block Diagram of find <code>mark.vi</code>	308
5.85	Block Diagram of the Main Program	308

[\[Team LiB \]](#)

[!\[\]\(a48045bf840f60e99d28ce32cf91bb81_img.jpg\) PREVIOUS](#) [!\[\]\(e5ed933a14d18cab9d3a93bb1a8ebc31_img.jpg\) NEXT !\[\]\(3719e566d66f65520a8a032eabbbcf20_img.jpg\)](#)

List of Tables

1.1	Summary of Discussed National Instruments' Software Packages	5
1.2	Possible Hardware Extensions for Image Processing PCs	6
2.1	Comparison of CCD and CMOS Technology	52
2.2	Comparison of CCIR and RS 170 Video Standards	55
2.3	Line Numbering in CCIR and RS 170 Video Standards	56
2.4	Comparison of PAL and NTSC Color Video Standards	66
2.5	Standards for Digital Video	66
2.6	Typical Values of the US Reflection Factor R	69
2.7	Typical Values of Relaxation Times T_1 and T_2 at 1 Tesla	77
3.1	Frame Grabbing Methods	80
3.2	Maximum 1394 Data Payload Size	88
3.3	Camera Link Configurations	109
3.4	Comparison of Digital Camera Interfaces	110
3.5	Comparison of Lossless Compression Algorithms	116
3.6	Structure of the BMP File Header	124
3.7	Structure of the BMP Info Header	125
3.8	Structure of the BMP RGB-QUAD Block	126
3.9	Structure of the GIF Header	127
3.10	Structure of the GIF Logical Screen Descriptor	127
3.11	Structure of the GIF Image Descriptor Block	128
3.12	Structure of a GIF Raster Data Block	128
3.13	Structure of the TIFF Header	129
3.14	TIFF IFD Block Structure	129
3.15	TIFF Tag Structure	130
3.16	Tag Data Types	130
3.17	Image Organization Tags	130
3.18	Image Pointer Tags	131
3.19	Pixel Description Tags	131
3.20	Data Orientation Tags	132
3.21	Data Compression Tags	132
3.22	Document and Scanner Description Tags	132
3.23	Storage Management Tags	132

3.24	Ink Management Tags	133
3.25	JPEG Management Tags	133
3.26	YC _b C _r Management Tags	133
3.27	CHUNK Structure	134
3.28	Header (IHDR) CHUNK	135
3.29	Palette (PLTE) CHUNK	135
3.30	Primary Chromacities and White Point (cHRM) CHUNK	135
3.31	Physical Pixel Dimension (pHYs) CHUNK	136
3.32	Textual Data (tEXt) CHUNK	136
3.33	Image Last Modification Time (tIME) CHUNK	136
3.34	PCX File Header	137
3.35	JFIF SOI Segment	138
3.36	JFIF EOI Segment	138
3.37	JFIF APP0 Segment	139
3.38	JFIF Extension APP0 Segment	139
3.39	Define Huffman Table (DHT) Segment	139
3.40	Define Arithmetic Coding (DAC) Segment	140
3.41	Define Quantization Table (DQT) Segment	140
3.42	Comparison of Image Standards	140
3.43	Tags of a DICOM Image Header	144
5.1	Comparison of Focus Quality Rating Methods	288
5.2	Classes Used in NETQUEST	303

[[Team LiB](#)]

[◀ PREVIOUS](#) [NEXT ▶](#)

List of Exercises

<u>Exercise 1.1:</u>	Image Creation	9
<u>Exercise 1.2:</u>	Color Image	11
<u>Exercise 1.3:</u>	IMAQ Vision Builder	22
<u>Exercise 2.1:</u>	Charge Transfer Efficiency	37
<u>Exercise 2.2:</u>	Blooming Effect Simulation	43
<u>Exercise 2.3:</u>	Deinterlacing Images	57
<u>Exercise 2.4:</u>	Color Space Transformation	63
<u>Exercise 2.5:</u>	Iterative Calculation of a CT Image	72
<u>Exercise 3.1:</u>	1394 Camera Images	94
<u>Exercise 3.2:</u>	USB Camera Images	105
<u>Exercise 3.3:</u>	Calculating DCT Coefficients	118
<u>Exercise 3.4:</u>	DICOM Communication	146
<u>Exercise 3.5:</u>	Importing DICOM Images	149
<u>Exercise 4.1:</u>	Histogramand Histogram	152
<u>Exercise 4.2:</u>	Look-up Tables	155
<u>Exercise 4.3:</u>	Equalizing Images	160
<u>Exercise 4.4:</u>	Manual Creation of LuTs	160
<u>Exercise 4.5:</u>	BCG Look-up Table	162
<u>Exercise 4.6:</u>	Filter Kernel Movement	164
<u>Exercise 4.7:</u>	IMAQ Vision Filter Kernels	166
<u>Exercise 4.8:</u>	Frequency Representation of Images	177
<u>Exercise 4.9:</u>	Truncation Filtering	180
<u>Exercise 4.10:</u>	Attenuation Filtering	182
<u>Exercise 4.11:</u>	Thresholded Image	185
<u>Exercise 4.12:</u>	Auto Thresholding	187
<u>Exercise 4.13:</u>	Morphology Functions	191
<u>Exercise 4.14:</u>	Removing Particles	202
<u>Exercise 4.15:</u>	Rejecting Border Particles	202
<u>Exercise 4.16:</u>	Particle Filtering	203
<u>Exercise 4.17:</u>	Filling Holes	209
<u>Exercise 4.18:</u>	Creating Convex Particles	209
<u>Exercise 4.19:</u>	Separating Particles	212

<u>Exercise 4.20:</u>	Skeleton Images	213
<u>Exercise 4.21:</u>	Gray Morphology Functions	217
<u>Exercise 5.1:</u>	Line Profile in Images	223
<u>Exercise 5.2:</u>	Linear Averages	229
<u>Exercise 5.3:</u>	Simple Edge Detector	231
<u>Exercise 5.4:</u>	Complex Edge Tool	233
<u>Exercise 5.5:</u>	Peak-Valley Detector	233
<u>Exercise 5.6:</u>	Finding Horizontal Edges	235
<u>Exercise 5.7:</u>	Finding Circular Edges	237
<u>Exercise 5.8:</u>	Distance and Danielsson	240
<u>Exercise 5.9:</u>	Labelling Particles	243
<u>Exercise 5.10:</u>	Segmentation of Images	243
<u>Exercise 5.11:</u>	Finding Circles	246
<u>Exercise 5.12:</u>	Counting Objects	251
<u>Exercise 5.13:</u>	Clamping Distances	252
<u>Exercise 5.14:</u>	Basic Particle Analysis	255
<u>Exercise 5.15:</u>	Complex Particle Analysis	257
<u>Exercise 5.16:</u>	Choosing Additional Measurements	259
<u>Exercise 5.17:</u>	Image Calibration	266

[[Team LiB](#)]

◀ PREVIOUS [NEXT ▶](#)

Preface

The book you hold in your hands is part of National Instruments and Prentice Hall PTR's Virtual Instrumentation series, covering the toolbox and function library IMAQ™ Vision, the IMAQ Vision Builder, and the NI Vision Builder for Automated Inspection, which are used for image processing, image analysis, and machine vision. It is intended for engineers and professionals, as well as for students, who want to take their first steps in the fields of image processing. Today, many engineers have a lot of experience with LabVIEW™, mostly with data acquisition (DAQ); so they can now also use this tool for their image processing or machine vision tasks.

In this book, I have tried to combine the image processing and analysis functions with a basic knowledge of imaging fundamentals, like image generation, image transport, image storage, and image compression. Although I know that not all of the tasks my readers have to deal with require this knowledge, these sections may be a reference for later use.

Some statements on the requirements for the exercises and the examples: you need a *LabVIEW* version 6.0 or higher; actually, I wrote all of the exercises with a 6.0 (or 6i) version (which is obvious especially in the diagram screen shots), but all of them are tested with 6.1 as well. I cannot give any guarantee that the *LabVIEW* and *IMAQ Vision* programs (VIs) work with version 5 or lower (especially the ones from the CD-ROM will not; but if you program them yourself, they may). You can download an evaluation version of *LabVIEW* from www.ni.com.

Additionally, you need, of course, National Instruments' *IMAQ Vision* toolbox. Unfortunately, no evaluation version of *IMAQ Vision* is available (only a multimedia demo), so you have to buy it. The *IMAQ Vision* multimedia demo is part of the attached CD. By the way, do not confuse the *IMAQ Vision* toolbox with NI *IMAQ*, which contains the most important imaging drivers and is part of any *LabVIEW* installation.

Very good tools for most imaging tasks are *IMAQ Vision Builder* and *NI Vision Builder for Automated Inspection* (NI Vision Builder AI). The *IMAQ Vision Builder* helps you build image processing and analysis applications by constructing a script file and converting it into *LabVIEW* and *IMAQ Vision* programs. We will use the *IMAQ Vision Builder* in some of our exercises because in some cases it is easier to get quick and reliable results, although it is possible to program all of those exercises in *LabVIEW* and *IMAQ Vision* as well.

While I was just writing the (what I thought) final lines of this preface, National Instruments released a new tool, the NI Vision Builder for Automated Inspection (NI Vision Builder AI). This stand-alone software makes it even easier to set up and run simple machine vision applications; you do not even have to have *LabVIEW* installed on your system. We will discuss the Vision Builder AI in [Chapter 1](#), although it will not be used for the exercises. You can find an evaluation version of Vision Builder AI on the CD-ROM. (Please read more about the attached CD in *About the CD-ROM* at the end of this book.)

This book does not cover all *IMAQ Vision* functions, especially not all utility functions like image management and manipulation VIs. The reason is that I do not want to provide a second *IMAQ Vision* User Manual. The User Manual is excellent, and it seems to make more sense to me to focus on some interesting and useful functions, which are explained in the book's examples.

Moreover, this book is *not* a guide to good and structured *LabVIEW* programming; some exercises are definitely not good examples. For instance, most exercises in [Chapters 4](#) and [5](#) open an image and an image workspace but do not close them, which really hurts a good programmer who learned to write structured software. The reason for not closing the image itself is that the image remains on the desktop and the results are visible. Also, if you do not close the workspace, the image is not corrupted by other open windows of the operating system.

So, hopefully I provided a useful set of fundamentals and exercises covering some of the most common image processing, image analysis, and machine vision tasks. If you have any proposals, questions, or simply comments, please contact me personally at t.klinger@cti.ac.at.

[[Team LiB](#)]

 PREVIOUS  NEXT 

Acknowledgements

A number of people helped me a lot with this book: First of all, Bernard Goodwin, who was my first contact to Prentice Hall and one of my strongest motivations. Special thanks also to Michelle Vincenti and Jane Bonnell, who did a great job making this book a good product.

Another thank you goes to the following people from National Instruments headquarters: Ravi Marawar, Jason Mulliner, and Gail Folkins. I also got very valuable support from Guenther Stefan and the entire staff of the National Instruments Austrian section. Thanks to you all.

Moreover, I would like to give special thanks to Christine Marko and Marvin Hoffland, who spent a lot of their time correcting my English, and to Martin Schauperl, who did most of the work concerning the attached CD-ROM.

Finally, my biggest thanks go to my family: my wife Judith and my two kids, Uschi and Peter. You were very patient with me, even when I spent weekends and nights writing and creating exercises instead of spending my precious time with you.

*Thomas Klinger
Villach, Austria*

[\[Team LiB \]](#)

[!\[\]\(f152152632903f782d05f2953c43b347_img.jpg\) PREVIOUS](#) [!\[\]\(9190417dc825cc417a79995d1514ca2f_img.jpg\) NEXT](#)

Disclaimer

[Warning Regarding Medical and Clinical Use of National Instruments Products](#)

[\[Team LiB \]](#)

[!\[\]\(dfd2df6cc884969130953c94dfde9751_img.jpg\) PREVIOUS](#) [!\[\]\(32b146ef4188bfc72f097d20e61ced60_img.jpg\) NEXT](#)

Warning Regarding Medical and Clinical Use of National Instruments Products

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. Applications of National Instruments products involving medical or clinical treatment can create a potential for death or bodily injury caused by product failure or by errors on the part of the user or application designer. 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.

Chapter 1. Introduction and Definitions

This chapter contains information about some fundamental things you may find useful before we get started. The *Introduction* deals with more details about the book's structure, hardware, software, and network requirements and recommendations. In the *Definitions* section some terms, which are used later in this book, are defined. An introduction to IMAQ™ Vision Builder completes the chapter.

Introduction

Electronic image processing is a rapidly evolving technology. Because of the decreasing prices of digital cameras, image processing and analysis applications are within a price category that was reserved for cheap sensor and measurement arrangements only a few years ago. Using a camera as a "universal sensor" and the appropriate image processing and analysis software, the applications are

- more flexible for reconfiguration,
- nowadays cheaper than a conventional sensor application, and
- easily programmable with modern software tools.

On the other hand, these applications have to compete with the human vision system; sometimes, this is easy competition and sometimes not. For example, consider teaching a two-year-old child the definition of a car (which in most cases is one of the first words a child can say). After a few days, your child can recognize not only cars, but also trucks, vans, pickups, and a lot more. Next, try this with a computer vision system. With current technology, you will find it almost impossible.

The great advantages of image vision systems can be seen, for example, in automation and inspection applications. In a failure mode and effective analysis (FMEA), which is a quality tool for the location and quantification of possible product failures, human visual inspection in a production process will always get a 10 on a scale from 1 to 10; that means that human visual inspection gives the highest risk for product faults. Every machine or computer vision system is more reliable than the human eye.

This is what we have to deal with; not the reliability, but definitions: what to measure; how to measure, count, and analyze which kind of objects; what to recognize; and so on. You will find that intelligent software can cover a lot of the problems mentioned above, yet not all.

Structure of This Book

When I was first confronted with image processing, I found out that I knew very little about some essential things: first of all, how images are generated. I therefore included a chapter called "Image Generation," which explains some fundamentals about cameras, frame grabber cards, and other imaging devices, especially medical devices.

The next problem was this: how do images get from A to B fast enough for the imaging application and how are they stored in B? (Mostly, A refers to a camera or an imaging device, and B is a computer or a single hard drive.) The chapter "Image Distribution," starting at page 79, deals a little bit with the things between A and B—the most common bus systems and protocols for imaging applications.

You will find these four chapters in this book (excluding this one):

- Image Acquisition;
- Image Distribution;
- Image Processing;
- Image Analysis.

I talk about the difference between image processing and image analysis later in this chapter.

In this first chapter I give some fundamental definitions; if you are familiar with them, you can start at chapter two. On the other hand, it is satisfying for an author if professionals read his book and find something interesting in every chapter. Moreover, I list the hardware and software configuration of our laboratory, with which I created the exercises in this book. If you are a computer expert, you will not need this information, and you can skip this part as well.

What you *will* need for further understanding of this book, is fundamental knowledge of LabVIEW™ programming (that means, programming in "G"). LabVIEW is a programming environment, developed by the company National Instruments (Austin, Texas). You should know how to do the following tasks:

- build own programs in LabVIEW;
- distinguish between LabVIEW's different data types;
- create sub-VIs;
- solve simple data acquisition tasks with LabVIEW and the corresponding hardware.

Most of the examples are "drawn" in LabVIEW's graphical programming language, G. G is a data flow language, which also shows some structure chart elements. It should be possible, therefore, even for untrained programmers, to follow the signal and data flow of the programs. You can obtain detailed help regarding programming in G in these publications:

- *LabVIEW User Manual* [1],
- *LabVIEW Measurement Manual* [2],
- *G Programming Reference Manual* [3],
- *IMAQ Vision User Manual* [4], and
- *IMAQ Vision for G Reference Manual* [5].

Finally, this chapter contains a short introduction to IMAQ Vision Builder, a program that might not be so well known. Vision Builder helps you develop your own image processing and analysis tasks; but the usefulness of this program cannot be described in one sentence, so please, try it yourself. Thanks to National Instruments for this great piece of software!

Software and Hardware Requirements

Software and Utilities

You will need the following National Instruments software packages for the completion of the exercises in the five chapters of this book; listed below are the programs that are absolutely necessary:

- LabVIEW 6i or higher;
- IMAQ Vision 6.0 or higher.

Unfortunately, there is no evaluation version of IMAQ Vision; only a multimedia demo (which you can find on the attached CD-ROM or download from www.ni.com). You can download an

evaluation version of LabVIEW from www.ni.com as well.

The next list contains recommended packages from National Instruments; you will not need them to complete the exercises, but they are useful (especially NI Vision Builder for Automated Inspection, because there is an evaluation version available on the attached CD-ROM):

- IMAQ for 1394 Cameras 1.0 or higher;
- IMAQ Vision Builder 6.0 or higher;
- NI Vision Builder for Automated Inspection (AI) 1.0 or higher.

Please do not confuse IMAQ Vision (a toolbox for image processing and analysis tasks) with NI-IMAQ, which contains the drivers for imaging hardware and simple functions for managing and displaying images. NI-IMAQ is part of every common LabVIEW installation.

Complicated, isn't it? [Table 1.1](#) summarizes the packages and provides information about evaluation versions, demo versions, and the software on the attached CD-ROM. As mentioned above, you can download any NI demo or evaluation from www.ni.com.

An introduction to IMAQ Vision Builder and NI Vision Builder AI is part of this chapter and starts at page 15. If you do not have IMAQ Vision Builder, it is just more difficult for you to generate the VIs; the Vision Builder does not provide more functionality! IMAQ for 1394 enables you to use 1394 (FireWire™ digital cameras in your hardware configuration, even within a 1394 network, as described in the following section. (You can find more information about IMAQ for 1394 in [\[6\]](#).)

Table 1.1. Summary of Discussed National Instruments' Software Packages

Name	Version	Demo	Eval.	on CD
LabVIEW	≥ 6.0		X	
IMAQ Vision Toolkit	≥ 6.0	X		X
IMAQ Vision Builder	≥ 6.0			
NI Vision Builder for Automated Inspection (AI)	≥ 1.0		X	X
IMAQ for 1394 Cameras	≥ 1.0			

Finally, here is a list of useful software tools that may help you understand some of the exercises. Most of them are freeware, and you can download them from the links section of the attached CD-ROM.

- MediaChance Hot Pixel Eliminator (<http://www.mediachance.com/digicam/hotpixels.htm>);
- Pegasus ImageXpress (<http://www.pegasustools.com>);
- JPEG 2000 Generation Tool
(<http://www.aware.com/products/compression/jpeg2000.html>);
- AccuSoft DICOM Toolkits (<http://www.accusoft.com>);
- DICOM viewer (e.g., DICOM scope 3.5 or higher at www.otechimg.com/special.php);
-

- Common image processing software (e.g., Corel Photo Paint).

Hardware Configuration

Because of the many different (and possible) PC configurations, it is impossible to give a working guarantee for a specific setup, but here is the configuration we use in our lab.[\[1\]](#)

[1] Our lab is located at the Carinthia Tech Institute, University of Applied Sciences, School for MedIT, Carinthia, Austria.

We run the image processing software I described above on Compaq Deskpro Midi-Tower PCs (PIII/1 GHz). The fully equipped laboratory contains eight of them, so the maximum number of students that can work there at the same time is 16. These PCs are equipped with different PCI extension cards, which have been selected for various tasks and enable the use of various additional devices. [Table 1.2](#) gives an overview.

Table 1.2. Possible Hardware Extensions for Image Processing PCs

Extension	Supplier	Function
IMAQ PCI-1407	NI	monochrome frame grabber
IMAQ PCI-1408	NI	monochrome and still color frame grabber
IMAQ PCI-1411	NI	color frame grabber
IMAQ PCI-1424	NI	digital camera image acquisition board
IMAQ PCI-1428	NI	Camera Link image acquisition board
PCI 1394 OHCI	any OHCI	IEEE 1394 PCI card
DFW-V300	Sony	IEEE 1394 digital color camera
DFW-VL500	Sony	IEEE 1394 camera with zoom and focus
XC-75CE	Sony	monochrome analog camera
MC1301	Microtron	CMOS high-resolution camera (Camera Link)

Additional Image Sources

To cover the medical part, we included some diagnostic imaging devices in our lab; we use images of them in some of the exercises later in [Chapter 4](#) and [5](#), and we have equipped some workplaces with visual presenters:

- Ultrasound Imager (Hitachi EUB-310, [Figure 1.1](#));

[Figure 1.1. Ultrasound Imager \(left\) and Refractometer \(right\)](#)



- Ophthalmologic Refractometer/Keratometer (Canon R10, [Figure 1.1](#));
- Scientific Microscope (Leica DMLA, [Figure 1.2](#));

Figure 1.2. Scientific Microscope (left) and Visual Presenter (right)



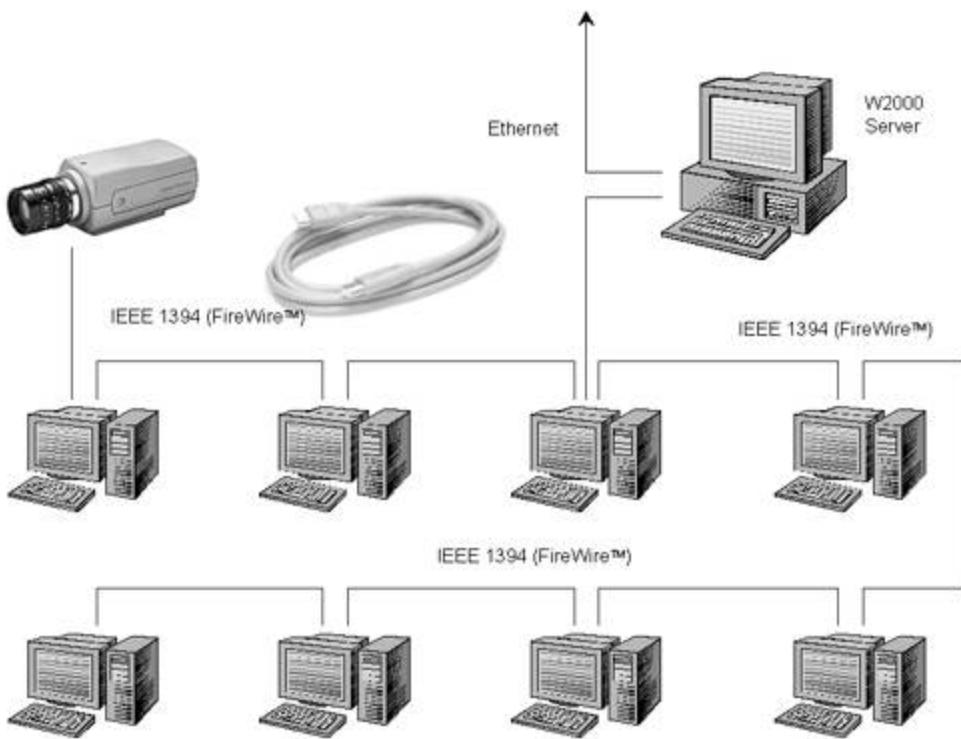
- Visual Presenter (Elmo EV-2500AF PAL, [Figure 1.2](#)).

We use the visual presenters because they are a convenient system consisting of a color camera, a rack, and two adjustable light sources, which are sufficient for a number of imaging applications (see [Figure 1.2](#)). Moreover, zoom and focus of the camera can be controlled over the built-in RS-232 interface, which makes the visual presenter ideal for the development of our own autofocus algorithms, as in the respective section of [Chapter 5](#).

Network Configuration

As can be seen in [Table 1.2](#), every PC is equipped with an IEEE 1394 OHCI card. This enables not only the use of IEEE 1394 cameras, but also the network connection through IEEE 1394. You can find more information about this bus system in [Chapter 3](#). [Figure 1.3](#) shows an example for a network using IEEE 1394 for connecting the PCs among each other as well as for connecting digital 1394 cameras.

Figure 1.3. Network Structure with Simultaneous Use of Ethernet and IEEE 1394



The network connection through IEEE 1394 has the following advantages:

- Digital IEEE 1394 cameras, which are connected to the 1394 network, can provide their data to all of the PCs in the 1394 network that have the appropriate drivers installed.
- As explained in [Chapter 3](#), IEEE 1394 provides isochronous data transmission, which is required for real-time as well for video applications.
- Finally, the use of an appropriate driver software[\[2\]](#) enables full TCP/IP over IEEE 1394. [Figure 1.3](#) shows that in this case a Windows 2000 server is necessary to connect the 1394 network with the rest of the world.

[2] Examples are FireNet™ 2.01 by Unibrain, Greece or the built-in functionality of Windows XP.

Some Definitions

What Is an Image?

Naturally, we are only interested in a definition of "image" that is essential for image processing. A common method is to define an *image* / as a rectangular matrix (called image matrix)

Equation 1.1

$$\mathcal{I} = [s(x,y)]$$

with image rows (defining the row counter or row index x) and image columns (column counter or column index y). One row value together with a column value defines a small image area called pixel (from picture element), which is assigned a value representing the brightness of the pixel.

One possibility is to assign gray-scale values $s(x,y)$ of the *gray-scale set* $G = \{0, 1, \dots, 255\}$. The gray-scale value 0 corresponds to black and 255 to white. Such an image is called an 8-bit gray-scale image with

Equation 1.2

$$s(x,y) \in G$$

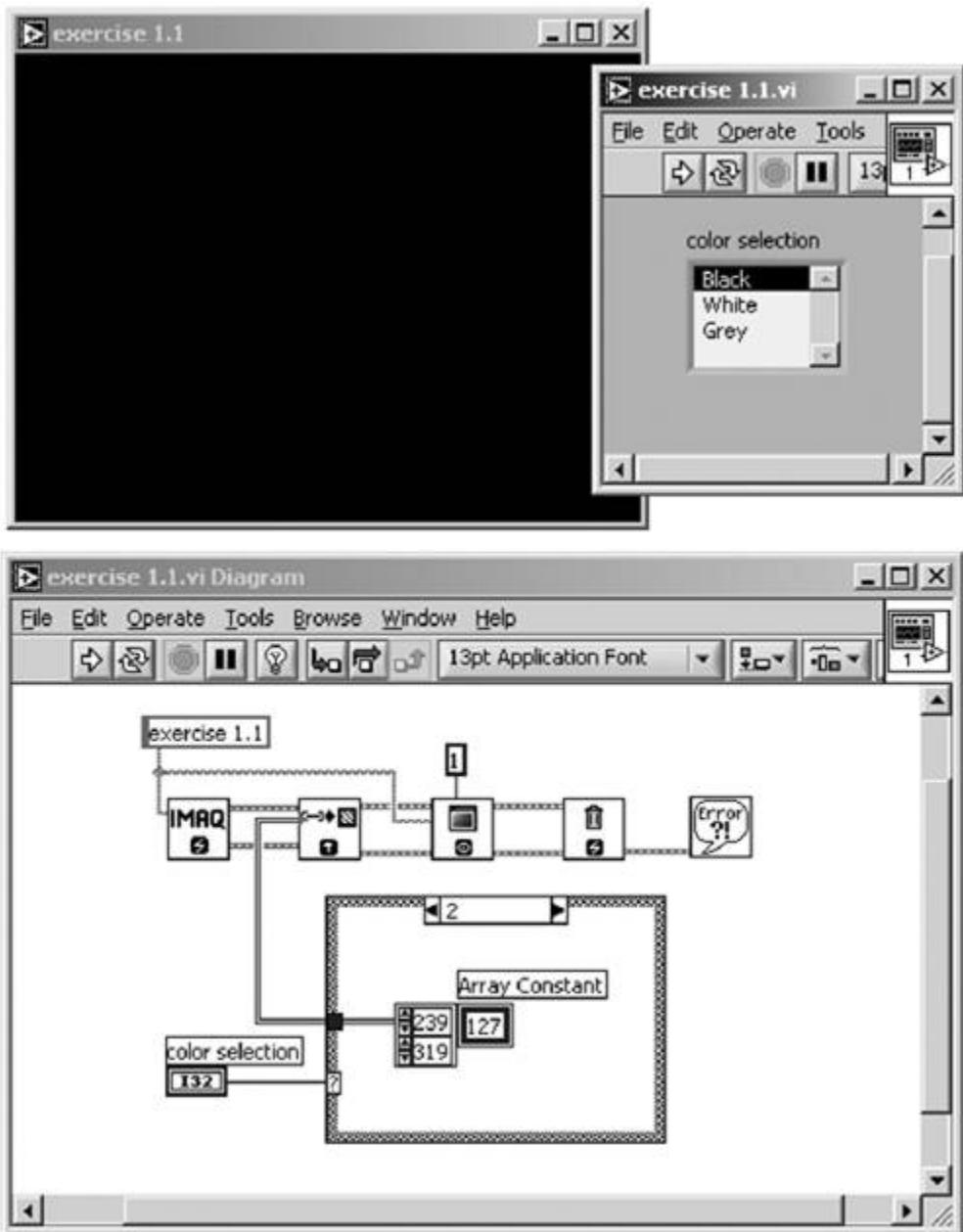
Exercise 1.1: Image Creation.

Create the LabVIEW program shown in [Figure 1.4](#). It shows that a simple image can be created by a rectangular matrix. The IMAQ function `ArrayToImage` provides the conversion from data type "array" to data type "image." Now you can assign the matrix elements (identical) values between 0 and 255.

Moreover, [Figure 1.4](#) shows the general handling of the data type "image" in LabVIEW and IMAQ Vision. The first step is to reserve a memory area with the function `IMAQ Create`; when the program stops, the memory area should be released again with `IMAQ Dispose`. The image itself is visible only if you show it on the screen with the function `IMAQ WindDraw`. Note that IMAQ Vision uses additional windows for images; the front panel window of LabVIEW does not offer any controls or indicators.

The image size used in this exercise is 320 x 240 pixels, that is, a quarter of VGA resolution of 640 x 480 pixels, and is common with video cameras. Note that the matrix indices in LabVIEW start with 0; the maximum row and column indices therefore have the values 319 and 239.

Figure 1.4. Definition of an Image as a Rectangular Matrix



The assignment to the 8-bit gray-scale set \mathcal{G} is arbitrary. Sometimes an image pixel is represented by less than the 256 gray-scale values; a *binary image* consists only of the values 0 and 1 (black and white). Therefore, 1 bit per pixel can be sufficient for image description.

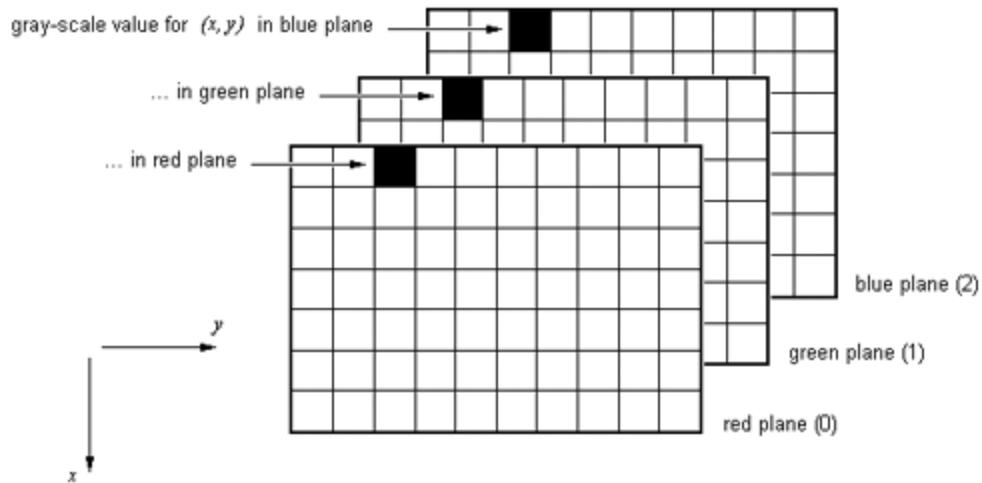
On the other hand, sometimes 256 values may not be enough for displaying all of the information an image contains. Typically, a *color image* cannot be represented by the model shown in Eq. (1.1). In that case, we can extend the image / by using more *planes* and then we talk about *multiplane images* :

Equation 1.3

$$\mathcal{I} = [s(x, y, n)]$$

with n as the *plane counter*. Figure 1.5 shows a three-plane image used for storing color information; the color red is assigned to plane 0, green to plane 1, and blue to plane 2. A single pixel of an N -plane image is represented by an N -dimensional vector

Figure 1.5. Definition of a Color Image as Multiplane Image



Equation 1.4

$$\vec{s}(x, y) = \begin{pmatrix} g_0 \\ g_1 \\ \vdots \\ g_{N-1} \end{pmatrix},$$

where the components g_n are elements of the gray-scale set \mathcal{G} [12].

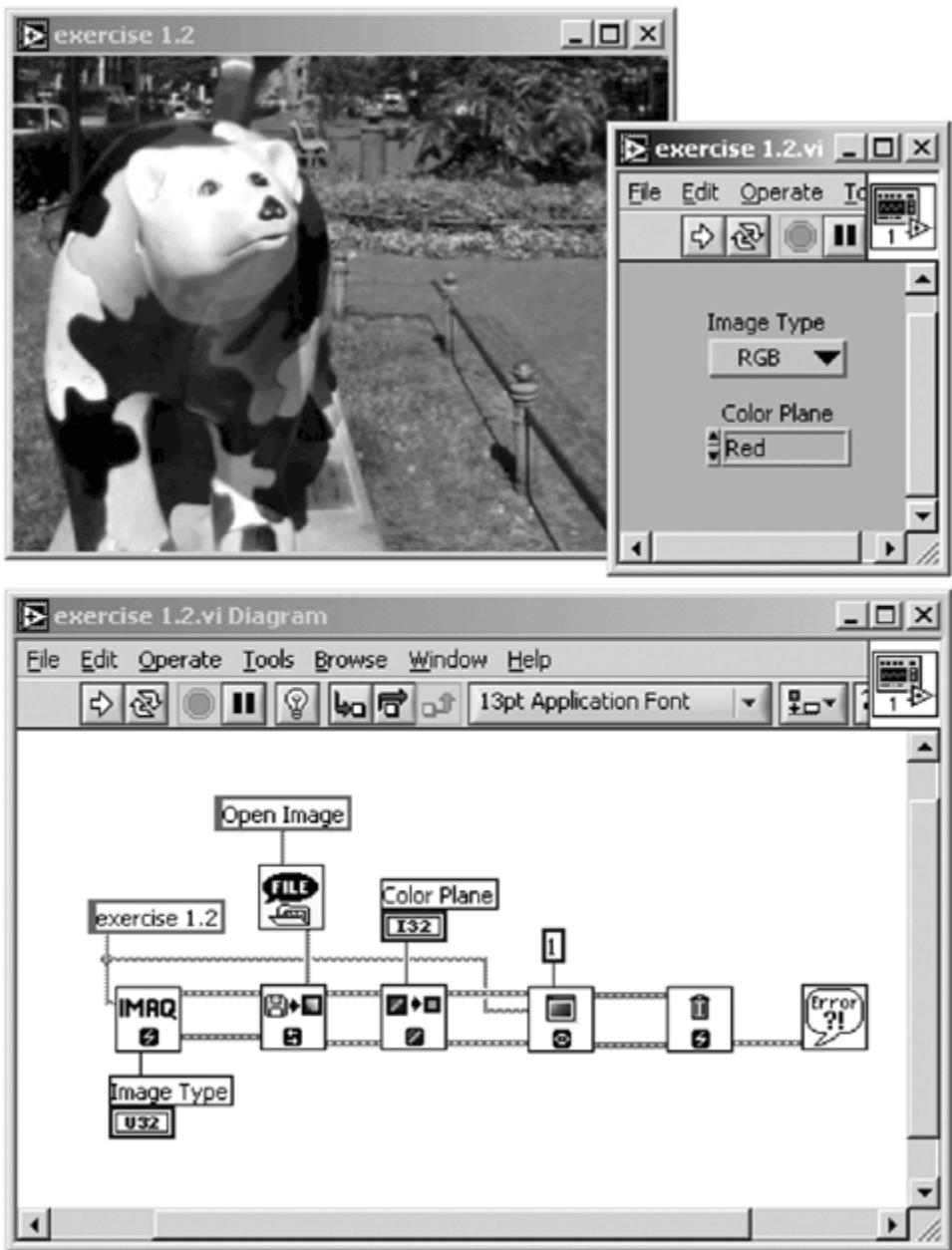
Note that [Figure 1.5](#) also shows position and orientation of the row and column indices. The starting point for both is the upper-left corner of the image; the row index x increases to the bottom, the column index y to the right.

Exercise 1.2: Color Image.

Create the LabVIEW program shown in [Figure 1.6](#). Read in any desired color image through the dialog box; it is important that the image be defined as an RGB image by the function [IMAQ Create](#); otherwise, the planes cannot be extracted.

Select the desired image plane with the control element "color plane," watch the display information, and compare it with the real colors of the image. You can try the planes Hue, Saturation, Luminance, Value, and Intensity as well; they are related to other color models, which are discussed later in this book and may give better results in color recognition tasks.

Figure 1.6. Definition of an RGB-Color Image



The Difference: Image Processing or Image Analysis?

We have already heard these terms several times in this book. Usually, it can be assumed that both expressions have about the same meaning, but if we go deeper into detail, some differences can be found.

First of all, please notice that the following definitions are subjective; they are mine, and they differ a little bit from those of National Instruments. Nevertheless, I believe they make sense.

I use the term *image processing* for all manipulations on an image (as defined above) if the output of the manipulation is again an image. For example, take a look at [Figure 1.7](#). It shows two images; the right one is the result of manipulation of the left one. It is easy to see (and we will also learn later) that this manipulation influences the brightness and the contrast of the original image. Obviously, the output is an image.

Figure 1.7. Image Processing Example. The right image results from brightness and contrast manipulation of the left one.

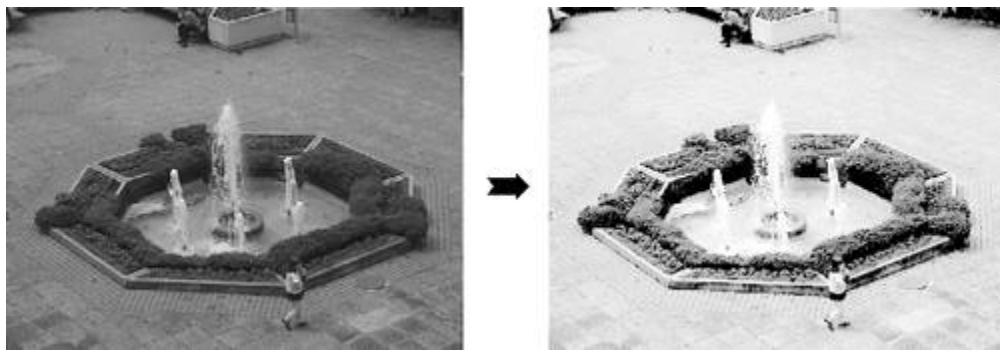
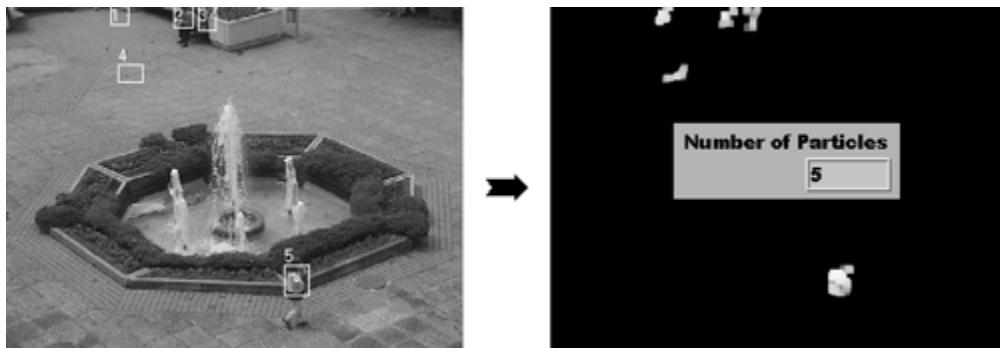


Image analysis is used when the output of a manipulation is *not* an image. Let us look at the example in [Figure 1.8](#). We can see some kind of an image in the right as well, but the result of the manipulation is a number: 5; this means that five objects were detected in the left image.

Figure 1.8. Image Analysis Example. The object detection algorithm returns the number of detected objects in the left image.



The images were taken from an application we discuss in [Chapter 5: Object Detection and Counting in Public Places](#).

Finally: Machine Vision

The term *Machine Vision* or *Computer Vision* is often used for the entire subject, including image processing and image analysis. If we really want to define Machine Vision, we have to first clarify what *vision* itself is.

Undoubtedly, vision is *the* human sense that provides most of the information a human has to process. Rough estimations say that the data rate for continuous viewing is about 10 megabits per second. It is obvious that this huge amount of data cannot be processed in real time (defined in the next section); it has to be filtered by the visual cortex.

We define Machine Vision as high-level applications that come close to human vision capabilities (remember the car example); but we should always keep in mind that these human capabilities for recognition will not be reached by Machine Vision; for reliability, speed and accuracy on the other hand, they will.

Real Time or "Really Fast"?

Mostly, it is assumed that a system running under real-time conditions should react "really fast," thus providing reaction times that are as short as possible. It is assumed that a very fast PC using Windows as the operating system is capable of providing real-time conditions.

This assumption is definitely incorrect. *Real time* simply means that every operation has to be completed within acceptable delay times, however the system ensures completion. Therefore,

Windows as a quite "unreliable" operating system is not suitable for most real-time applications.[\[3\]](#)

[3] On the other hand, that does not mean that Windows is "real-time unsuitable" in general.

Virtual Instrumentation

According to a course manual from National Instruments, one of the "creators" of *virtual instrumentation*, this term is defined as

- the use of industry-standard computers,
- equipped with user-friendly application software,
- cost-effective hardware and driver software

that together perform the functions of traditional instruments. The advantages of virtual instrumentation are obvious:

- Virtual instruments are easily adaptable to changing demands;
- the user interface can be adapted to the needs of different users;
- user interface, hardware, and application software are suitable for modular use.

[\[Team LiB \]](#)

A set of small navigation icons typically found in LaTeX Beamer presentations, including arrows for navigation and symbols for search and refresh.

Introduction to IMAQ Vision Builder

This section is not intended to replace the tutorial that comes with IMAQ Vision Builder [7] or even to represent an introduction to vision concepts [8]; the latter are part of this book's [Chapters 4](#) and [5](#). Rather, the purpose of this section is to show you the power of this prototyping tool; again, it is not possible to solve more complicated problems or to obtain more functionality. Everything you can do with IMAQ Vision Builder, you can do with IMAQ Vision itself; the difference lies in the method by which you do it. Further information about IMAQ Vision Builder can be found in:

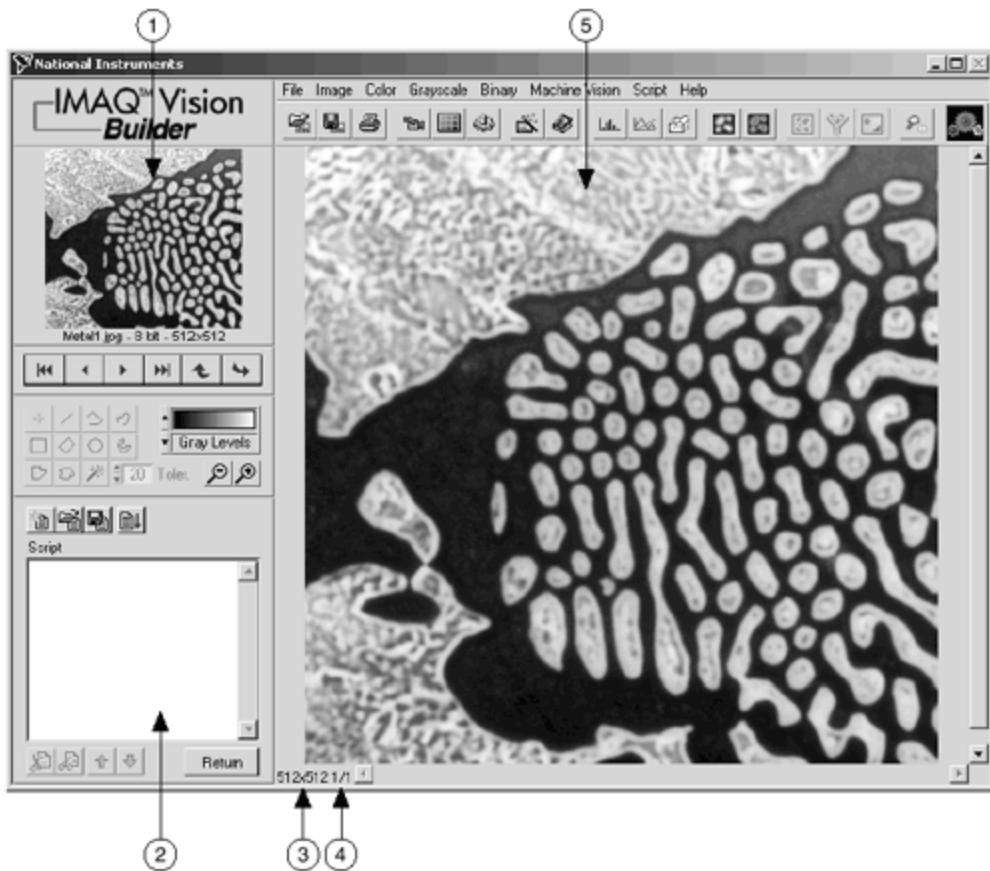
- *IMAQ Vision Builder Tutorial* [7];
- *IMAQ Vision Concepts Manual* [8];
- *IMAQ Vision Builder Release Notes* [9].

These manuals are shipped with the IMAQ Vision Builder software.

IMAQ Vision Builder Environment

[Figure 1.9](#) shows the typical IMAQ Vision Builder environment with a single image loaded. The environment mainly consists of the following parts:

Figure 1.9. IMAQ Vision Builder Environment. 1 Reference Window, 2 Script Window, 3 Image Size, 4 Zoom Ratio, 5 Processing Window
[7]

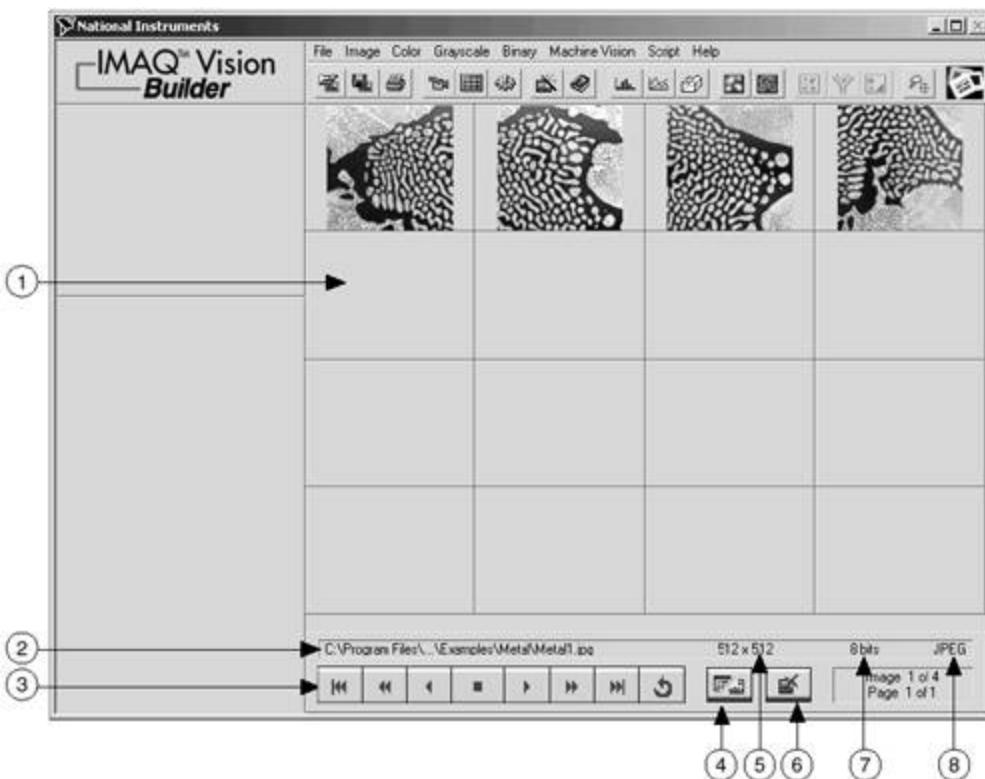


1. The *Reference Window* displays the original version of the image.
2. All manipulations applied to the loaded image can be stored in the *Script Window*.
3. The *Image Size* can be seen here, whereas
4. this area indicates the actual *Zoom Ratio*.
5. The *Processing Window* immediately shows all of your image manipulations.

A benefit of this program is that you cannot get lost in complex menu structures. All of the image processing and analysis happens in the environment of [Figure 1.9](#). Only one different desktop may appear; the *Image Browser* ([Figure 1.10](#)) with the following main elements:

1. The Image Browser itself shows the images in memory in either thumbnail or full-size view;
2. *Image Location* gives the full path to the image file;
3. the *Browse Buttons* are used for browsing the images in memory;
4. the *Thumbnail/Full-Size Toggle* button, if pressed, displays the first image in full-size view or 16 images in thumbnail view, respectively;
5. the *Image Size* is displayed in area 5 of [Figure 1.10](#);
6. *Close Selected Image(s)* does what it says;
7. *Image Type* and
8. *File Format* are the respective indicators.

Figure 1.10. IMAQ Vision Builder Image Browser. 1 Image Browser, 2 Image Location, 3 Browse Buttons, 4 Thumbnail/Full-Size Toggle, 5 Image Size, 6 Close Selected Image(s), 7 Image Type, 8 File Format [7]



Be careful not to confuse the image browser view in full size with the processing mode view; they are quite similar. You need only select a menu item from the Image, Color, Grayscale, Binary, or Machine Vision submenus to change from browser to processing view.

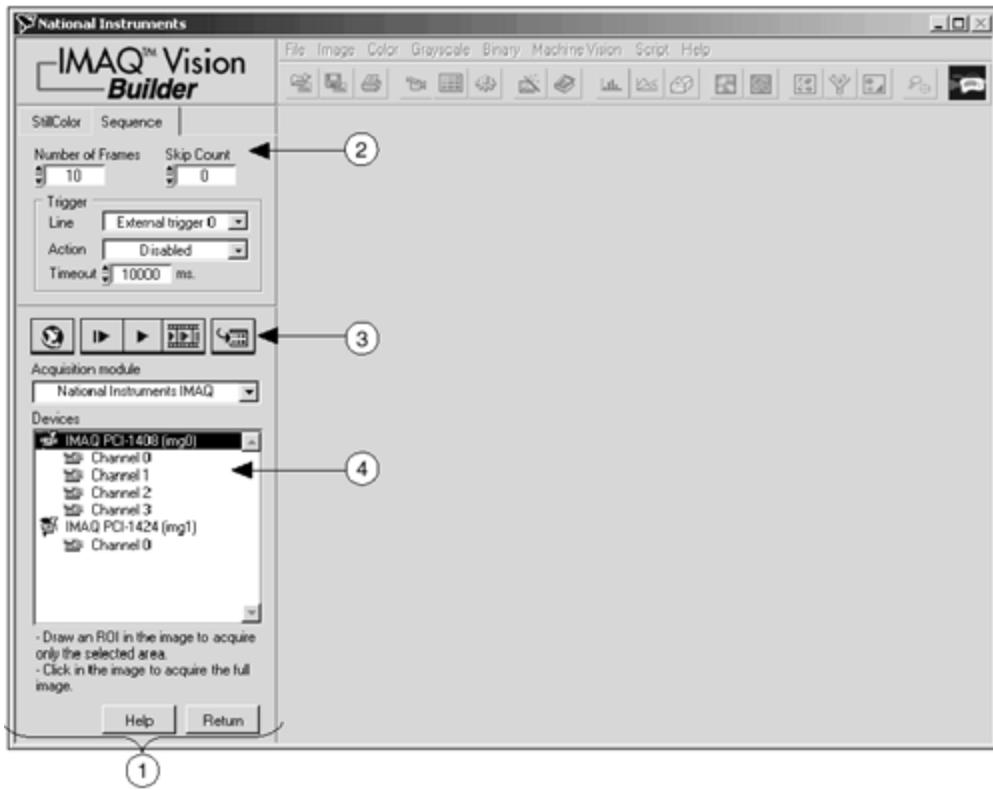
Acquiring Images

IMAQ Vision Builder provides three possibilities for loading images into the image browser:

- loading an image from a file;
- acquiring an image from an image data source (e.g., a camera);
- using the built-in simulation data.

The most interesting of these three is acquiring images directly from your hardware. Let us look at the example in [Figure 1.11](#).

Figure 1.11. Acquiring Images into IMAQ Vision Builder. 1 Acquisition Window, 2 Acquisition Property Page, 3 Store Acquired Image in Browser Button, 4 IMAQ Image Acquisition Board and Channels [7]



Like National Instruments' Measurement and Automation Explorer (MAX), IMAQ Vision Builder shows your available hardware in a separate window (4 in [Figure 1.11](#)). Here, an IMAQ PCI-1408 with four video inputs (analog) can be seen together with a PCI-1424 (a 32-bit digital image acquisition board). After you select your desired image source, three types of image acquisition are offered:

- *Snap*: acquires and displays a single image.
- *Grab*: acquires and displays a continuous sequence (e.g., for camera focusing).
- *Sequence*: similar to grab, with the difference that you can change specific settings. The images are sent to the image browser.

You can select one of the three acquiring options by pressing one of the selection buttons in the middle of the left area.

Depending on which image acquisition board you have installed in your computer, some properties of your acquisition hardware are displayed (2 in [Figure 1.11](#)). This example shows

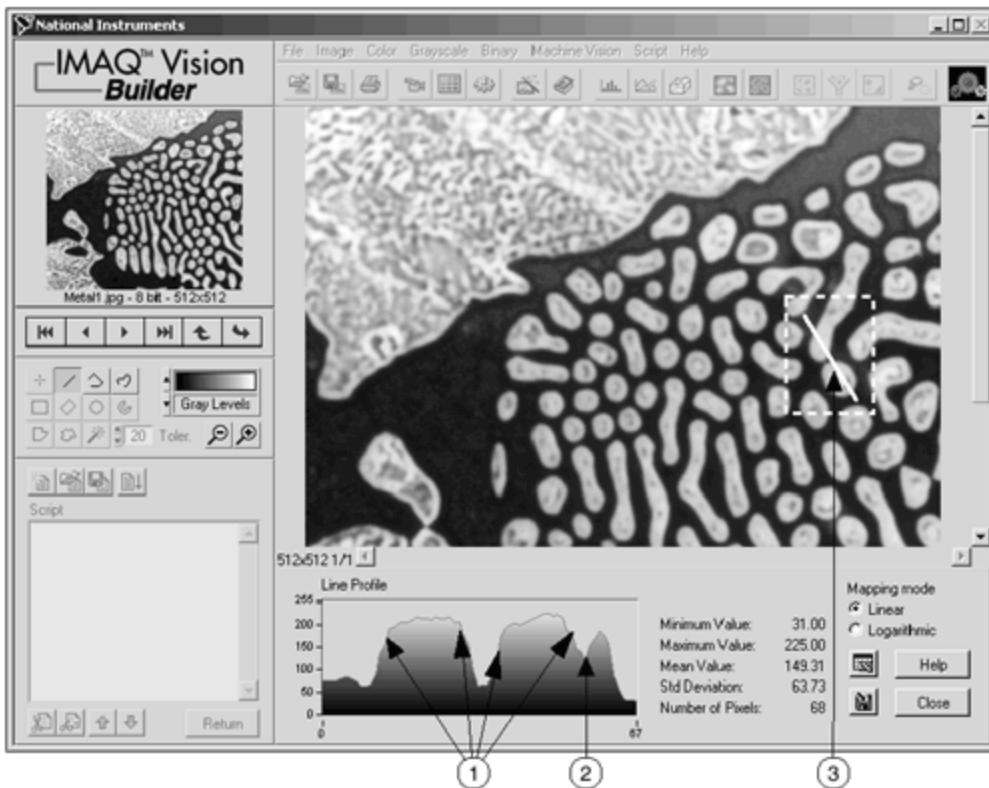
- the *Number of Frames* you want to acquire,
- the *Skip Count* or the number of frames to skip between acquisitions,
- the physical trigger *Line*,
- the triggering *Action*, and
- the *Timeout*, within which the trigger must occur.

Vision Builder Functions

The image processing and analysis functions of IMAQ Vision Builder are identical to those provided by IMAQ Vision itself. [Figure 1.12](#) shows an example, in which a line profile function is

used for detecting object edges.

Figure 1.12. Edge Detection with a Line Profile. 1 Edges of Particles, 2 Fluctuation in Pixel Values, 3 Segment Drawn with Line Tool [7]



After you select Line Profile from the Image menu, the Vision Builder expects you to draw a line in the image. Therefore, it provides you with some simple drawing tools, similar to those of a drawing program like Microsoft Paint. The resulting profile along the line (3 in Figure 1.12) is shown in the bottom area (1 and 2 in Figure 1.12). You will notice three profiles for color images and one single profile for gray-scale images. The line profile function can be used for the (visual) detection of edges along the drawn line. We discuss other methods later.

Figure 1.13 shows another—slightly more complex—example. Here, Pattern Matching, Edge Detection, and Caliper tools collect measurement data from a metal bracket. Pattern Matching detects the holes in the left and the right area of the bracket, Caliper measures the distance between them, and Edge Detection finds the top and bottom edges.

Figure 1.13. Using the Caliper Tool [7]



Here is a list of the most important image analysis and processing functions provided by IMAQ Vision Builder 6.0 (structured by menu items):

- Image
 - Histogram
 - Line Profile
 - Measure
 - 3D View
 - Brightness
 - Image Mask
 - Geometry
- Color
 - Color Operators
 - Extract Color Planes
 - Color Threshold
 - Color Location
 - Color Pattern Matching
 - Color Matching
- Grayscale

- Lookup Table
- Filters
- Gray Morphology
- FFT Filter
- Operators
- Conversion
- Threshold
- Quantify
- Centroid
- Binary
 - Basic Morphology
 - Advanced Morphology
 - Particle Filter
 - Invert Binary Image
 - Shape Matching
 - Particle Analysis
 - Circle Detection
- Machine Vision
 - Edge Detector
 - Find Straight Edge
 - Find Circular Edge
 - Clamp
 - Pattern Matching
 - Caliper

Exercise 1.3: IMAQ Vision Builder.

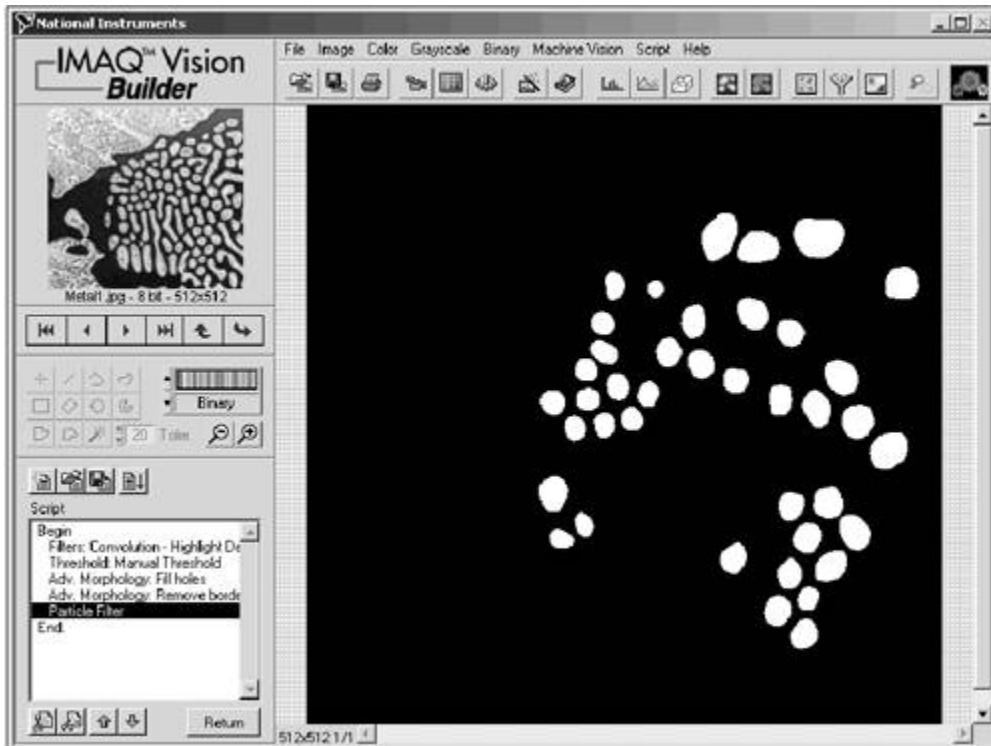
Try the color-plane extraction example from [Exercise 1.2](#). Load the same picture you used in that exercise into IMAQ Vision Builder (Take one with clearly visible areas of different colors and you will obtain interesting results; if you do not have one, take `bear.bmp` from the CD.). Select the Extract Color Planes option from the Color menu and compare the results with those from [Exercise 1.2](#).

Creating Script Files

When you have set up your image analysis, image processing, or machine vision application with IMAQ Vision Builder, you will notice that the Vision Builder might not be the best choice for running your application automatically. Therefore, the Vision Builder provides some possibilities for creating running applications. One of them is the *script file*.

[Figure 1.14](#) shows an example from the IMAQ Vision Builder tutorial [7]. Here, some processing functions are used to find out how many circular "blobs" of a certain size can be seen in the image. Here, we do not talk about the way these functions are working, only how the application can be created.

Figure 1.14. Creating a Script File for Blob Analysis [7]



If the single functions are tested and stored in the script window (usually with the Apply button), the script window will show a number of sequential entries. If you step through these entries or choose Run Script from one of the upper buttons, you will always get the same processing result. Now you can save your script file for later use with the Vision Builder, but, as mentioned above, this is not the best solution. The other possibility is to create a *builder file*, a text file that provides you with enough information to create a LabVIEW VI.

The lines below show the builder file for the application, taken from the tutorial and from [Figure 1.14](#). An experienced LabVIEW user (like you) can program the application easily with this information.

```
IMAQ Vision Builder 6.0
List of functions generated :
Sunday, 16. September 2001 09:34

STEP #1 Filters : Convolution - Highlight Details
IMAQ Vision VI IMAQ Convolute
C Function imgConvolve
Visual Basic Methods CWIMAQVision.Convolute
```

```

Parameters:
Kernel [0] [0] Float (SGL) -1,000000
Kernel [0] [1] Float (SGL) -1,000000
Kernel [0] [2] Float (SGL) -1,000000
Kernel [1] [0] Float (SGL) -1,000000
Kernel [1] [1] Float (SGL) 10,000000
Kernel [1] [2] Float (SGL) -1,000000
Kernel [2] [0] Float (SGL) -1,000000
Kernel [2] [1] Float (SGL) -1,000000
Kernel [2] [2] Float (SGL) -1,000000

STEP #2 Threshold : Manual Threshold
IMAQ Vision VI IMAQ Threshold
C Function imaqThreshold
Visual Basic Methods CWIMAQVision.Threshold
Parameters:
Range.Lower value Float (SGL) 130,000000
Range.Upper value Float (SGL) 255,000000

IMAQ Vision VI IMAQ Cast Image
C Function imaqCast
Visual Basic Methods CWIMAQVision.Cast
Parameters:
Image Type Long (I32) 0

Connections:
Connect output "Image Dst Out" of "IMAQ Threshold"
    to input "Image Src" of "IMAQ Cast Image".
Connect output "error out" of "IMAQ Threshold"
    to input "error in (no error)" of "IMAQ Cast Image".

STEP #3 Adv. Morphology : Remove borders objects
IMAQ Vision VI IMAQ RejectBorder
C Function imaqRejectBorder
Visual Basic Methods CWIMAQVision.RejectBorder
Parameters:
Connectivity 4/8 Boolean TRUE

STEP #4 Adv. Morphology : Fill holes
IMAQ Vision VI IMAQ FillHole
C Function imaqFillHoles
Visual Basic Methods CWIMAQVision.FillHole
Parameters:
Connectivity 4/8 Boolean TRUE

STEP #5 Particle Filter
IMAQ Vision VI IMAQ Particle Filter
C Function imaqParticleFilter
Visual Basic Methods CWIMAQVision.ParticleFilter
Parameters:
Connectivity 4/8 Boolean TRUE
Mode Boolean FALSE
Selection Values[0].Parameter Unsigned Long (U32) 45
Selection Values[0].Lower Value Float (SGL) 1,000000
Selection Values[0].Upper Value Float (SGL) 1,060000
Selection Values[0].Exclude Interval Boolean FALSE

STEP #6 Particle Analysis
IMAQ Vision VI IMAQ ComplexParticle
C Function imaqGetParticleInfo
Visual Basic Methods CWIMAQVision.Particle
Parameters:
Connectivity 4/8 Boolean TRUE

```

```

Number of Particles Long (I32) 0

IMAQ Vision VI IMAQ ComplexMeasure
C Function imaqCalcCoeff
Visual Basic Methods CWIMAQVision.ParticleCoefficients
Parameters:
Parameters[0] Long (I32) 0
Parameters[1] Long (I32) 8
Parameters[2] Long (I32) 9
Parameters[3] Long (I32) 19
Parameters[4] Long (I32) 35
Parameters[5] Long (I32) 45
Coefficients(2D) Long (I32) 0

Connections:
Connect output "Complex Reports"
  of "IMAQ ComplexParticle"
  to input "Complex Reports"
  of "IMAQ ComplexMeasure".
Connect output "error out"
  of "IMAQ ComplexParticle"
  to input "error in (no error)"
  of "IMAQ ComplexMeasure".

```

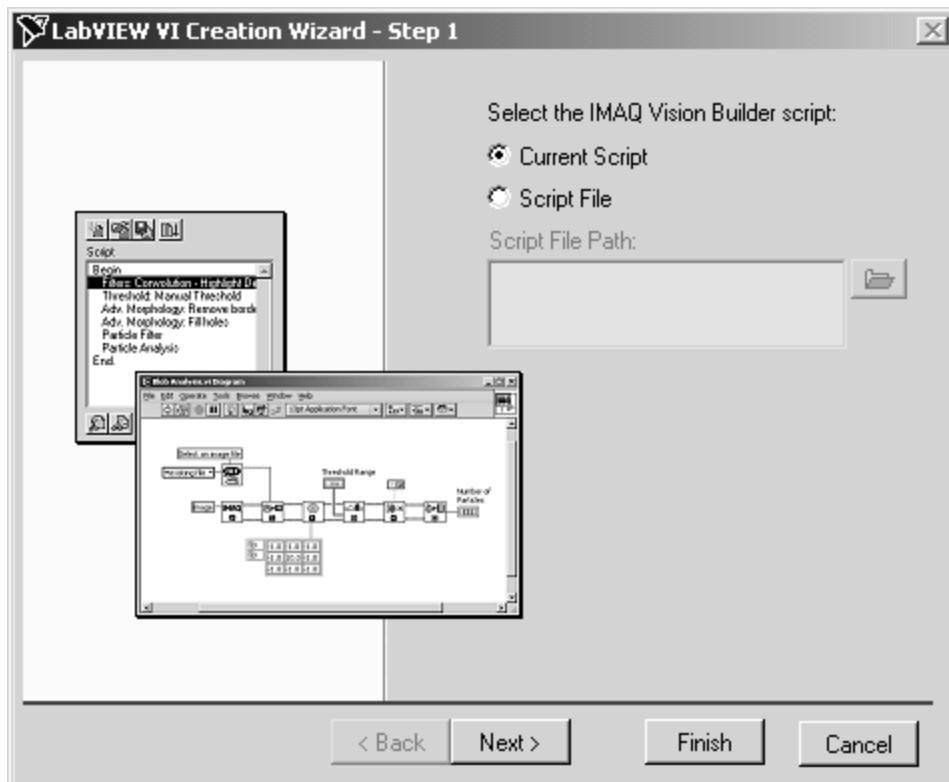
Comment: Display your image with a Binary palette.

Creating LabVIEW VIs

You may argue that this procedure is still a lot of work (and you are right). If you are a lucky owner of IMAQ Vision Builder 6.0, there is a much more convenient solution: the LabVIEW VI Creation Wizard ([Figure 1.15](#)).

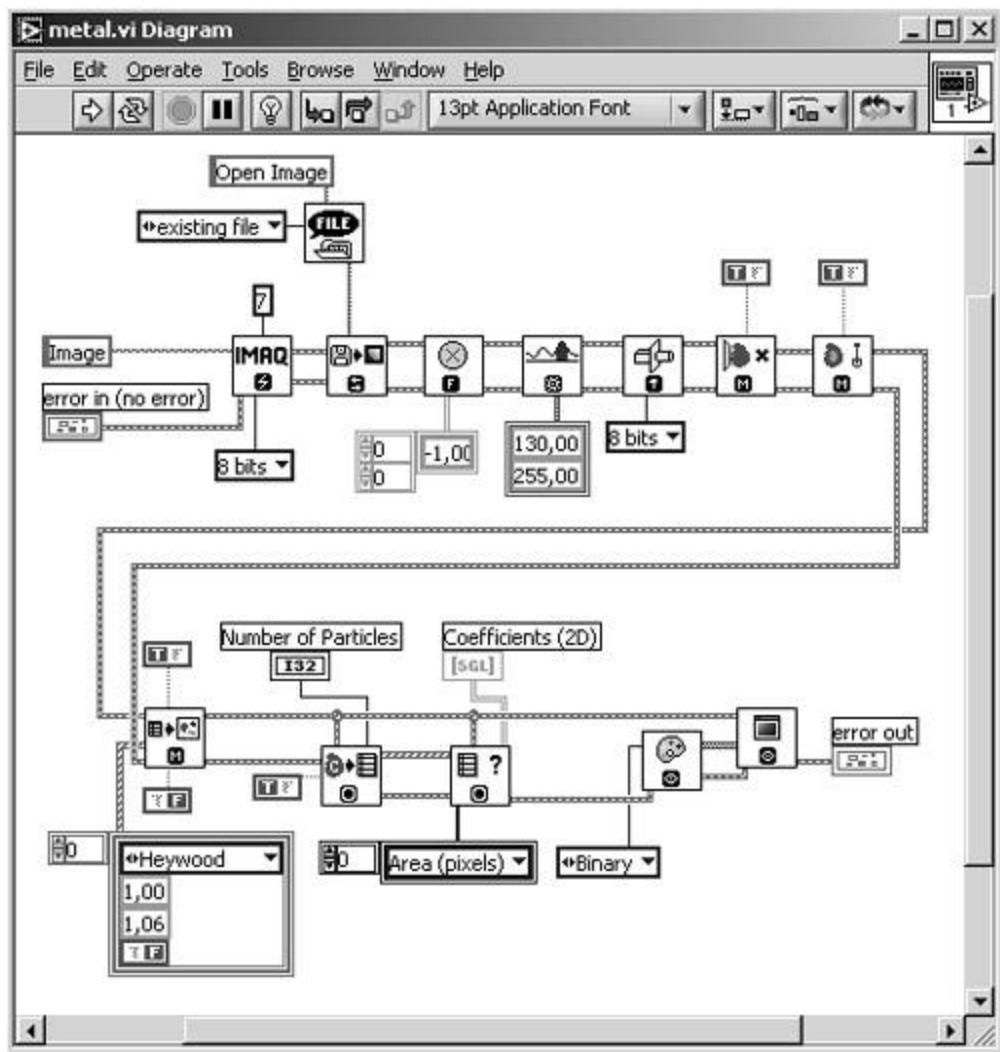
All you have to do is to select Create LabVIEW VI from the Script menu and the first dialog box shown in [Figure 1.15](#) appears. After choosing some settings (for example, if the VI should be created from the current script or another one), you choose whether you want certain values set as constants in your VI or as editable indicators. After you have done this, the VI will be created automatically.

Figure 1.15. LabVIEW VI Creation Wizard



[Figure 1.16](#) shows the VI created from the tutorial script `metal.scr`. I rearranged the icons to fit the figure to the book page. Remarkably, this VI is ready to run and can be created with the help of IMAQ Vision Builder in the total time of about one hour!

Figure 1.16. `metal.vi` Created by the VI Creation Wizard



Finally, I have not mentioned some other procedures IMAQ Vision Builder provides; for example, transferring numerical data (such as that obtained by the Line Profile function) into Microsoft Excel; but I think there should be some things you should find out for yourself. Have fun!

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

NI Vision Builder for Automated Inspection

Another tool from National Instruments is the *NI Vision Builder for Automated Inspection*, or *NI Vision Builder AI*. You can install an evaluation version from the attached CD (or download it from www.ni.com) together with the following documents:

- *NI Vision Builder for Automated Inspection Tutorial* [10],
- *NI Vision Builder for Automated Inspection Release Notes* [11].

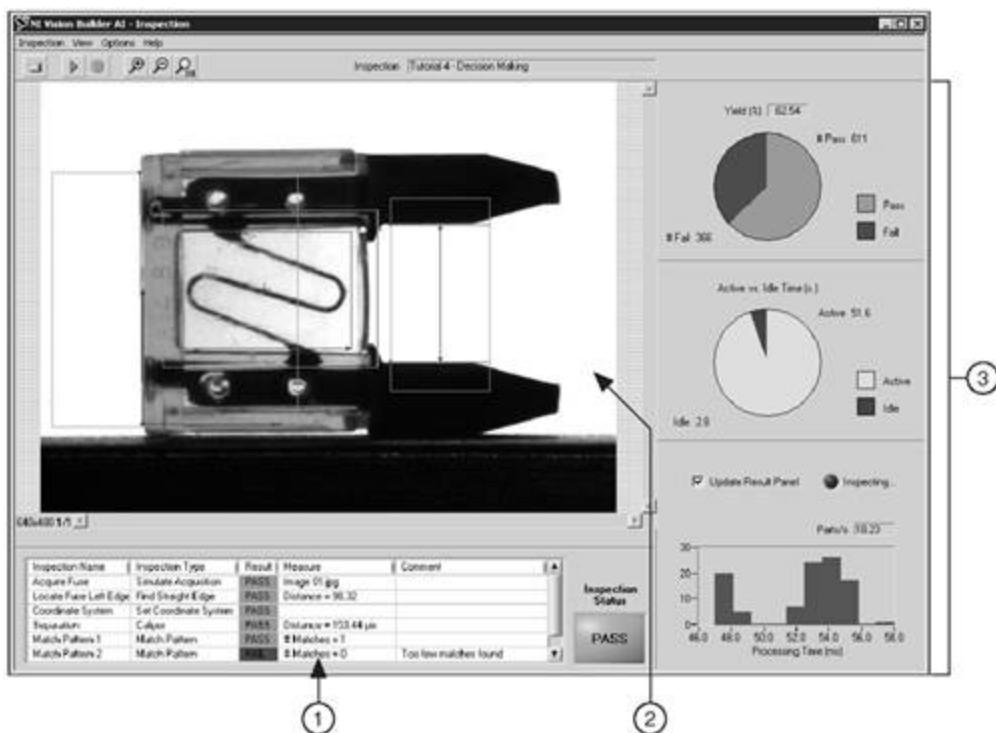
The Vision Builder AI allows for configuration and inspection of complete automated inspection tasks, which can run as a stand-alone software without LabVIEW and IMAQ Vision. With this tool, it is no longer necessary to program such tasks in LabVIEW and IMAQ Vision. Therefore, the NI Vision Builder AI provides two interfaces (or "levels"):

- The *Configuration Interface* (see [Figure 1.17](#));
- The *Inspection Interface* (see [Figure 1.18](#)).

Figure 1.17. NI Vision Builder AI Configuration Interface. 1 Main Window, 2 Inspection Diagram Window, 3 Inspection Steps Palette, 4 Embedded Help Window [10]



Figure 1.18. NI Vision Builder AI Inspection Interface. 1 Results Panel, 2 Display Window, 3 Inspection Statistics Panel [10]



Configuration Interface

The configuration interface of NI Vision Builder AI is used to configure the inspection tasks. As [Figure 1.17](#) shows, the user interface is divided into four areas:

- The *Main Window* basically shows the image that is currently processed; also other items like the Decision Making property page or the Serial I/O property page may be displayed here.
- The *Inspection Diagram Window* displays the already defined inspection steps.
- The *Inspection Steps Palette* provides a menu for the inspection methods. The following menu items are present:
 - Acquire Images
 - Enhance Images
 - Locate Features
 - Measure Features
 - Check for Presence
 - Identify Parts
 - Use Additional Tools
- The *Embedded Help Window*, as its name indicates, gives useful information about the current step.

Very interesting is the menu option Image Assistant (to be found in the menu Enhance Images): If you choose it, you get a user interface similar to the "old" IMAQ Vision Builder with its image processing functions.

- The *Embedded Help Window*, as its name indicates, gives useful information about the current step.

You can configure your inspection application by simply adding inspection steps (Figures 1.19 to 1.21 show some examples). If an inspection step does not make sense in your sequence, the program will tell you. Read more about the details of the inspection steps in [10].

Figure 1.19. Finding a Straight Edge in NI Vision Builder AI [10]

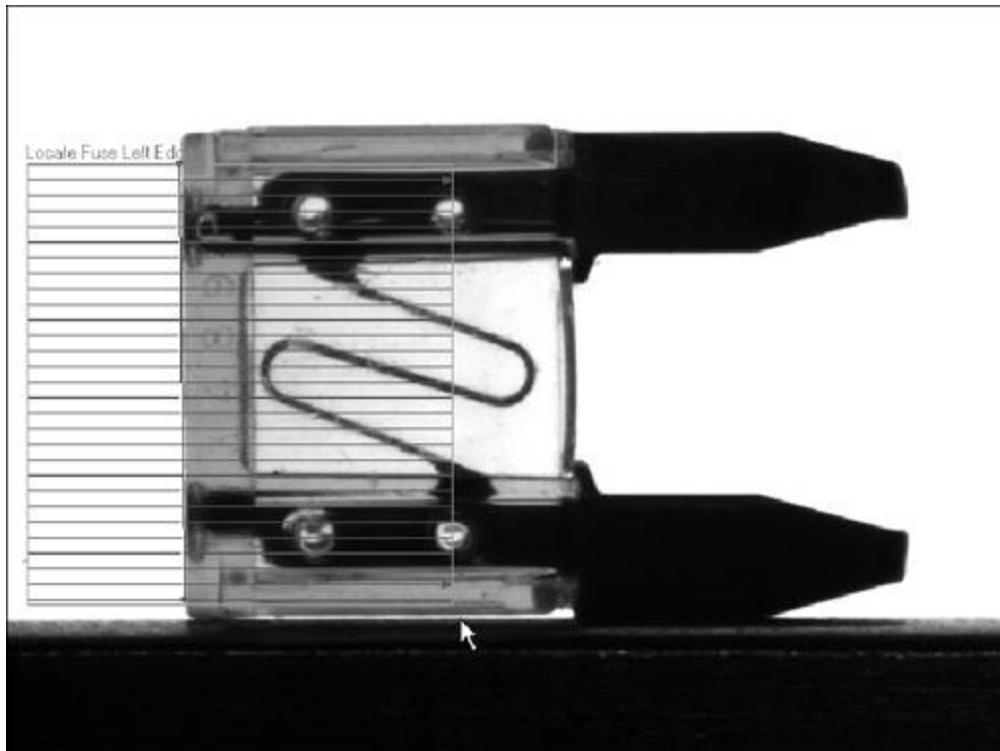


Figure 1.20. Measuring the Distance Between Two Edges [10]

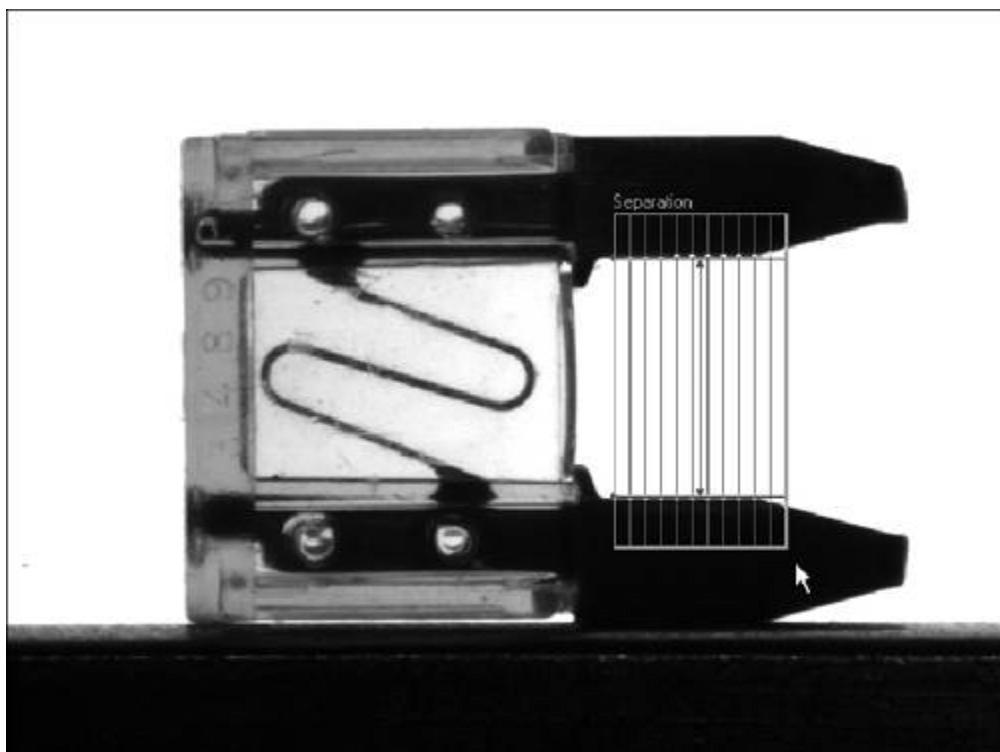
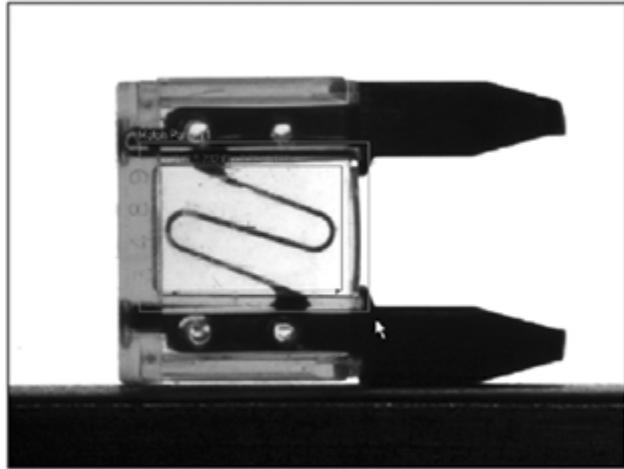


Figure 1.21. Pattern Matching in a Limited Search Region [10]



Inspection Interface

After the inspection application is saved in the Configuration Interface, you can run it in the Inspection Interface. Again, this window is divided into parts:

- The *Results Panel* lists all steps of the inspection process, including name, result (PASS or FAIL), measurements, and comments. It is easy to see whether an object has passed the entire inspection or failed in one or more inspection steps.
- The *Display Window* displays the object under test together with some additional information like search areas.
- The *Inspection Statistics Panel* contains three useful indicators displaying:
 - The yield (the ratio between PASS and FAIL);
 - The ratio between active and idle time;
 - The processing time of the application.

You can switch between the two interfaces by using the menu items File / Switch to Inspection Interface and Inspection / Switch to Configuration Interface.

[\[Team LIB \]](#)

[PREVIOUS](#) [NEXT](#)

Chapter 2. Image Acquisition

This chapter deals with the most interesting issues of image generation. We focus on *CCD* cameras and sensors because they currently occupy the biggest part of the camera market. *CMOS* cameras are discussed with respect to their functional difference, their advantages, and their drawbacks. *Video standards*, analog as well as digital, are described to help you understand the transmission and the display techniques of video signals. Another main part deals with *color* description of images. Finally, *image sources* that do not use cameras are mentioned and explained.

Charge-Coupled Devices

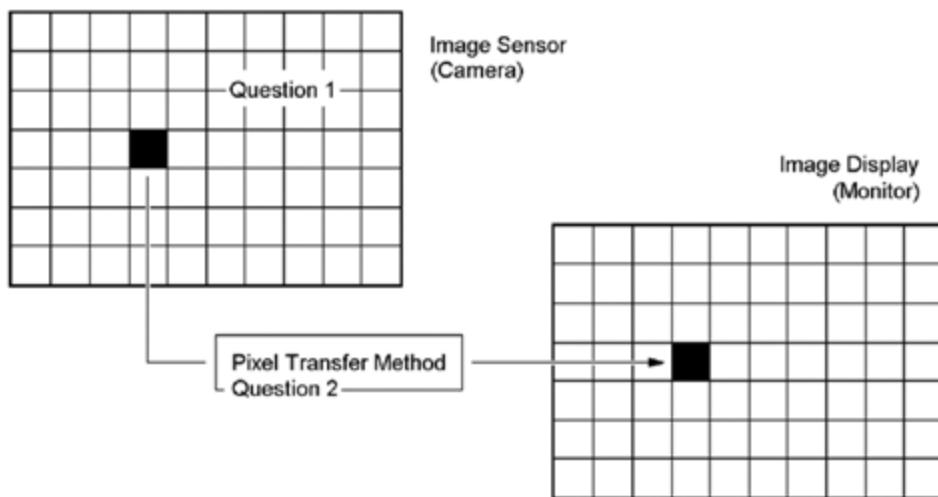
If you consider the matrix structure of an image we discussed in the previous chapter, you might ask yourself the following questions:

1. What should be the look of a sensor that can acquire images of the previously defined matrix type?
2. How is the data transferred between the sensor and the display?[\[1\]](#)

[1] This is a very general expression; I will simply use "computer" or "PC" in the future.

The answer to question 1 seems easy: in general, the sensor will have the same structure as the image itself: the structure of a rectangular matrix, which obviously seems to be the optimal solution. We talk about another possible answer to this question later in this chapter.

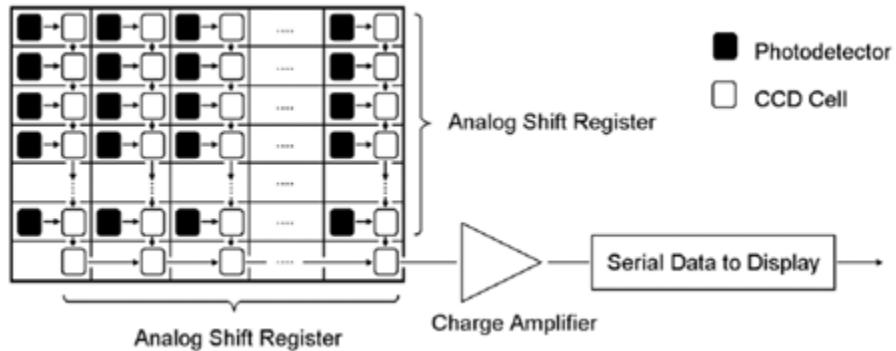
Figure 2.1. Questions Regarding Pixel Transfer



The second question is more difficult to answer; obviously, it is not possible to connect each pixel to the respective area on the pixel display, in most cases, a computer monitor. In this book, we often use image sizes of 320 x 240 pixels; a direct connection would require $320 \cdot 240 = 76,800$ cables between sensor and PC. Therefore, in the first step, the sensor data has to be collected, transferred over cables or other media following a data protocol that is always somehow serial, and finally rearranged for display on a monitor (see [Figure 2.1](#)).

CCDs, or charge-coupled devices, are image sensors that already provide data preparation in the manner mentioned above. Imagine a sensor that consists of a number of photodetectors, one for each pixel. After these photodetectors have measured a certain brightness for the corresponding pixel, this value is shifted vertically into the next row below (see [Figure 2.2](#)).

Figure 2.2. Principle of a CCD Sensor



Of course, the brightness value cannot be shifted from one photodetector to another; we need other components for this operation. Therefore, each photodetector is connected to a MOS capacitor in combination with a transfer gate; [Figure 2.2](#) shows these two elements as a CCD cell. Every time a new value set reaches the bottom row of the image sensor (here, the CCD cells do not have corresponding photodetectors), the values are shifted horizontally to the sensor output.

Principle of Functionality

MOS capacitors (MOS stands for metal oxide semiconductor) can store a number of electrons (described by the electrical charge Q), which were previously generated by the corresponding photodetector, for a certain time. The electrical charge Q results from the photodetector current

Equation 2.1

$$I_{ph} = S \Phi_e ,$$

with

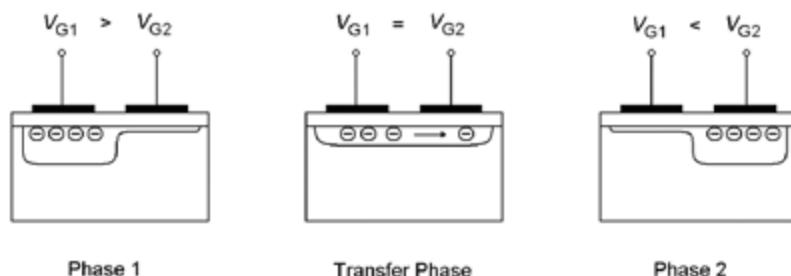
I_{ph} ... photo(detector) current;

S ... light sensitivity, depending on the light wavelength;

Φ_e ... light radiation power;

and is therefore related to the brightness of the light in the respective sensor area.

Figure 2.3. CCD Transfer Mechanism [14]



The transfer mechanism from one CCD cell to another is shown in [Figure 2.3](#). If the voltage level V_{G1} at the gate of the MOS capacitor that contains the electrical charge Q is higher than the voltage level V_{G2} at the gate of the next MOS capacitor, the charge is kept in capacitor 1

(phase 1 or initial phase). If V_{G1} changes from High to Low and V_{G2} changes from Low to High, the electrons are transferred (in this figure) from left to right.

It is quite easy to imagine that if the system shown in [Figure 2.3](#) is continued to the left and to the right, the transfer will not work in a specific direction; the electrons would be transferred to the right side and to the left side as well. Therefore, real CCD systems use more transfer phases to direct the electrons in a defined direction; in [Figure 2.2](#), to the bottom of the sensor.

Unfortunately, some of the electrons are lost during the transfer process; this phenomenon is described by the *charge transfer efficiency* or CTE. This value defines the percentage of charge that is transmitted during a single transfer process; because of the accumulation of this value, CTE should be significantly higher than 99.99%.

[Figure 2.4](#) shows the CTE for an array of MOS capacitors with a gate length of 4 μm , depending on the pulse length of the control signal; here, the pulse length should be at least 1.5 ns in order to guarantee a satisfactory CTE value.

Figure 2.4. Charge Transfer Efficiency (CTE) as a Function of Pulse Length [[14](#)]

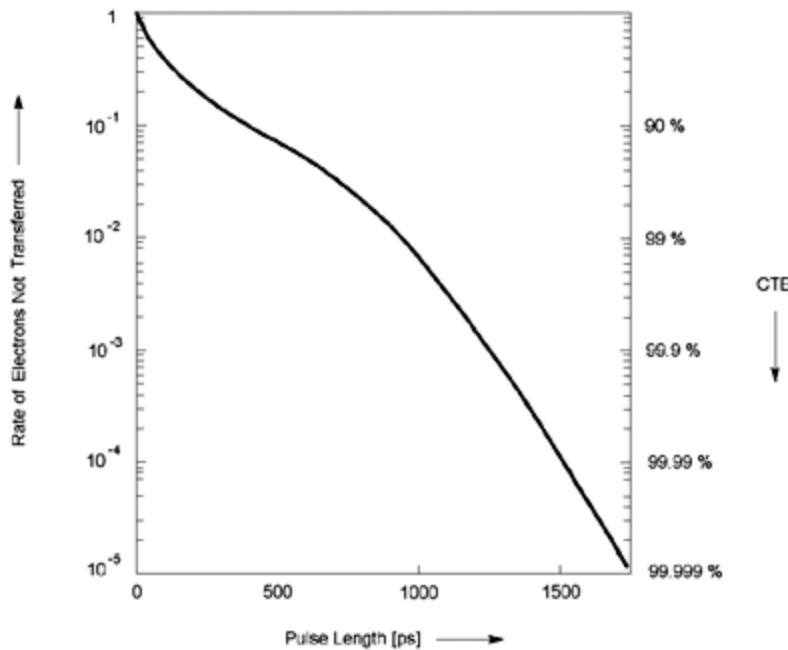
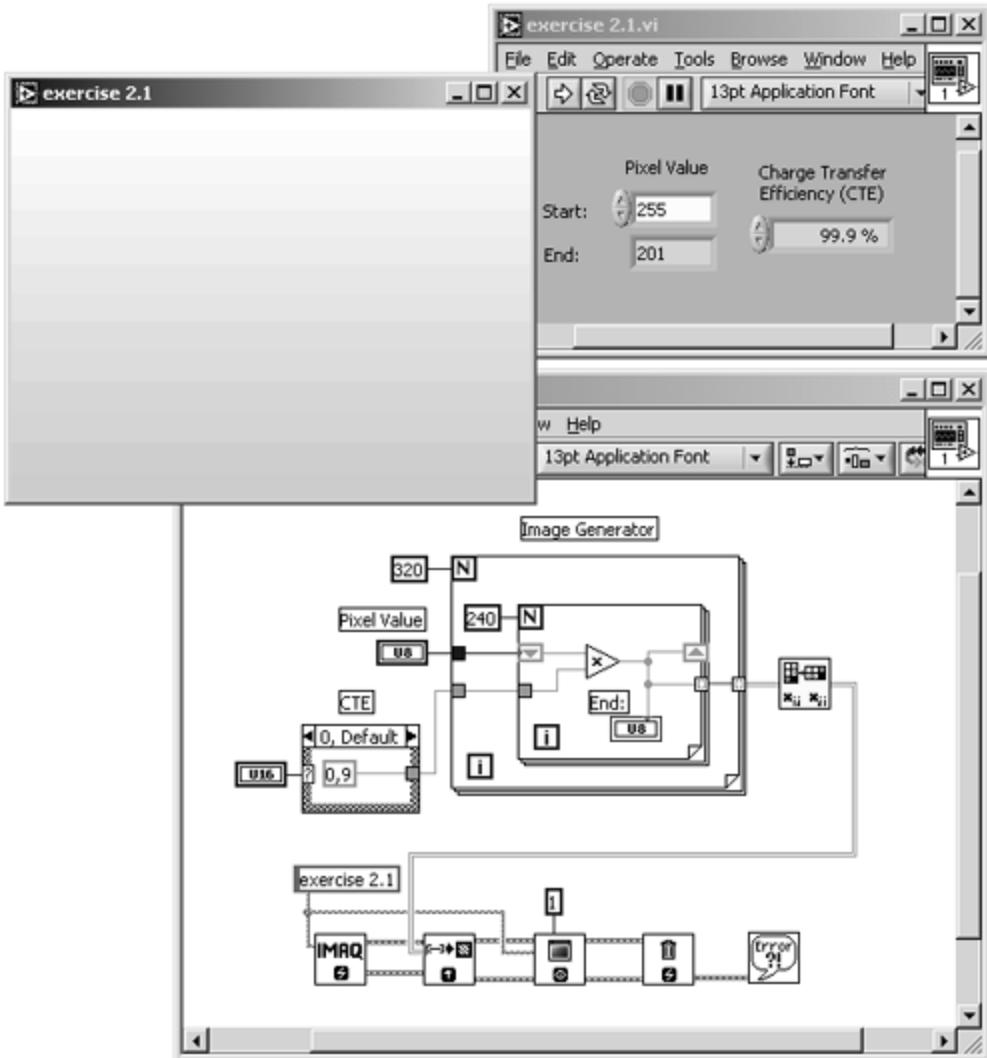


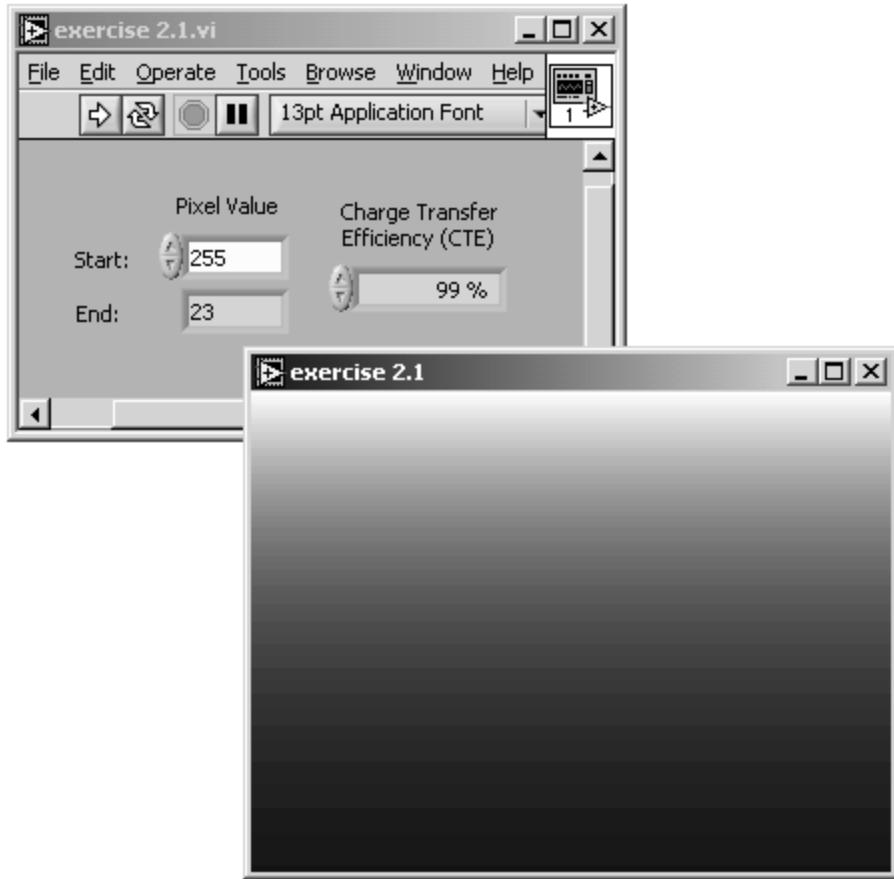
Figure 2.5. Impact of Charge Transfer Efficiency (CTE) on Pixel Brightness



Exercise 2.1: Charge Transfer Efficiency.

Create a LabVIEW VI that visualizes the impact of the charge transfer efficiency (CTE) on the brightness value of a 320×240 image. [Figure 2.5](#) shows a possible solution, in which only the vertical influence of the CTE is taken into account. Change the value of CTE from 90% to 99.999% and watch the decrease of the pixel value in the lowest row, which should be 255. You can see in [Figure 2.5](#) that a CTE value of 99.9% already causes a significant brightness decrease in the lower rows of the image. A CTE of 99%, as shown in [Figure 2.6](#), makes the image unusable.

[Figure 2.6. Charge Transfer Efficiency \(CTE\) Far Too Low](#)

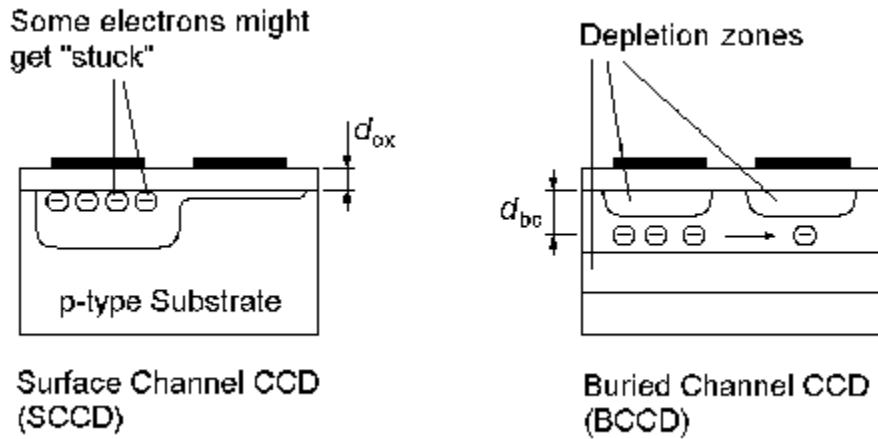


Surface Channel CCDs

The configuration shown in [Figure 2.3](#) is called *surface channel CCD*, or *SCCD*, because the electrons of the charge Q are kept on the surface of the semiconductor area. This leads to a significant disadvantage; a number of electrons "stick" at the semiconductor-oxide junction and are not transferred to the next MOS capacitor. This number of electrons varies statistically and therefore leads to a noise signal, called *transfer noise*.

A possible solution would be not to empty the MOS capacitors completely during the transfer; the fixed electrons can remain on the junction and will not contribute to the transfer noise. Again, reality shows that this does not work in all areas of the capacitor, though the transfer current will decrease, but not disappear.

Figure 2.7. Structure of Surface Channel and Buried Channel CCDs



Buried Channel CCDs

[Figure 2.7](#) shows the structure of a surface channel CCD (left) in comparison with a *buried channel CCD* (or BCCD; right). Here, a thin depletion zone is inserted at the Si-SiO₂ junction. The electron charge is therefore kept between the two depletion zones shown in the right half of [Figure 2.7](#) and can be totally transferred to the next MOS capacitor because the Si-SiO₂ junction, which normally causes stuck electrons, is at a safe distance. This structure leads to a significantly lower transfer noise.

In an SCCD cell, the amount of charge that can be stored in the capacitor is given by the area-specific oxide capacitance C'_ox , influenced only by the oxide thickness d_{ox} and the dielectricity constant of the oxide ϵ_{ox} . In a BCCD, the electrons are kept at a distance d_{bc} from the Si-SiO₂ junction, which will lead to a resulting capacitance

Equation 2.2

$$\frac{1}{C'_{BCCD}} = \frac{1}{C'_{ox}} + \frac{d_{bc}}{2\epsilon_{Si}} ,$$

responsible for the amount of charge that can be stored in a BCCD. That this value is significantly lower than the maximum possible charge in an SCCD is one of the major drawbacks of BCCDs. The relation

Equation 2.3

$$\frac{C'_{ox}}{C'_{BCCD}} = 1 + \frac{\epsilon_{ox}}{\epsilon_{Si}} \frac{d_{bc}}{2 d_{ox}}$$

describes the capacitance reduction in a BCCD, which can be up to 50%.

Another advantage of BCCDs is that the pulse frequency for the transfer process can be significantly higher; *peristaltic CCDs* (PCCDs) can be clocked with some 100 MHz [[14](#)].

Properties

Let us summarize topics we just discussed:

- CCDs or charge-coupled devices consist of at least a single shift register, in most cases of a combination of shift registers, which form a two-dimensional array for the use in image sensors.
- The CCD cells of these shift registers contain an amount of charge, which is related to the brightness[\[2\]](#) at the respective CCD cell and will later be represented by one pixel (picture element).

[2] This implies that the shift register is carrying analog information. In digital systems, we are working with shift registers with digital information, so CCDs are really an exception.

- The shift register concept enables easy parallel-serial conversion by shifting the values to the sensor bottom and reading out a single information line.

The following sections discuss the most relevant properties of CCDs that influence the image quality and therefore influence all image processing and analysis methods.

Sensitivity and Resolution

The *spectral sensitivity* of a CCD is in general determined by the photoelements. Various filter layers can be used to obtain a sensitivity maximum in different spectral regions; this is the way color CCDs are realized: by three photoelements for each pixel and red, green, and blue color filters.

Infrared sensitivity can be obtained if Schottky photoelements are used in the CCD. They can detect wave lengths up to $5.7 \mu\text{m}$ and temperature differences down to 0.02°C [\[14\]](#). Infrared sensitive cameras can be used, for example, in military or medical applications.

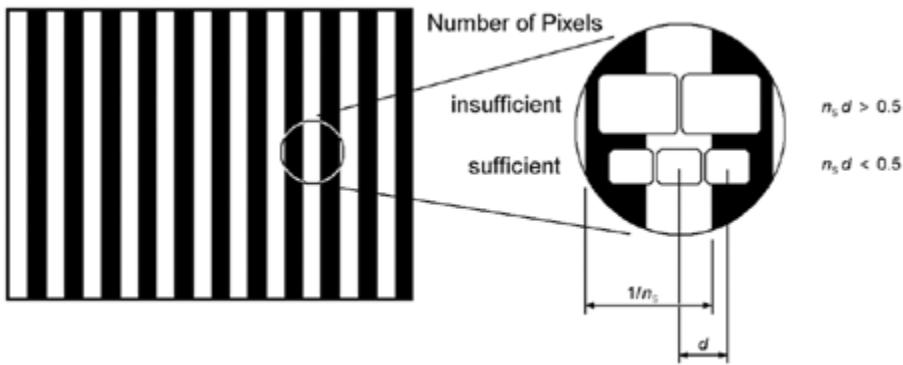
The *resolution capability* of a CCD can be described by the *modulation transfer function*, or MTF. It is defined as the modulation of the output signal if the CCD sensor looks at an image consisting of black and white stripes. If d is the distance between two sensor pixels and n_s the number of black and white stripe pairs, the product d/n_s gives a measure for the resolution capability; if $d/n_s \ll 1$, the resolution of the black and white stripes is easy. For $d/n_s > 0.5$, it is not possible to distinguish between two neighboring lines. The function can be mathematically described by

Equation 2.4

$$\text{MTF}_0 = \frac{\sin(\pi n_s d)}{\pi n_s d}$$

It is easy to see that at least two sensor pixels are required to detect a black-and-white stripe pair (see [Figure 2.8](#)).

Figure 2.8. Visualization of the Modulation Transfer Function (MTF).
How many pixels are needed to distinguish between a certain
number of black and white lines?



The index 0 in Eq. (2.4) indicates that this function is theoretical. The real MTF is influenced by a number of other factors, such as the charge transfer efficiency CTE or light diffraction and dispersion.

Noise and "Hot Pixels"

The *noise* generated in a CCD is described by the *signal-noise ratio* SNR

Equation 2.5

$$\text{SNR} = \frac{n_{\text{signal}}}{n_{\text{noise}}}$$

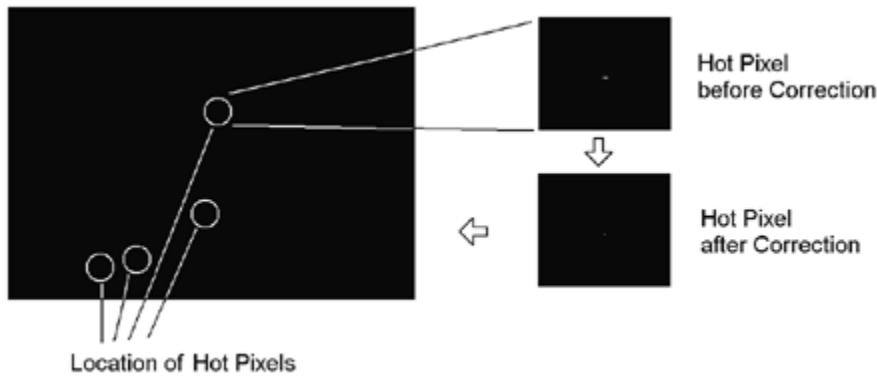
with n_{signal} as the number of electrons that contributes to the image signal and n_{noise} as the number of "noise electrons"; noise can be divided into two main parts:

- The *fixed pattern noise*, FPN, which is caused by the physical differences of the single CCD cells. If all of the photoelements and CCD cells were identical, the FPN would be zero.
 - The *statistical noise*, which can itself be divided into
 - the *photon noise*, given by the square root of the number of signal electrons $\sqrt{n_{\text{signal}}}$;
 - the *CCD noise*, caused by electrons generated in the CCD cells; [3]
- [3] To be exact, we would also have to count FPN as a part of the CCD noise.
- the *amplifier noise*, which is generated in the output amplifier.

A special kind of noise that occurs in almost every CCD camera is caused by *hot pixels*. They have their origins in small contaminations or production faults in the CCD sensor area and lead to a higher charge loss in the respective CCD cell during the exposure time. I took a picture with my own digital camera (Olympus CAMedia C-2000 Z), manual exposure time 1/2s, in a totally dark room and additionally closed the camera objective with my hand. You can find the image on the attached CD-ROM.

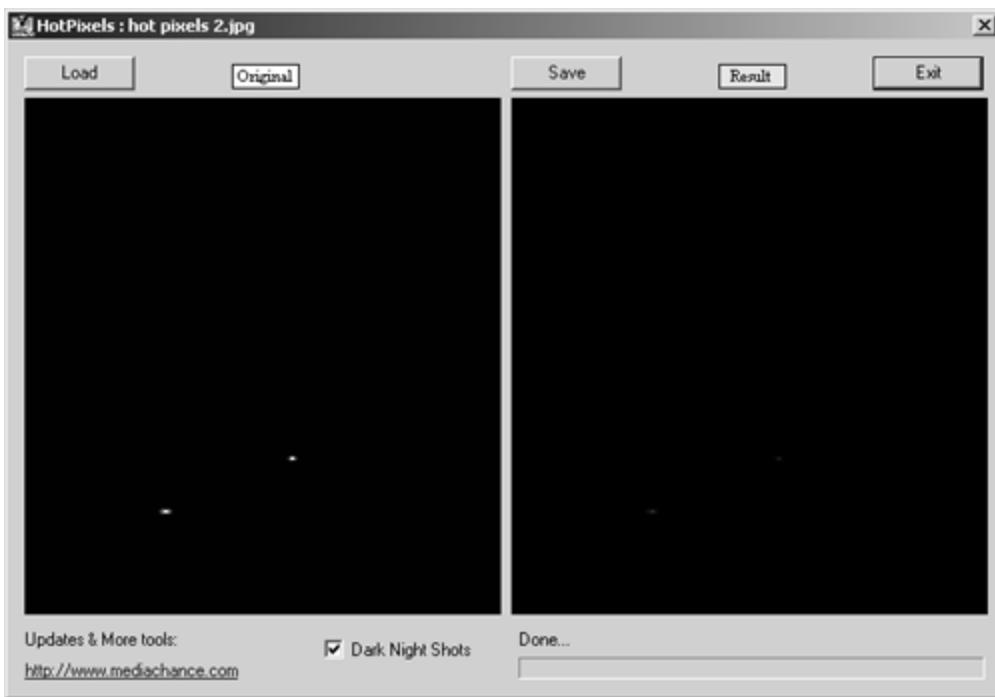
[Figure 2.9](#) shows the locations of the detected hot pixels of my camera. One can reduce the hot pixel effect by averaging, not in the CCD chip itself, of course, but in the digital image. Some companies offer freeware tools for this procedure, for example, MediaChance (<http://www.mediachance.com/digicam/hotpixels.htm>).

Figure 2.9. Hot Pixels of a 2.1 Megapixel CCD Camera



[Figure 2.10](#) shows the processing window of the MediaChance Hot Pixel Eliminator with my hot pixel image loaded and processed.

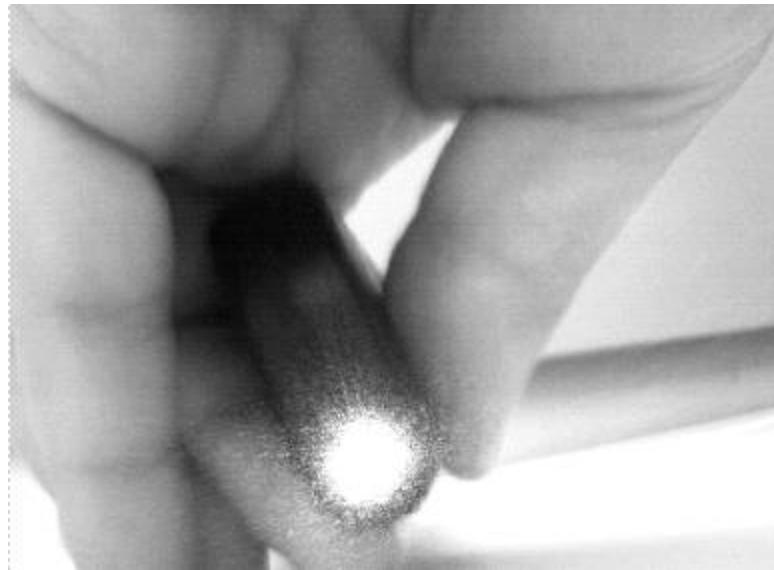
Figure 2.10. MediaChance Hot Pixel Eliminator



Blooming

The *blooming* effect occurs especially in simple CCD image sensors and is caused by an overflow of electrons in the neighboring CCD cells. To the viewer, the result is an enlargement of the very light areas of the image. [Figure 2.11](#) shows this effect by lighting the CCD chip with a laser pointer. The area of the light output is not as sharp as it should be, but the area (especially in the top) around the light output is frayed and shows enlarged light areas.

Figure 2.11. Blooming Effect Caused by a Laser Pointer

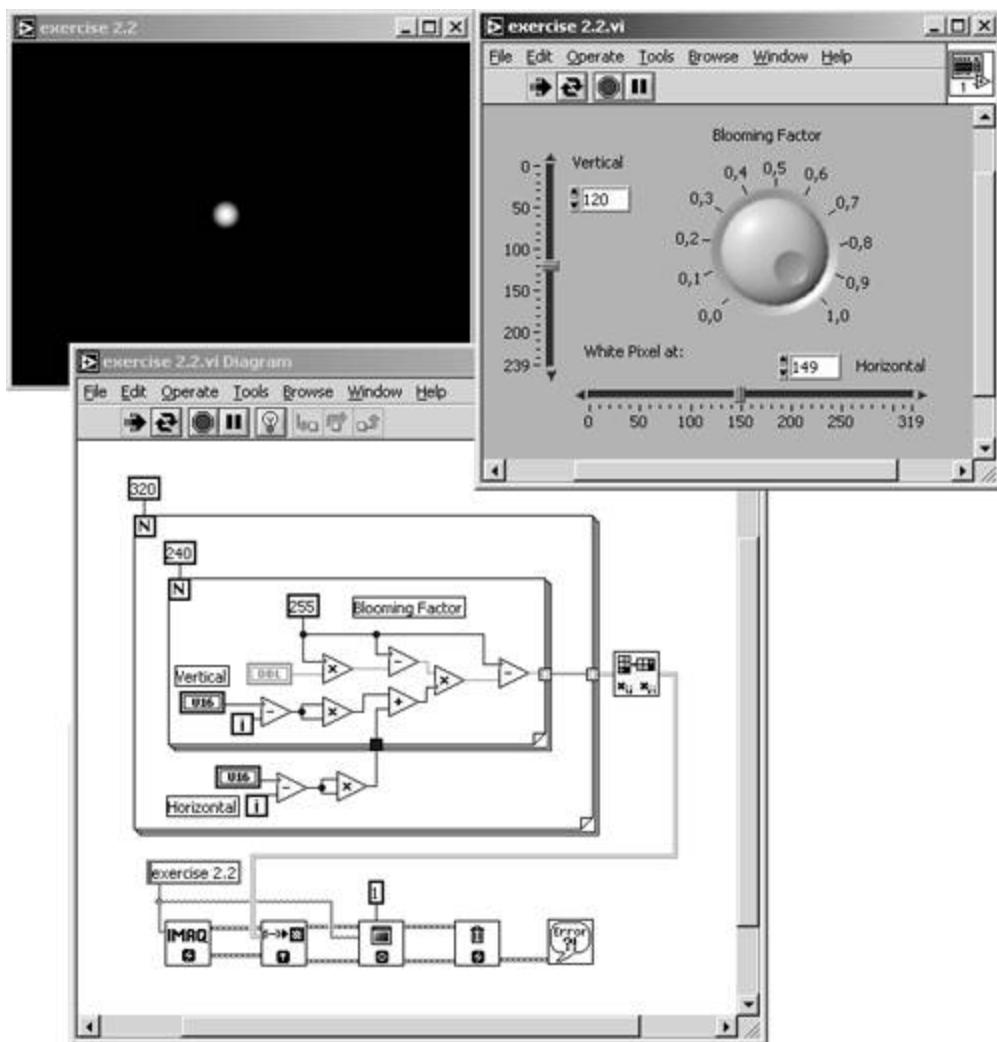


Modern CCDs have a number of mechanisms and techniques that should inhibit blooming:

- *Antiblooming gates* are located horizontally or vertically beside the CCD cells. If they are carrying the respective voltage, the electrons cannot be transferred in the neighbor cell but are caught by the antiblooming gate.
- *Clocked antiblooming* uses the natural recombination of electrons with "holes" in the semiconductor. The holes are generated by a specific clock structure and frequency.

We can add an exercise here, although it does not describe the blooming effect on a very technical level; it is just a simple LabVIEW exercise.

Figure 2.12. Blooming Effect (Exercise)



Exercise 2.2: Blooming Effect Simulation.

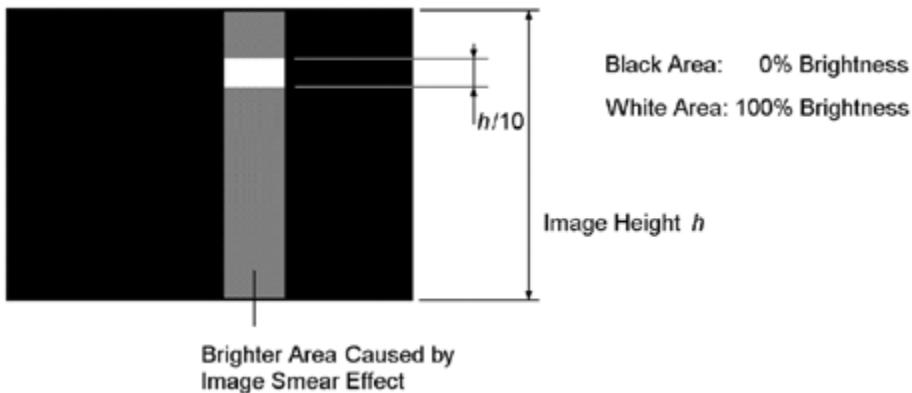
Create a VI that creates a single white pixel in a black image. The pixel can be moved by two numeric controls, one for the horizontal and one for the vertical position. Using another numeric control, you should be able to "influence" the surrounding pixels, depending on their distance from the white pixel (see [Figure 2.12](#)).

Image Smear

Smear is a fault signal that can be seen in the vertical neighborhood of a bright area in a dark image. It is generated by various causes, depending on the CCD technology and structure. Common to all causes is that the CCD cell content is changed during the transfer process.

[Figure 2.13](#) shows a test image that is used for the measurement of the image smear effect. It consists of a white rectangle (100% brightness) of 10% height of the full image inside a totally black image (0% brightness). The resulting stripes above and below the white rectangle are caused by the image smear effect.

Figure 2.13. Test Image for Measuring the Smear Effect



Realization of CCD Devices

Until now we discussed only the theoretical aspects of CCD sensors. We did not talk about the practical realization, though sometimes we assumed that the most common application of CCD sensors are CCD cameras.

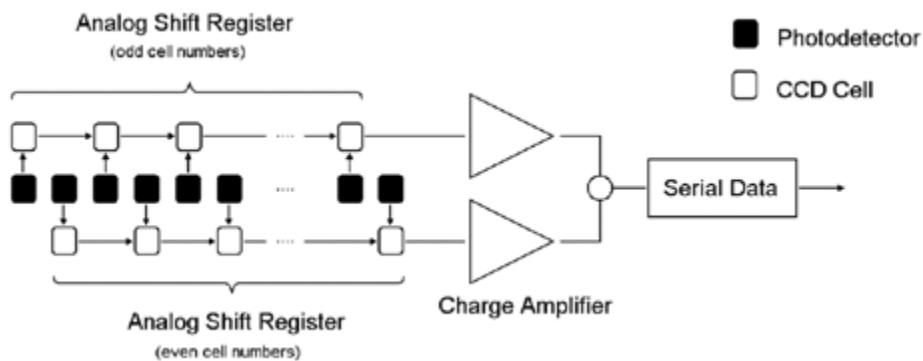
This section deals with some examples for practical realizations, starting with linear CCD sensors for measuring lengths or positions and then discussing the two main CCD structuring techniques: frame transfer configuration and interline configuration.

Linear CCD Sensors

In a *linear CCD sensor*, the CCD cells are arranged in a single line. The sensor can be used for measuring lengths or positions, with imaging the object to be measured by means of a lens on the linear sensor. The measurement itself is quite easy: simply count the sensor elements to an area of significant contrast change.

Usually, linear CCD sensors have two analog shift registers for odd and even pixels (see [Figure 2.14](#)). This leads to two big advantages [14]:

Figure 2.14. Structure of a Linear CCD Sensor



- The CCD clock frequency can be doubled without limiting the CTE.
- The size of the photoelements can only be half of the size of the shift register elements; this doubles the CCD's resolution.

It is also possible to use linear CCD sensors for image acquisition if the object of interest is moved relatively to the sensor. The best example of this technique is a fax machine; we talk

about other applications using line-scan cameras later in this chapter.

Image Sensors

Recall the CCD principle we discussed in the beginning of this chapter, and especially [Figure 2.2](#). This type of sensor, in which the charge is transferred horizontally from the photodetectors to the CCD cells and then vertically in the CCD cells from top to the bottom of the sensor, is called an *interline transfer sensor*.

This type of sensor is used for CCDs that allow short exposure times; [4] the major disadvantage of interline transfer sensors is that the light-sensitive area of each pixel is drastically reduced by the CCD capacitors, antiblooming gates, and so on. This effect is described by the *fill factor*, which specifies the fraction of the light-sensitive area to the total area.

[4] Do not confuse the term "exposure" in combination with a CCD sensor with the exposure mechanism of a mechanical camera. In general, a CCD uses a pixel-reset gate to control the exposure time; the same gate can be used to prevent horizontal blooming.

[Figure 2.15](#) shows a CCD camera chip with interline structure; [Figure 2.16](#) compares this structure with another possibility: the *frame transfer sensor*. Here, the sensor consists of two areas of approximately the same size (see right picture in [Figure 2.16](#)). The area at the top is light sensitive and captures the image information, which is then transferred to the shielded lower area, where the data is read out. Meanwhile, the sensitive area can capture the next image.

Figure 2.15. CCD Sensor Chip with Interline Structure

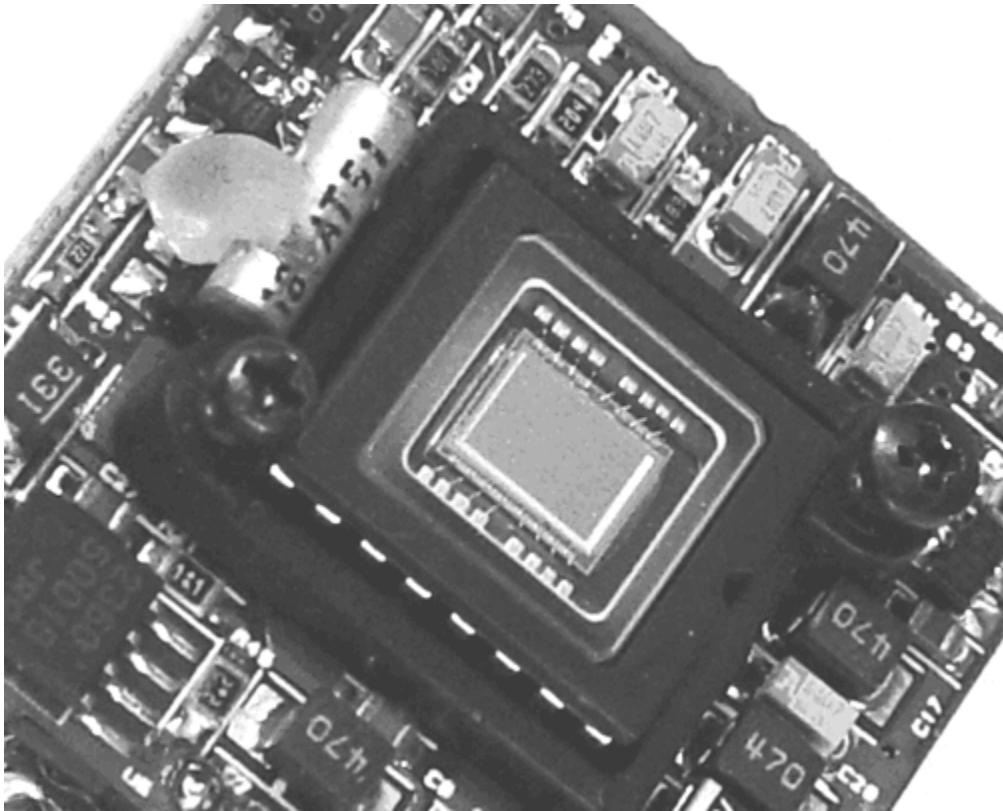
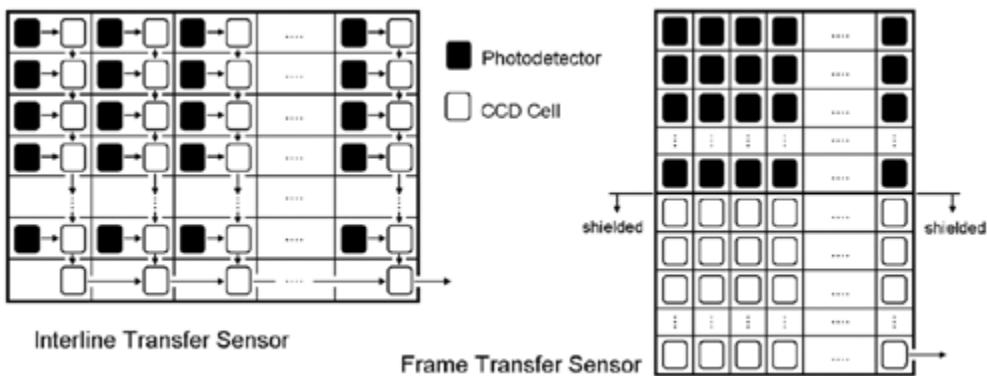
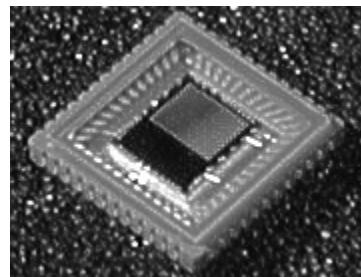


Figure 2.16. Comparison of Interline and Frame Transfer Structures



Because the shift registers of a frame transfer sensor are light sensitive themselves, the fill factor of this sensor is much higher than the fill factor of an interline sensor. [Figure 2.17](#) shows a CCD chip with frame transfer structure.

Figure 2.17. CCD Sensor Chip with Frame Transfer Structure



[\[Team LiB \]](#)

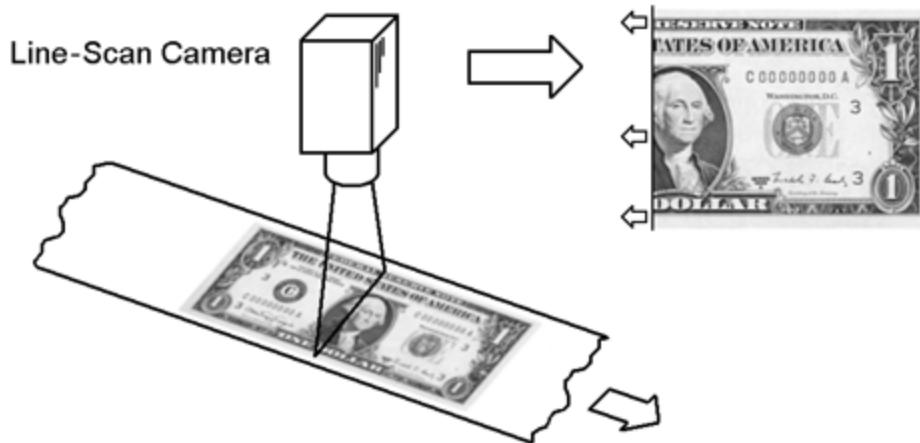
[◀ PREVIOUS](#) [NEXT ▶](#)

Line-Scan Cameras

In [Figure 2.14](#) we showed a one-dimensional CCD array for measuring positions or lengths of objects. In general, we can use the same sensor to acquire two-dimensional images; we only have to scan the second dimension. There are two principal ways to do this:

- The sensor moves relative to the object to be scanned; this principle is used in flat-bed image scanners, whereby two-dimensional images (mostly on paper) can be transferred into digital media.
- The object moves relative to the sensor; at first glance, there seems to be no difference from the first possibility, but some differences do exist:
 - Normally, objects can be moved quickly and easily. If a large number of objects had to be scanned, it would obviously take more time and effort to position the camera rather than to simply position the objects.
 - The object is not limited regarding its length. [Figure 2.18](#) shows an example in which a one-dollar note is being scanned. It is easy to imagine that it would be no problem to add a number of dollar notes to a single image.

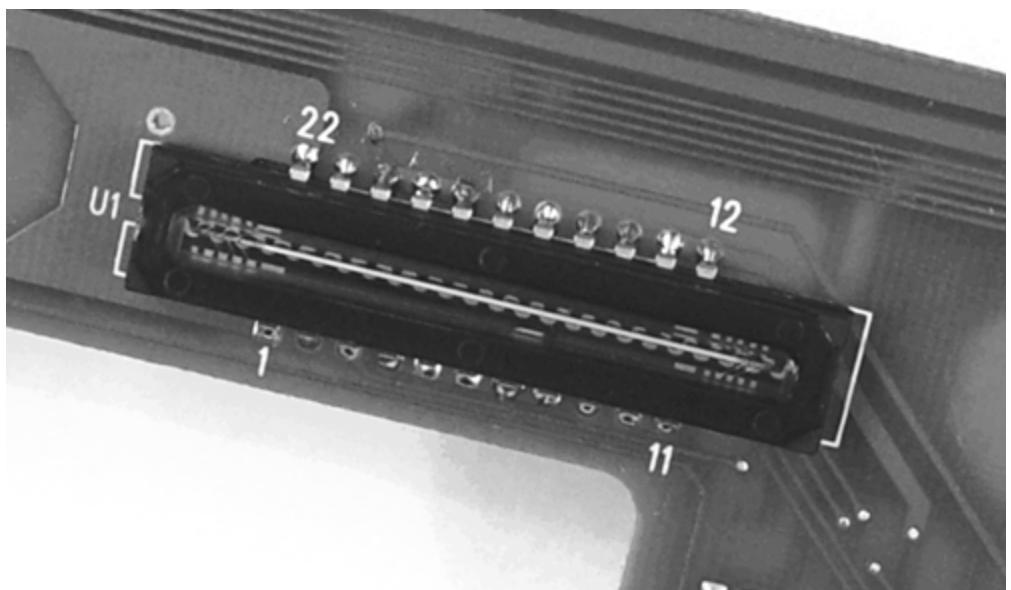
Figure 2.18. Principle of a Line-Scan Camera



With this technology, one can easily generate images with a large number of pixels. For example, the sensor of the line-scan camera in [Figure 2.18](#) has 2048 pixels, which means that it is a high-resolution type. Imagine further that the resolution in both directions should be the same, resulting in a total resolution of about $2048 \times 4800 \approx 9.8$ million pixels—and that with a simple sensor of 2048 CCD cells.

The main application area for line-scan cameras is product inspection; in particular, products transported by a conveyor belt or rotating cylindrical objects are easy to scan. [Figure 2.19](#) shows a line-scan sensor for those or similar applications.

Figure 2.19. Line-Scan Sensor Used in a Flat-Bed Scanner



[\[Team LiB \]](#)

[PREVIOUS](#) [NEXT](#)

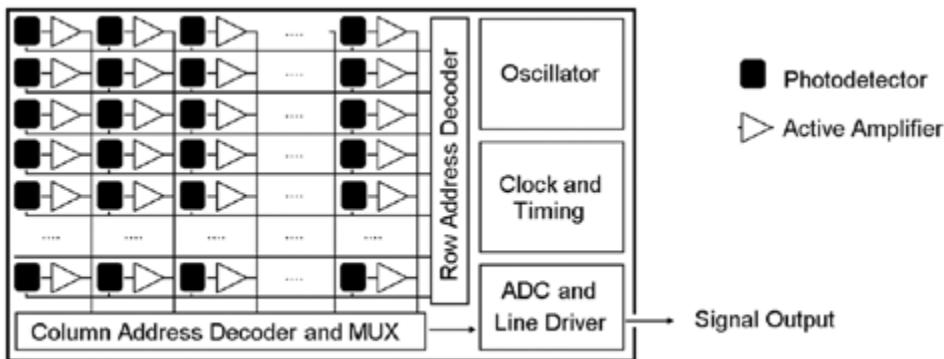
CMOS Image Sensors

Another image sensor technology has joined the market in the past years: image sensors which are structured like a CMOS^[5] memory chip and therefore need row and column address decoders. These sensors make use of a photon-to-electron-to-voltage conversion, taking place in each pixel element.

[5] Complementary metal oxide semiconductor: an MOS technology that makes use of complementary (n- and p-channel) transistor types.

[Figure 2.20](#) shows the principle of such *CMOS image sensors*, which leads to a number of advantages, as well as to some drawbacks. The main advantages are the following:

Figure 2.20. Principle of a CMOS Image Sensor

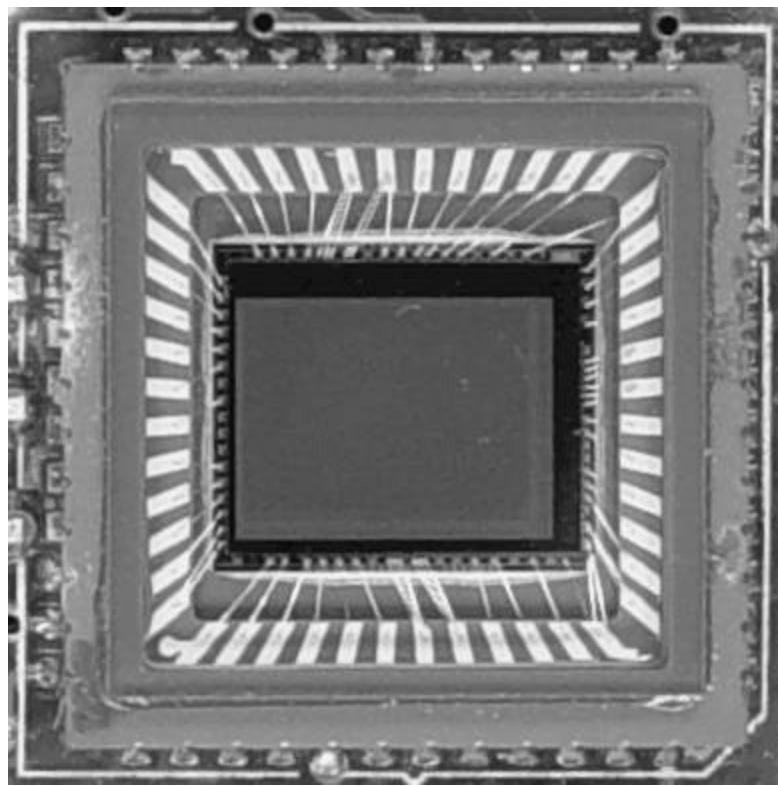


- It is possible to access not only the entire picture, but also a single picture element and a number of pixels, a *region of interest* (ROI), which is also needed in the image processing and analysis sections. An ROI can be defined by addressing the row and column address decoders.
- Some additional system components, like clocking and timing, can be integrated on the same chip with the image sensor. This makes the camera cheaper and more reliable.
- CMOS chips have natural blooming and image smear immunity.

Another main difference is the description of the image speed; CCD sensors are described in terms of *images per second* or *frames per second*, which is not useful with CMOS sensors if you consider the single pixel or the ROI access. Here, we are talking about the *pixel clock*; for example, a pixel clock of 5 MHz will lead to about 65 frames per second if the image frame is 320 x 240 pixels.

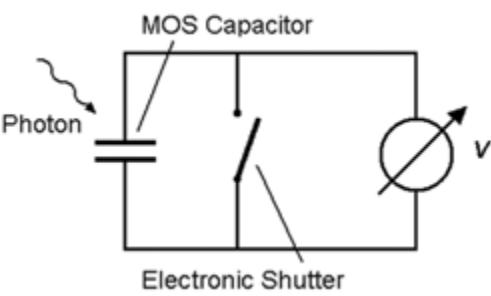
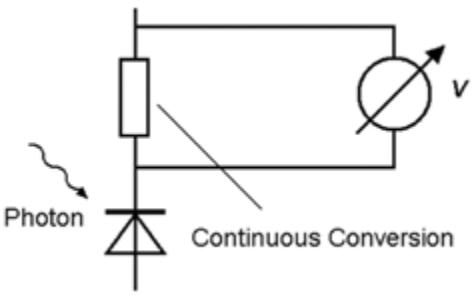
[Table 2.1](#) compares some properties of CMOS image sensors with those of CCDs. The first line (technology) explains how the photocurrent, which is generated by the light radiation on the pixel, is transferred in the related voltage. In CCD sensors, this is done after the electron charge, generated in an MOS capacitor, is transferred to the sensor output. In CMOS sensors, each pixel element has an integrated active amplifier, which generates the voltage and can be accessed by the row and column address decoders. [Figure 2.21](#) shows a CMOS sensor chip; on the right and on the bottom of the sensor the address decoder areas are visible.

Figure 2.21. CMOS (Color) Sensor Chip



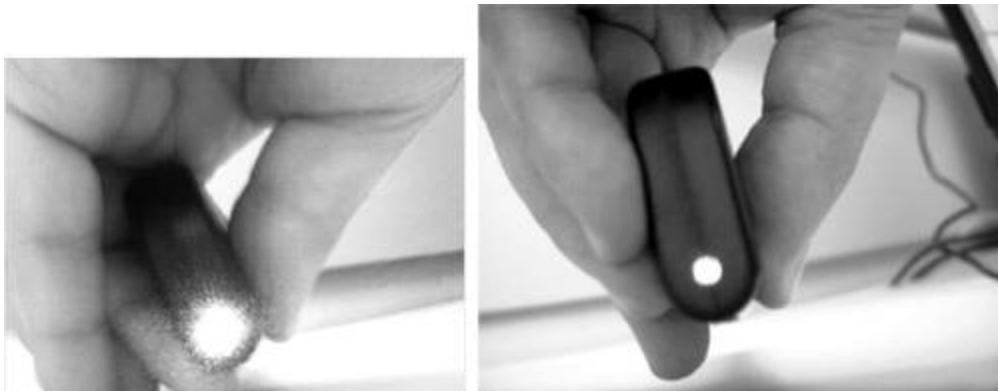
A great disadvantage of CMOS image sensors is their noise generation. The continuous conversion of the photocurrent in the circuit shown in [Table 2.1](#), adds an additional noise source to the signal generation. The main reason is that the CMOS production process is not primarily designed for photoelements, resulting in a poorer sensitivity. Thus, the necessary higher amplification leads to a significantly higher noise level. Moreover, as in CCD sensors, fixed pattern noise (FPN) is another main issue.

Table 2.1. Comparison of CCD and CMOS Technology

	CCD	CMOS
<i>Technology:</i>		
<i>Pixel access:</i>	only full image	single pixel possible
<i>Power:</i>	relatively high	low power consumption
<i>Integration:</i>	only sensor on chip	more functions on chip possible
<i>Conversion:</i>	after charge transfer (full image)	at each pixel
<i>Noise:</i>	FPN, statistical	additional noise sources
<i>Blooming:</i>	antiblooming techniques	natural blooming immunity
<i>Smear:</i>	caused by charge transfer	no image smear possible (Figure 2.23)

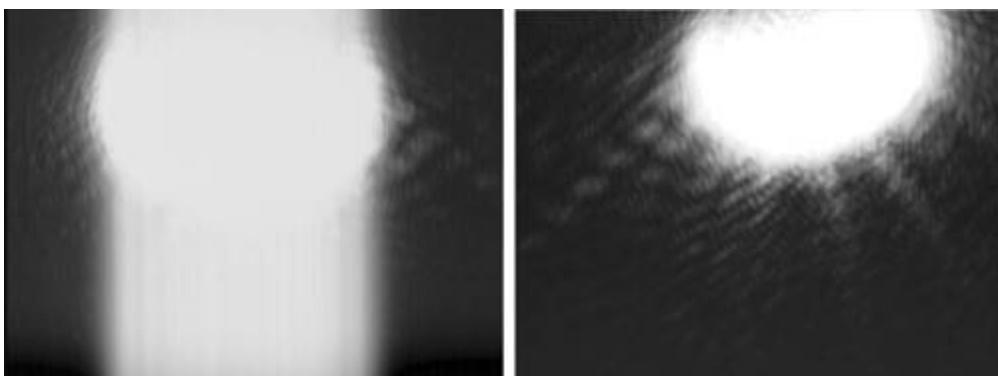
Because of the construction of CMOS image sensors, they show absolute immunity from blooming or image smear. Blooming—the "overflow" of a high amount of electrons in one cell into the neighbor cells—cannot occur in CMOS sensors, because the electron-to-voltage transfer already takes place in the cell itself. [Figure 2.22](#) shows again the blooming effect in a CCD image sensor ([Figure 2.11](#); the sensor itself can be seen in [Figure 2.15](#)), compared to the behavior of a CMOS sensor ([Figure 2.21](#)).

Figure 2.22. Blooming Effect in CCD (left) and CMOS Cameras (right)



[Figure 2.23](#) shows the behavior difference between CCD und CMOS regarding image smear sensitivity. Recall that image smear is the vertical transportation of "errors" during the image readout process. In this example, the sensor area (without lens) is illuminated with a laser pointer, thus flooding the respective sensor areas with photons. The left image, the CCD sensor response, shows a significant smear effect; the area beneath the illumination spot is lighted as well.

Figure 2.23. Smear Effect in CCD (left) and CMOS Cameras (right)



The right image shows the behavior of a CMOS sensor with the same laser pointer illumination. Here, the illumination spot is clearly limited, showing absolutely no image smear effect.

You can find more in-depth information about CMOS image sensors, for example, in [\[17\]](#) and [\[18\]](#).

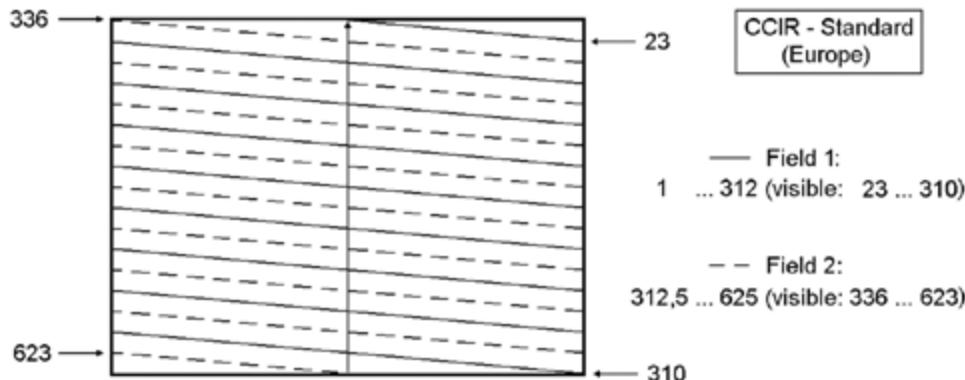
Video Standards

The reading of the data generated by a CCD sensor is influenced by the signal that is used for the transfer of the image content. Mostly, *video signals* are used to display the data on an analog video monitor; this leads to a number of requirements:

- The data must be divided into a standardized number of lines and pixels.
- The display refresh rate must be high enough to prevent flickering of the monitor.
- The timing of the signal should be standardized.

Currently, two standards are established worldwide: the European International Radio Consultative Committee (Comité Consultatif International des Radiocommunications, or CCIR) standard and the U.S. standard RS 170. In both standards, the full video image (called *frame*) is divided into two parts (called *fields*), which are displayed with a half-line offset, in order to double the picture refresh rate. [Figure 2.24](#) shows this principle, which is called *interlacing*, using the European CCIR standard.

Figure 2.24. Video Frames (European CCIR Standard)



Here, each frame consists of 312.5 video lines, but only 287 of them are visible.[\[6\]](#)[Table 2.2](#) shows some data for both the CCIR and the RS 170 standard; you can see that the vertical sampling frequency (which is also the refresh rate of the fields) obviously is derived from the power frequency in the respective geographic areas. A power frequency of 50 Hz (for Europe) leads to a frame refresh rate of 25 Hz, which is sufficient for movies.[\[7\]](#) With the RS 170 standard, the refresh rate is even higher.

[6] Some of these lines are necessary for the flyback of the electron beam; the rest can be used for transmitting additional information.

[7] In comparison, cinema movies usually have a refresh rate of 24 images per second.

Table 2.2. Comparison of CCIR and RS 170 Video Standards

Standard:	CCIR (Europe)	RS 170 (U.S.)
<i>Vertical sampling frequency:</i>	50 Hz	59.9 Hz
<i>Vertical sampling time:</i>	20 ms	16.68 ms
<i>Vertical flyback time:</i>	1.536 ms	1.271 ms
<i>Horizontal sampling frequency:</i>	15.625 kHz	15.734 kHz
<i>Horizontal sampling time:</i>	64 μ s	63.55 μ s
<i>Horizontal flyback time:</i>	11.52 μ s	10.76 μ s
<i>Number of lines:</i>	625	525
<i>Visible lines:</i>	574	485

[Figures 2.25](#) and [2.26](#) show the video timing diagrams for the CCIR and the RS 170 standards, respectively. A major difference between these two is that the numbering of field 2 in the RS 170 standard starts again at 1, whereas the numbering in CCIR is continuous. [Table 2.3](#) shows the line numbering scheme in both standards.

Figure 2.25. CCIR Standard Timing Diagram

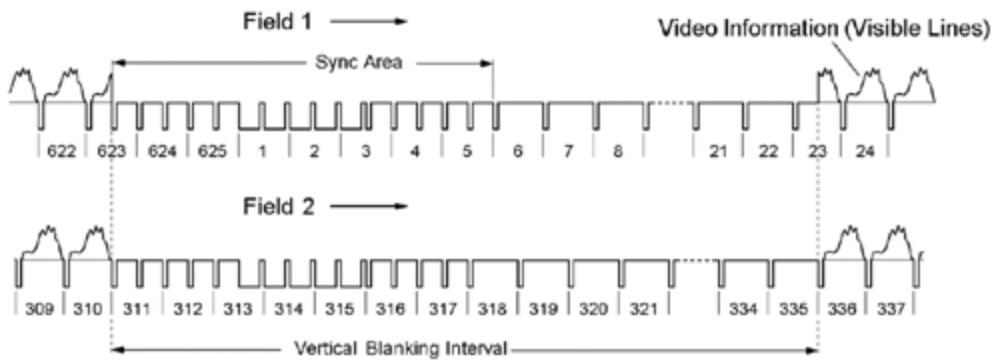


Figure 2.26. RS 170 Standard Timing Diagram

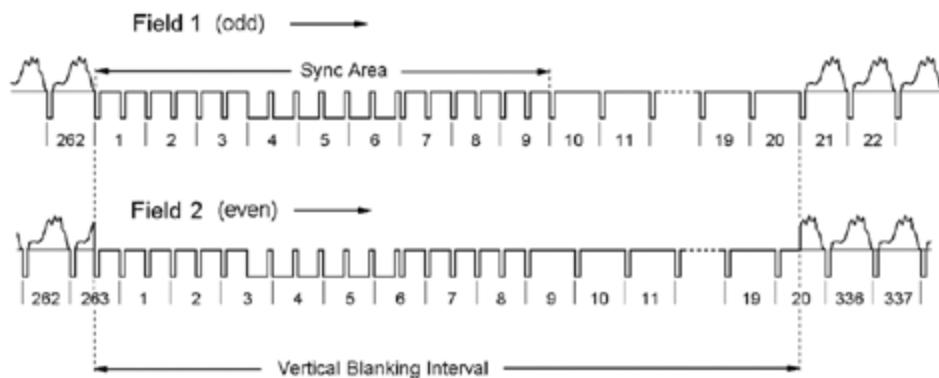


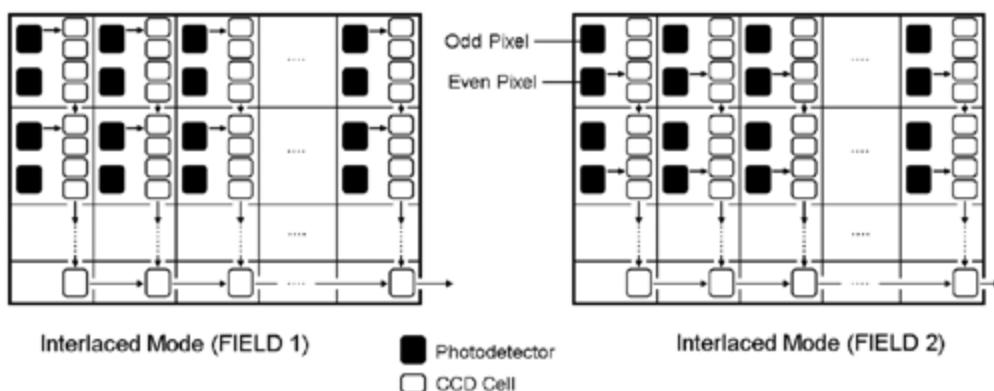
Table 2.3. Line Numbering in CCI R and RS 170 Video Standards

Standard:	CCIR (Europe)	RS 170 (U.S.)	
<i>Line numbering:</i>	Field 1: Field 2:	1...312 312.5...625	Field 1 (odd): Field 2 (even):
<i>Visible lines:</i>	Field 1: Field 2:	23...310 336...623	Field 1 (odd): Field 2 (even):
			1...262 1...263 23...262 20.5...263

Considering these techniques, especially interlacing, it is evident that the linear reading of the contents of a CCD sensor chip leads to complications. It would make more sense to read every second pixel for the generation of field 1, and afterwards the rest of the pixels for field 2.

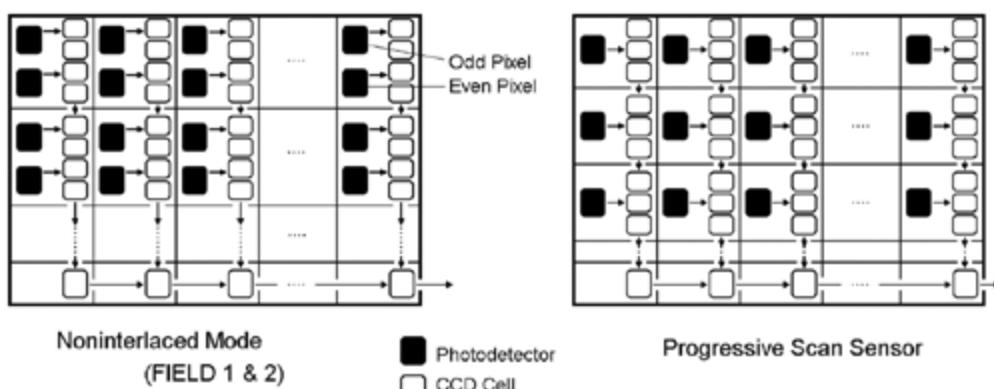
Actually, CCD sensors for specific video standards may include this functionality for the generation of images consisting of two fields. Here, a sensor element consists of two photoelements, one for the odd and one for the even pixel, and four CCD cells (see also [Figure 2.27](#)). Generating field 1, the odd pixels are read and transferred to the sensor output; field 2 uses the even pixels.

Figure 2.27. Interlaced Mode of a CCD Video Sensor



The same sensor can be used for *noninterlaced* mode, in which both pixel sensors are read, both fields are displayed in the same area, and the refresh rate is doubled; [Figure 2.28](#) shows this principle.

Figure 2.28. Noninterlaced Mode and Progressive Scan Sensor

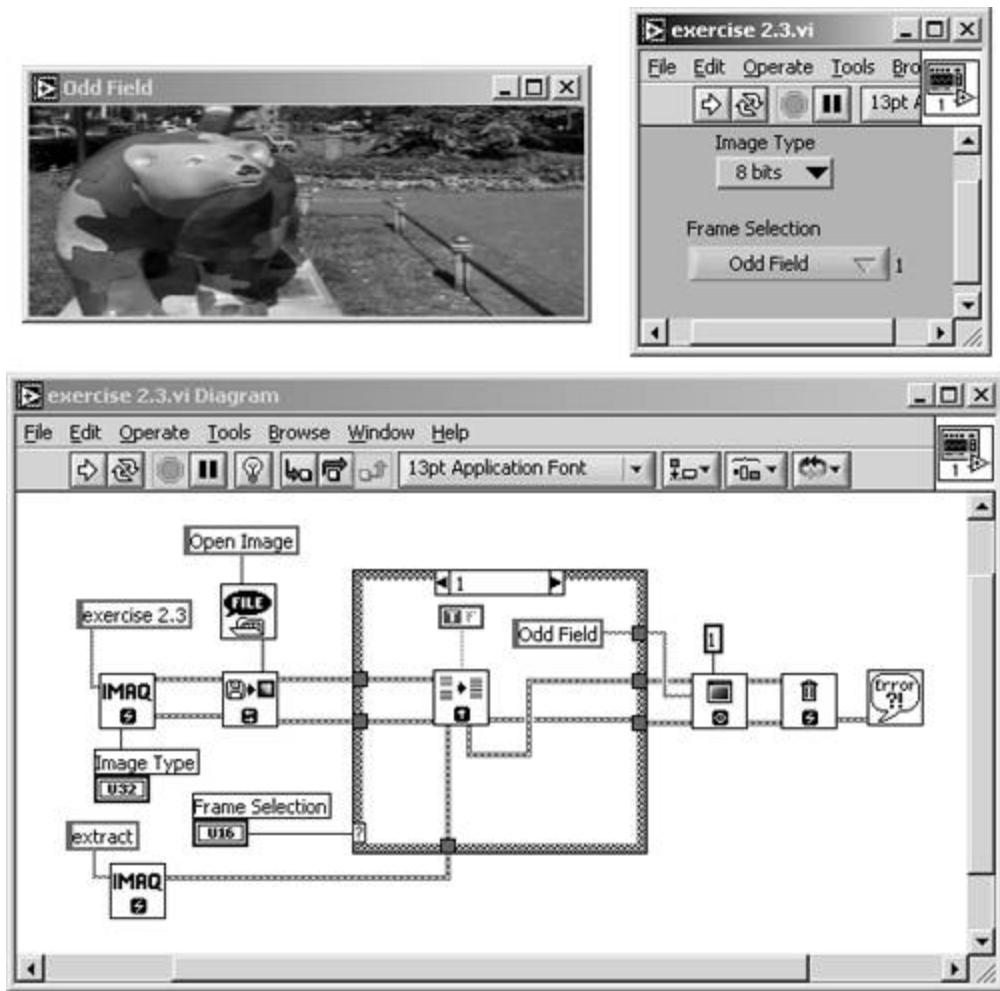


Exercise 2.3: Deinterlacing Images:

IMAQ Vision contains a function that performs functions such as interlacing and deinterlacing: [IMAQ Interlace](#). With this SubVI, it is possible either to extract the odd or even field areas from an image or to build an image, using the odd and even fields of previously acquired image data. [Figure 2.29](#) shows an example exercise.

"Odd field" and "even field" mean that the odd field contains all of the odd lines and the even field contains all of the even lines of an image, as described in [Table 2.3](#) and [Figure 2.27](#). If the captured image contains fast moving objects, the even and the odd field might differ; in some cases it makes sense to remove one of them and duplicate or interpolate the missing lines.

Figure 2.29. Image Reduction with IMAQ's Interlace Function



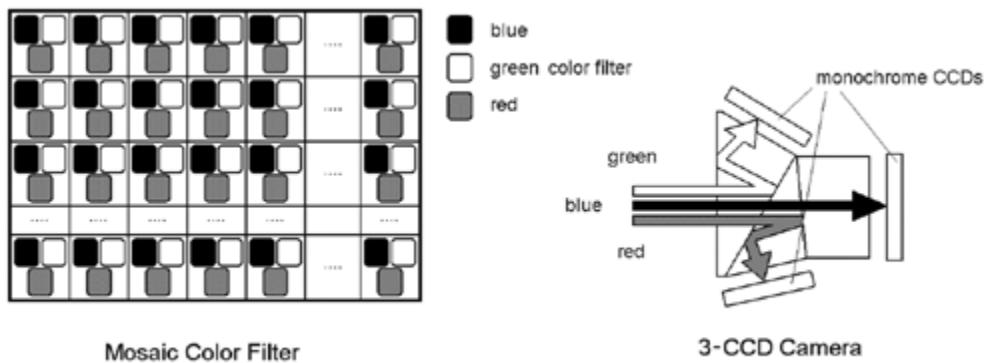
Finally, newer cameras, which generally allow for higher picture refresh rates, use *progressive scan sensors*, where the full image resolution is used. Typical resolutions are, for example, 640 x 480 or 1280 x 1024. As shown in [Figure 2.28](#), the sensor consists only of a single photoelement and three CCD cells per sensor element, which makes a higher integration of the sensor possible.

Color Images

Because photoelements can gather only brightness information of a pixel, special techniques are necessary to obtain color images. As mentioned above, a color image can be described by a multiplane image, in which each plane represents a single color, for example, red, blue, and green. On the camera side, two principles for obtaining color images are possible:

- If the CCD sensor uses a high number of color filters, as shown in [Figure 2.30](#)(left side), the color planes can be generated sequentially. This leads to a very complex sensor as well as to special timing requirements of the reading sequence.

Figure 2.30. Possibilities for Color CCD Sensors



- In high end cameras, the light beam is split into three components: a red, a blue, and a green beam, which affect monochrome CCD sensors (right side of [Figure 2.30](#)). The three color planes are generated simultaneously and merged to a color image by the camera electronics.

Color Models

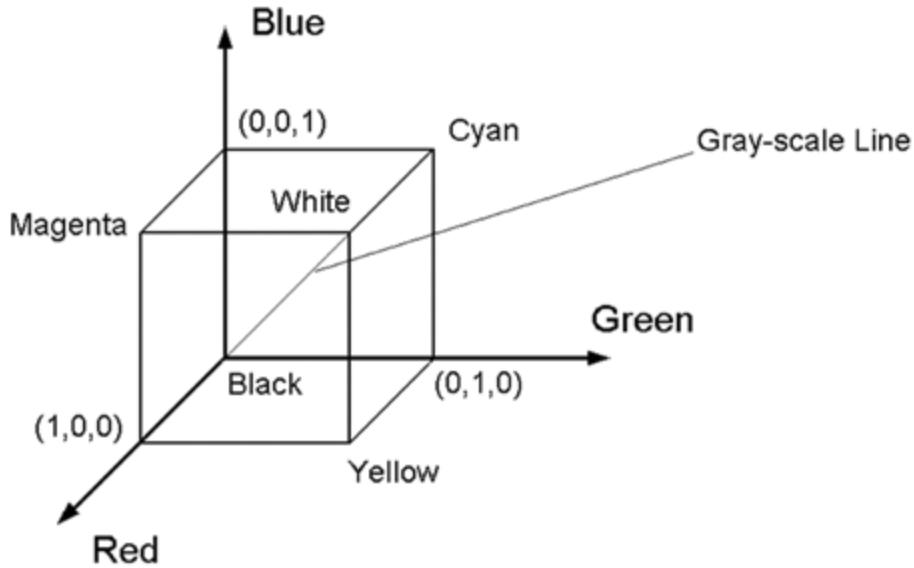
Because the planes of a color image are split into red, blue, and green, it is natural to define these colors as the basic elements of a color image. You will find out later that this specific *color model* leads to some disadvantages, especially if a computer has to distinguish between colors.

By the way, humans perceive color differently, and sometimes it is necessary to use color models, which make use of other properties like *intensity*, *saturation*, *hue*, or *luminance*. We first describe the RGB color model and then discuss these other description techniques.

The RGB Color Model

[Figure 2.31](#) shows the RGB color space, using a cube created by three axes representing pure red, green, and blue color. A main property of this color space is that the sum of all three basic colors, using maximum intensity, is white. Gray-scale values follow the line from black (the origin of the coordinate system) to white.

Figure 2.31. RGB Color Cube



Full intensity of a single color is defined with the value 1. Therefore, if a value of $(1, 1, 1)$ should be white, the RGB color model is obviously based on additive color mixing.

If a color image has to be converted into a gray-scale image, the following equations can be used. One possibility is the simple average of the color values, using R, G, B for the color values and GS for the gray-scale value:

Equation 2.6

$$GS = 0.333 \cdot R + 0.333 \cdot G + 0.333 \cdot B .$$

Another equation, which takes into account the luminance perception of the human eye, is

Equation 2.7

$$GS = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B .$$

The CMY (CMYK) Color Model

Also shown in [Figure 2.31](#) are the basic colors for a subtractive model: cyan, magenta, and yellow, which are the complements of the colors red, blue, and green. It is easy to go from RGB to CMY with these simple equations:

Equation 2.8

$$C = 1.0 - R$$

$$M = 1.0 - G$$

$$Y = 1.0 - B$$

or

Equation 2.9

$$\begin{pmatrix} C \\ M \\ Y \end{pmatrix} = \begin{pmatrix} 1.0 \\ 1.0 \\ 1.0 \end{pmatrix} - \begin{pmatrix} R \\ G \\ B \end{pmatrix} ;$$

and back to RGB with:

Equation 2.10

$$R = 1.0 - C$$

$$G = 1.0 - M$$

$$B = 1.0 - Y$$

or

Equation 2.11

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1.0 \\ 1.0 \\ 1.0 \end{pmatrix} - \begin{pmatrix} C \\ M \\ Y \end{pmatrix} .$$

If a color image has to be printed, black (K) as a fourth color is added to the model to achieve a purer black than the simple combination of the other three colors, resulting in the CMYK model. Transformation from CMY to CMYK is done by

Equation 2.12

$$K = \min(C_{\text{CMY}}, M_{\text{CMY}}, Y_{\text{CMY}})$$

$$C_{\text{CMYK}} = C_{\text{CMY}} - K$$

$$M_{\text{CMYK}} = M_{\text{CMY}} - K$$

$$Y_{\text{CMYK}} = Y_{\text{CMY}} - K ;$$

and from CMYK to CMY

Equation 2.13

$$C_{\text{CMY}} = C_{\text{CMYK}} + K$$

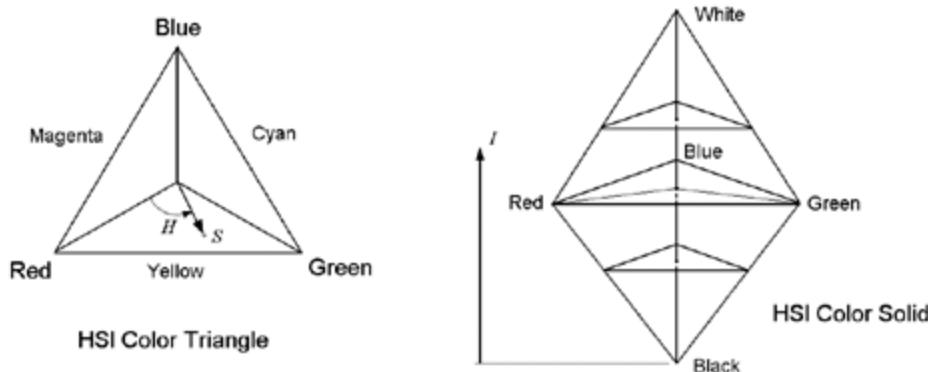
$$M_{\text{CMY}} = M_{\text{CMYK}} + K$$

$$Y_{\text{CMY}} = Y_{\text{CMYK}} + K .$$

The HSI Color Model

As mentioned above, it might be useful to use a different color system, based on more natural properties like hue, saturation, and color intensity. [Figure 2.32](#) shows that this color space can be represented by a solid shown in the right side of the figure. Any color point on the solid surface represents a fully saturated color.

Figure 2.32. HSI Color Triangle and Color Solid



The color hue is defined as the angle starting from the red axis; intensity is represented by the distance from the black point. The following formulas can be used to convert values from RGB to HSI:

Equation 2.14

$$I = \frac{1}{3} (R + G + B)$$

Equation 2.15

$$S = 1 - \frac{3}{R + G + B} [\min(R, G, B)]$$

Equation 2.16

$$H = \cos^{-1} \left[\frac{\frac{1}{2} [(R - G) + (R - B)]}{\sqrt{(R - G)^2 + (R - B)(G - B)}} \right]$$

LabVIEW often uses the *luminance* value instead of intensity, leading to an HSL model. The luminance is calculated by

Equation 2.17

$$L = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B ,$$

which corresponds to Eq. (2.7) and replaces Eq. (2.14).

Exercise 2.4: Color Space Transformation.

These fundamentals are perfect for the creation of a color space transformation calculator, shown in [Figure 2.33](#). You can adjust the sliders according to the red, green, and blue values of the desired color. The frame around these sliders is a color box that changes its color according to the RGB components.

The first (easy) task is to calculate the CMY or the CMYK values, respectively. The HSI/HSL values are more difficult, according to Eq. (2.14) to (2.17). Note that the unsigned 8-bit values have to be converted into 16-bit values if results that may exceed the value 255 are calculated (see [Figure 2.34](#)). I left the calculation of the hue value for your training.

Figure 2.33. Front Panel of the VI Created in [Exercise 2.4](#)

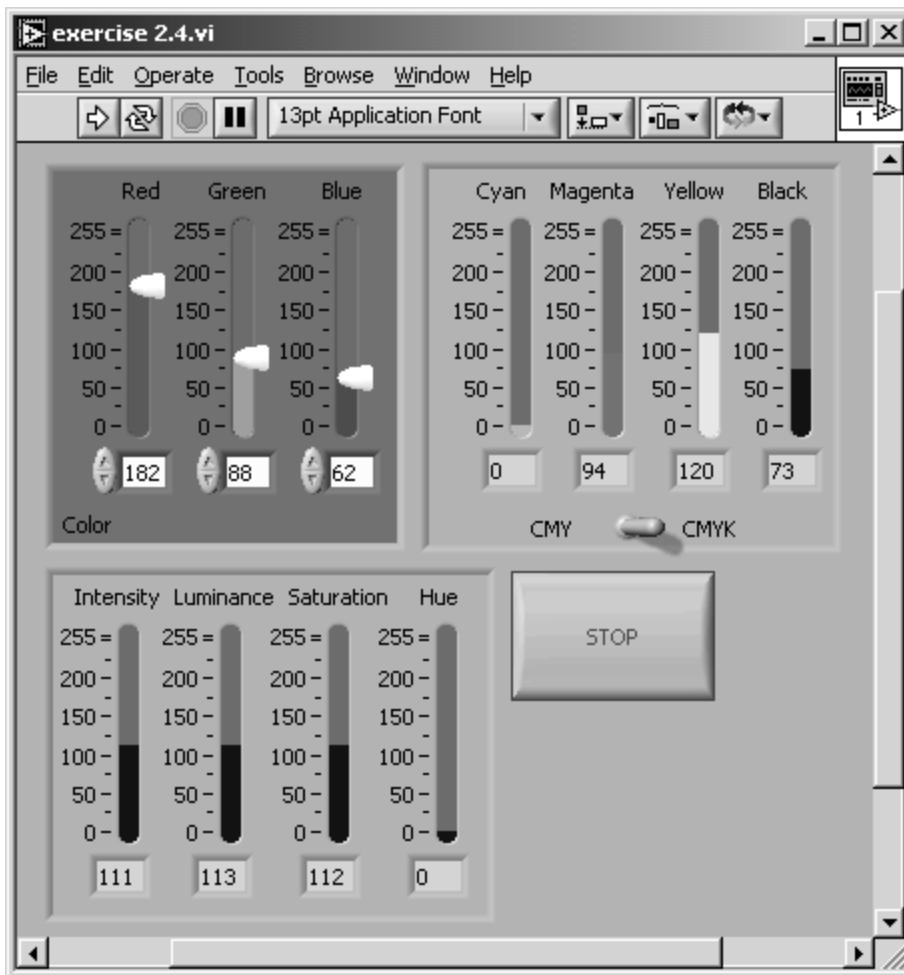
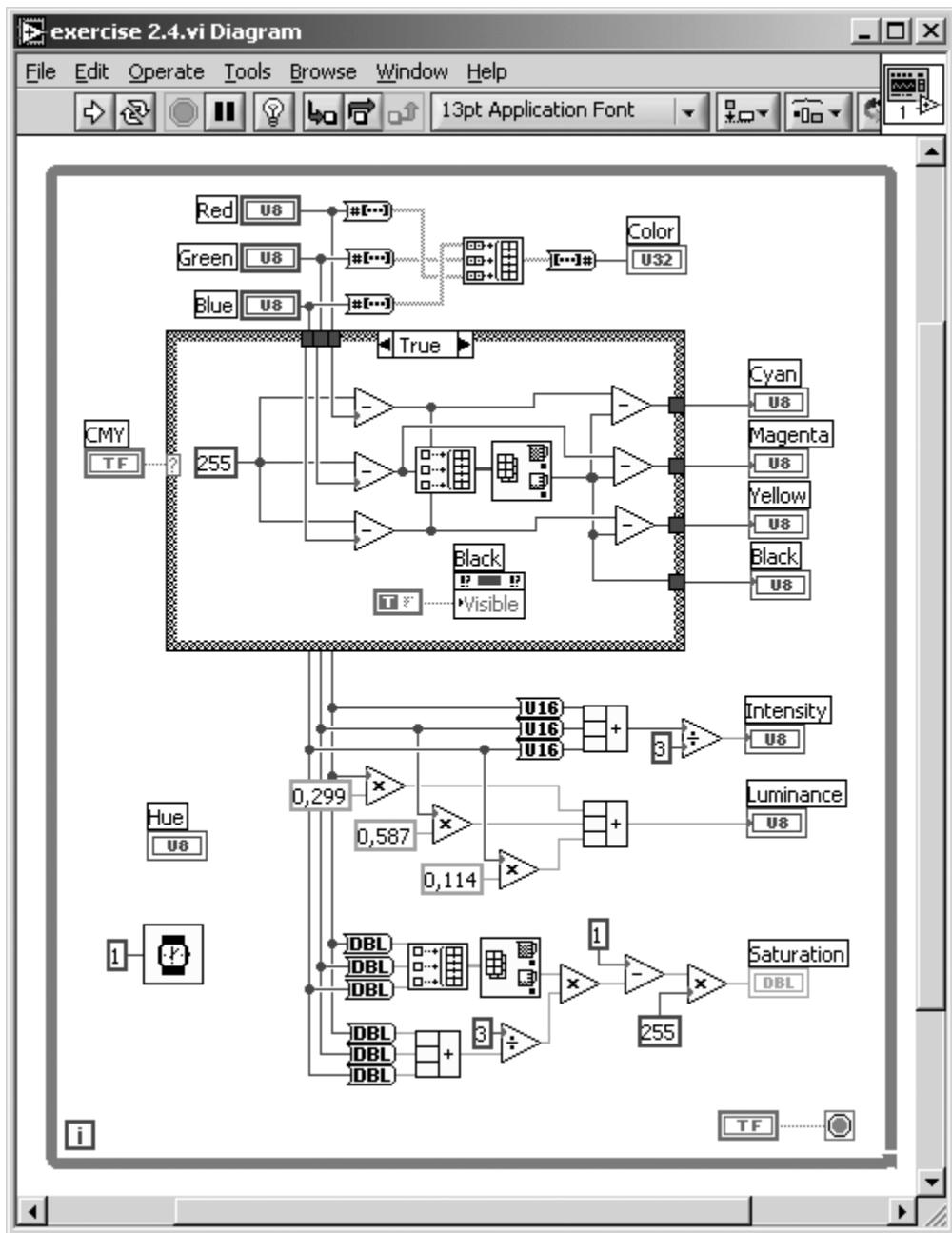


Figure 2.34. Diagram of the VI Created in [Exercise 2.4](#)



The YIQ Color Model

This model is used in commercial video broadcasting. Its main advantage is that the Y component contains all information of the monochrome video signal. This is important because monochrome TV receivers should be able to display suitable data, also from a color video signal. The conversion from RGB to YIQ is done with this equation:

Equation 2.18

$$\begin{pmatrix} Y \\ I \\ Q \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{pmatrix} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix} ;$$

and the conversion from YIQ to RGB:

Equation 2.19

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1.0 & 0.956 & 0.621 \\ 1.0 & -0.272 & -0.647 \\ 1.0 & -1.105 & 1.702 \end{pmatrix} \cdot \begin{pmatrix} Y \\ I \\ Q \end{pmatrix} \quad .$$

Moreover, this model takes advantage of the human sensitivity to luminance changes, which is much higher than to changes in hue or saturation.

Color Video Standards

As mentioned above, YIQ is a color model used by video standards. In general, the following types of color video signals are possible:

- *Component video* offers the best quality because each basic signal (e.g., Y, I, and Q) is sent as a separate video signal, which behavior of course requires more bandwidth.
- *Composite video*: Here, color and luminance signals are mixed into a single carrier wave C , for example, according to

Equation 2.20

$$C = Y + I \cos(\omega_C t) + Q \sin(\omega_C t) \quad ,$$

where ω_C is the frequency of the color subcarrier (a typical composite video signal is FBAS).

- *S-video* (or separated video) uses two channels, one for luminance and another for a composite color signal (another name for this signal is Y/C).

YIQ, for example, is used in the U.S. NTSC video standard. The European PAL standard uses a similar model called YUV, where U and V are simply generated by

Equation 2.21

$$U = R - Y \quad , \quad V = B - Y \quad .$$

Sometimes, U is also called C_r (redness) and V is called C_b (blueness), forming the model YC_bC_r . [Table 2.4](#) gives an overview of the two relevant color video standards, NTSC and PAL.

Table 2.4. Comparison of PAL and NTSC Color Video Standards

Standard:	PAL (Europe)	NTSC (U.S.)
<i>Monochrome video standard:</i>	CCIR	RS 170
<i>Color model:</i>	YUV (YC _b C _r)	YIQ

Color in Digital Video

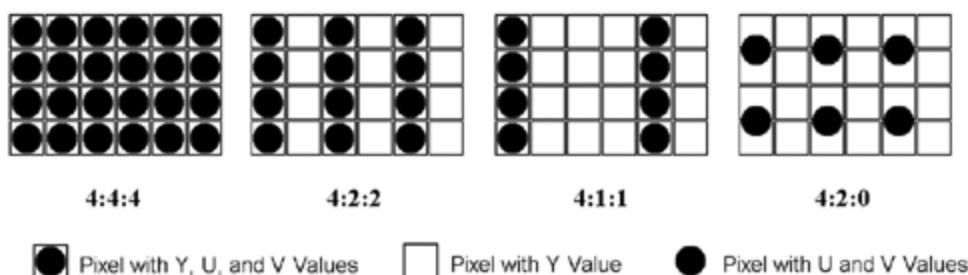
Digital video transfer and recording offer a number of advantages, such as random access, repeated recording, and no need for sync pulses; on the other hand, it requires a lot more bandwidth. Almost all digital video standards use component video and the YUV color model. [Table 2.5](#) shows properties of some standards.

Table 2.5. Standards for Digital Video

Standard:	CCIR 601	CCIR 601	CIF	QCIF
	525/60	625/50		
	NTSC	PAL/SECAM		
<i>Luminance resolution:</i>	720 x 485	720 x 576	352 x 288	176 x 144
<i>Color resolution:</i>	360 x 485	360 x 576	176 x 144	88 x 72
<i>Color subsampling:</i>	4:2:2	4:2:2	4:2:0	4:2:0
<i>Fields/sec:</i>	60	50	30	30
<i>Interlacing:</i>	Yes	Yes	No	No

The line "color subsampling" in [Table 2.5](#) shows some cryptic expressions like 4:2:2; they indicate that the color (or *chrominance*) information can be decimated to reduce the need for bandwidth. For example, 4:4:4 would mean that each pixel will contain the Y (luminance) value as well as the U and V (chrominance) values. Because of the lower sensitivity of the human eye to color, it is in most cases not necessary to sample the color information with the same resolution as the luminance information.

Figure 2.35. Color Subsampling in Digital Video



The expression 4:2:2 means that the chrominance signals U and V are horizontally subsampled by a factor of 2 (see [Figure 2.35](#)). In the same way, 4:1:1 indicates horizontal subsampling by

a factor of 4. The expression 4:2:0 is more difficult; here, subsampling is decimated in both the horizontal and vertical dimensions by a factor of 2. [Figure 2.35](#) illustrates that the color pixel is actually placed between the rows and columns of the brightness pixels.

[[Team LiB](#)]

[!\[\]\(11ebc88d286e0391b6b08e7f66ba7320_img.jpg\) PREVIOUS](#) [!\[\]\(8293201d2bbce9112d410b54020a00d0_img.jpg\) NEXT !\[\]\(fbab1471e9d301c21faf4ccaad34f1d9_img.jpg\)](#)

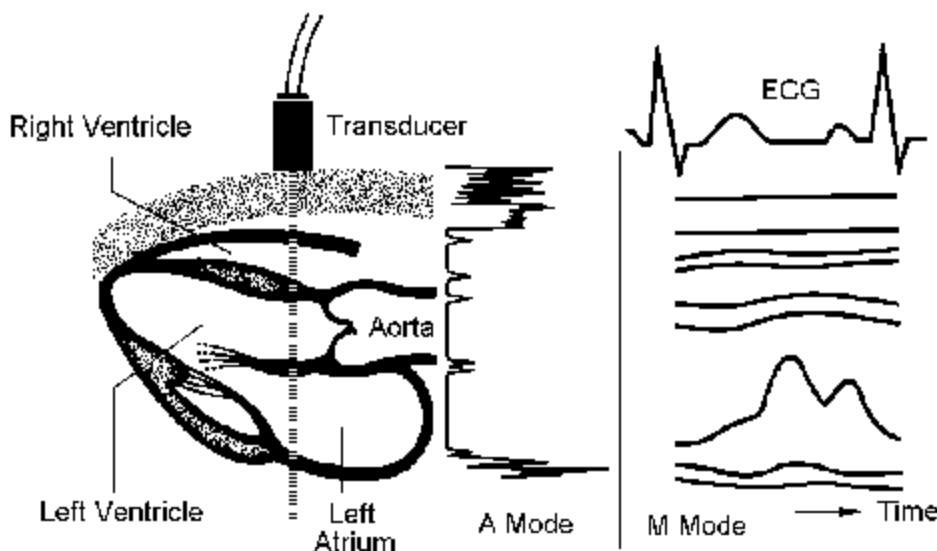
Other Image Sources

This section clarifies that (especially in the life sciences) image sensors or cameras themselves are not the only possibility for the generation of images. I discuss ultrasound imaging devices, computer tomography and its derivatives, and magnetic resonance imaging (MRI).

Ultrasound Imagers

Ultrasound imaging devices are based on the principle of the different sound impedances of different body tissue classes, resulting in reflection and refraction of the sound waves, similar to the respective optical effects. An ultrasound head is used for the emission as well as for the reception of short sound packets, as shown in [Figure 2.36](#).

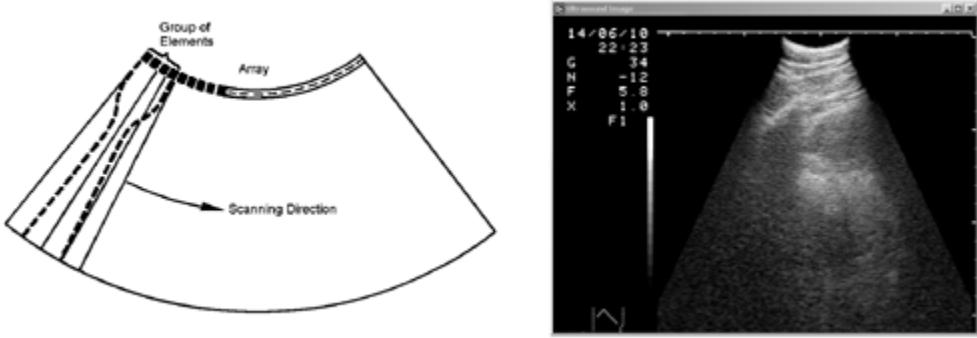
Figure 2.36. Principle of Ultrasound A and M Mode [[19](#)]



The *A mode* simply displays the reflections of the tissue boundaries ([Figure 2.36](#)). This information plotted over time is called *M mode*. [Figure 2.36](#) also shows an M mode display in connection with the electrocardiogram (ECG) [[19](#)].

For ultrasound "images" to be obtained from the area of interest, it is necessary to move the sound head over a defined region (2D ultrasound or *B mode*). Sound heads that perform this movement automatically are called *ultrasound scanners*. Mechanical scanners, used in the early days of ultrasound imaging devices, are nowadays replaced by electronic scanning heads. The example shown in [Figure 2.37](#) uses a curved array of ultrasound elements, which are activated in groups and electronically shifted in the desired direction.

Figure 2.37. Curved Array Ultrasound Head and Corresponding Image



Some more information about sound reflection and refraction: Like the corresponding optical effects, the reflection of sound waves at tissue boundaries with different sound impedance is described by the reflection factor R .

Equation 2.22

$$R = \left(\frac{Z_2 - Z_1}{Z_2 + Z_1} \right)^2 ,$$

with Z_i as the sound impedance of the respective area and the transmission factor T :

Equation 2.23

$$T = \frac{4 \cdot Z_1 \cdot Z_2}{(Z_1 + Z_2)^2} .$$

This is why a special ultrasound transmission gel has to be applied to the sound head before it is attached to the skin; otherwise, almost all of the sound energy would be reflected. [Table 2.6](#) shows examples for the reflection factor R at some special tissue boundaries.

Table 2.6. Typical Values of the US Reflection Factor R

from/to:	Fat	Muscle	Liver	Blood	Bone
Water:	0.047	0.020	0.035	0.007	0.570
Fat:		0.067	0.049	0.047	0.610
Muscle:			0.015	0.020	0.560
Liver:				0.028	0.550
Blood:					0.570

Computed Tomography

The principle of computed tomography (CT) is based on x-ray imaging, which makes use of the characteristic attenuation of x-rays by different tissue types. X-ray imaging generates only a single image, showing a simple projection of the interesting region. CT, on the other hand, can display cross sections of the body, so the imaging device has to "shoot" a large number of attenuation profiles from different positions around the body. An important issue is the

reconstruction of the resulting image.

Looking at the simplest case of a homogeneous object of interest,[\[8\]](#) we see that the intensity / of the detected radiation decreases exponentially with the object width d and is therefore given by

[8] This case also includes radiation of one single frequency.

Equation 2.24

$$I = I_0 \cdot e^{-\mu \cdot d} ,$$

whereas the attenuation P is defined by the product of the attenuation coefficient μ and the object width d :

Equation 2.25

$$P = \ln \frac{I_0}{I} = \mu \cdot d ;$$

finally, the attenuation coefficient itself is therefore

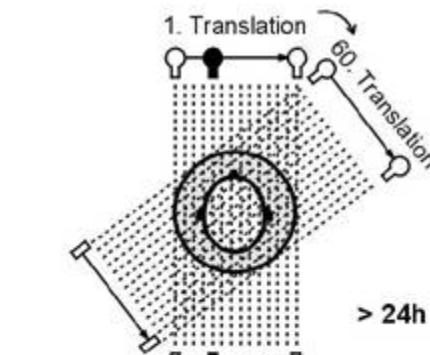
Equation 2.26

$$\mu = \frac{1}{d} \cdot \ln \frac{I_0}{I} .$$

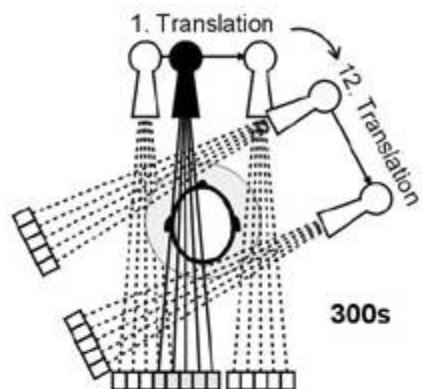
CT Device Generations

Computed tomography devices are usually classified in "generations," as shown in [Figure 2.38](#):

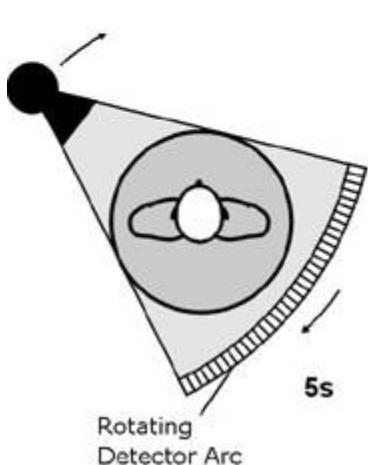
Figure 2.38. CT Device Generations 1 to 4 [\[20\]](#)

Pencil Beam (1970)

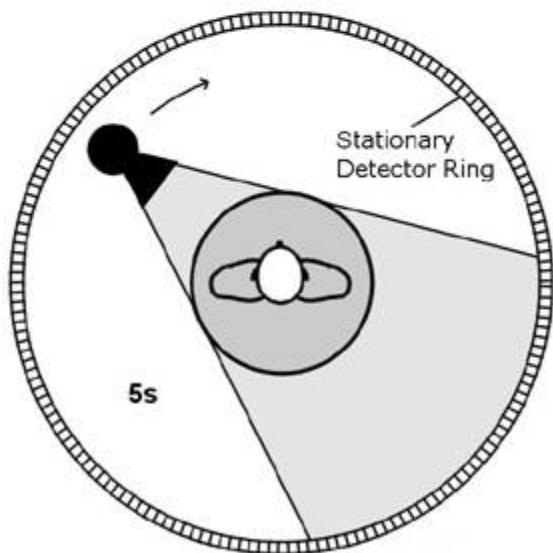
Generation 1: Translation / Rotation

Partial Fan Beam (1972)

Generation 2: Translation / Rotation

Fan Beam (1976)

Generation 3: Continuous Rotation

Fan Beam (1978)

Generation 4: Continuous Rotation

- Scanning by use of a single "pencil beam"; the beam is shifted and rotated (generation 1).
- The needle beam is replaced by a "partial fan beam," which again is shifted and rotated (generation 2).
- A fan beam, which is only rotated in connection with the detector array, is able to scan the entire angle area (generation 3).
- Now only the fan beam rotates, whereas the detector array is fixed and covers the entire area in ring form (generation 4).

CT Image Calculation and Back Projection

As you can imagine, it is not easy to obtain the resulting image (the spatial information), because each beam [9] delivers the total attenuation value of its route; we defined the resulting intensity/in Eq. (2.24) for the simple case of a homogeneous object. In most cases, this will not be true; Eq. (2.24) therefore becomes

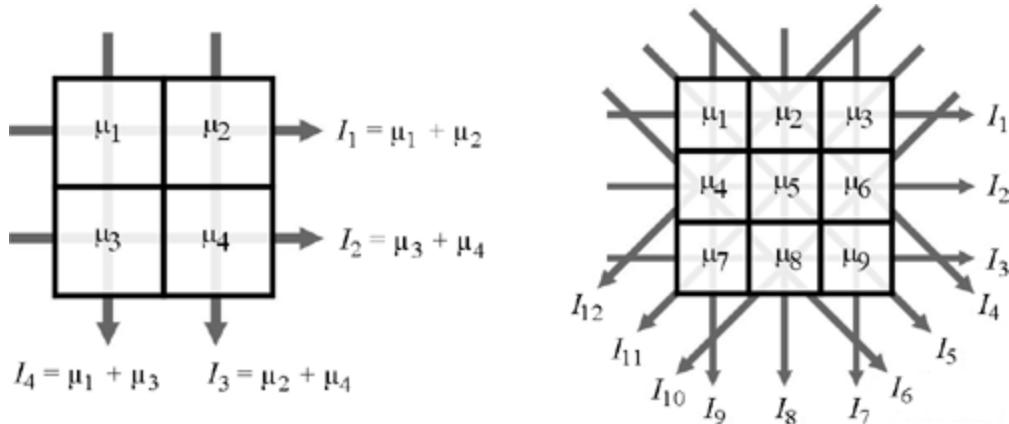
[9] Here, we are considering a CT device of generation 1.

Equation 2.27

$$I = I_0 \cdot e^{-\sum_{i=1}^n \mu_i d_i} ;$$

with μ as the attenuation coefficient in a specific area and d as the respective length of this area, for the intensity I in a certain direction.

Figure 2.39. Simple Calculation of a CT Image [20]



Special algorithms are necessary for computation of the original values μ of a certain area. The following solutions can be used:

1. As shown in [Figure 2.39](#), the simplest method is to solve the equation system consisting of N^2 variables resulting from the intensity values of the different directions. [Figure 2.39](#) shows this principle for 2×2 and 3×3 CT images.
2. It is easy to imagine that method 1 will lead to huge equation arrays and a much too high calculation amount if the pixel number of the image increases. Therefore, the equation systems of existing systems must be solved in an iterative way. An example of this method is described in [Exercise 2.5](#).
3. Today, the method of filtered back projection is usually used. Normal back projection (summation of the projections in each direction) would lead to unfocused images; therefore, the projections are combined with a convolution function to correct the focus on the edges.

A good example, which illustrates the method of back projection, can be found at the National Instruments web page www.ni.com; just go to "Developer Zone," search for "tomography," and download the LabVIEW library [Tomography.lvlib](#) (see [Figure 2.40](#)).

Figure 2.40. Tomography Simulation (National Instruments Example)

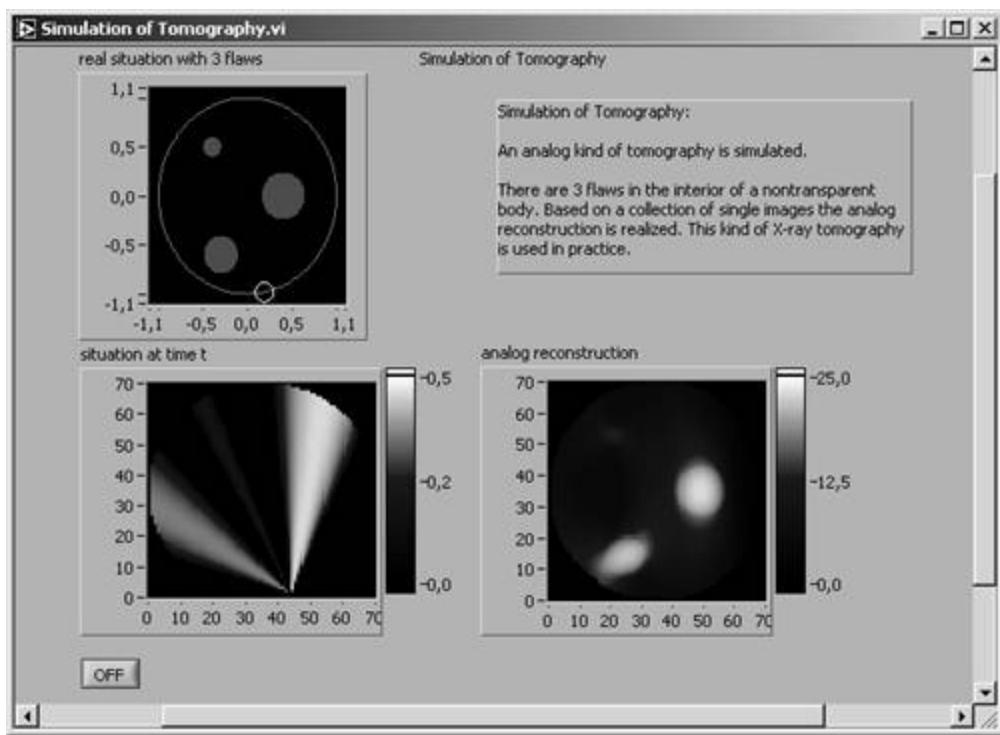
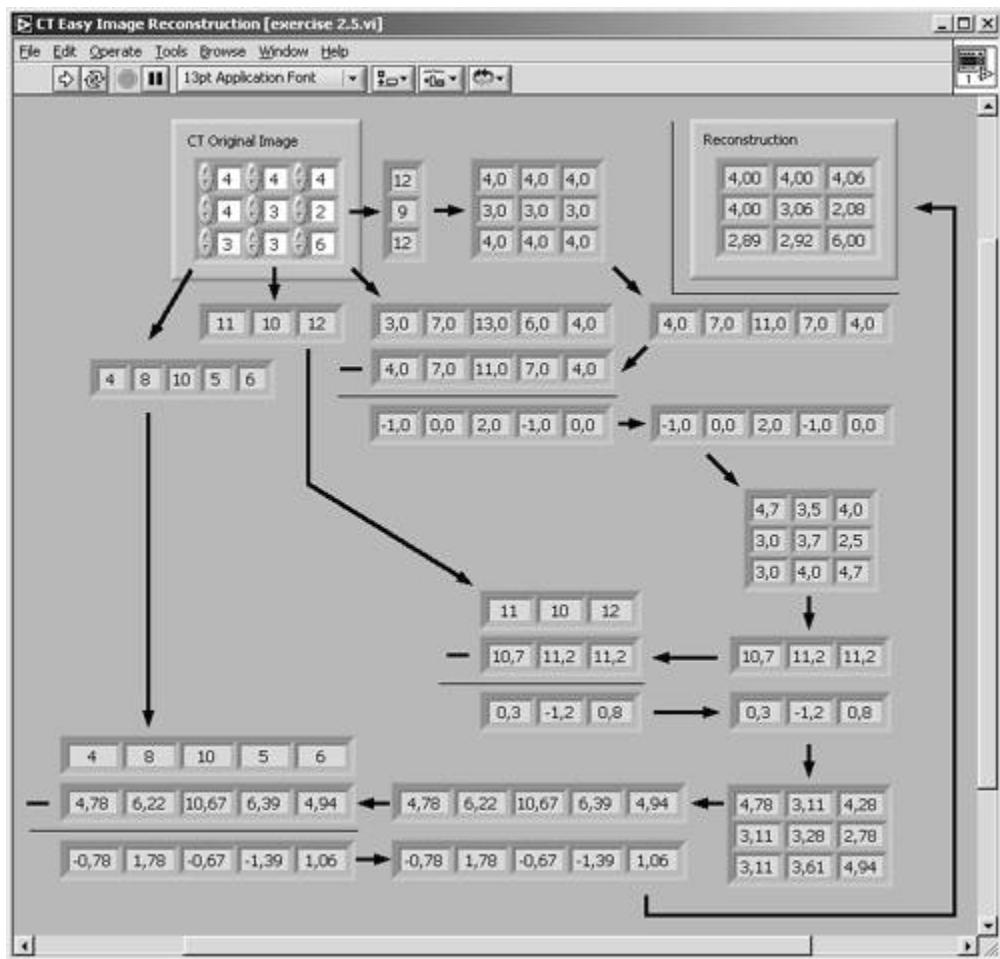


Figure 2.41. Iterative Calculation of a CT Image



Exercise 2.5: Iterative Calculation of a CT Image.

Try to make the iterative calculation (method 2) in a LabVIEW VI (front panel shown in [Figure 2.41](#)). Here, the intensity values of four directions are used, starting with the one to the right. The first estimation is based on the equal distribution of the intensity values over the entire lines. The values of the next direction (bottom right) are calculated and compared to the values of the original image. The difference of these results is applied to the first estimation, resulting in the second estimation, and so on.

As you will find out, the values already are quite close to the original values. You can get the values closer to the original if you run the previously described steps again with the new values. After a third run, the differences will be too small to notice.

Single Photon Emission Computed Tomography (SPECT)

This method is quite similar to computer tomography with respect to image generation. The main difference is that the total emitted gamma radiation, not an attenuation coefficient, of a specific direction is measured. Therefore, the same imaging techniques (usually filtered back projection) can be used.

Positron Emission Tomography (PET)

PET is based on the detection of gamma quanta that result from the collision of an electron with a positron. To make this possible, interesting molecules have to be marked before imaging with a positron radiator; we do not discuss this process here.

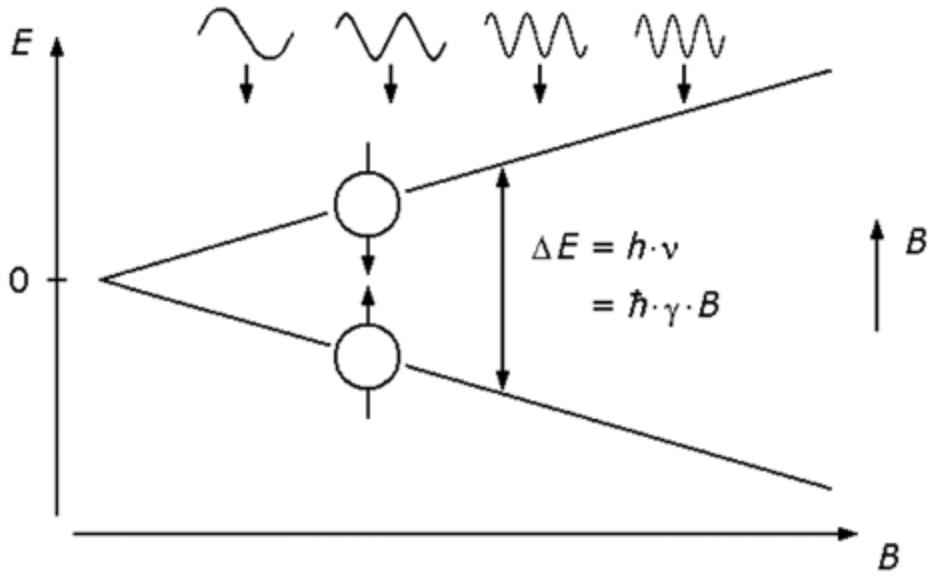
In general, for PET images the same image generation processes that have been described above can be used; here, the main difference is that the machine cannot decide ahead of time which direction is to be measured. However, it is possible to afterwards assign specific events to the appropriate projection directions.

Magnetic Resonance Imaging

Magnetic resonance imaging (MRI; sometimes NMRI for nuclear MRI) uses a special property of particular atoms and especially their nuclei; this property, called *spin*, is related to the number of protons and neutrons. The spin is represented by the symbol $\frac{1}{2}$ and always occurs in multiples of $\frac{1}{2}$. For example, the nucleus of ^1H has a spin of $\frac{1}{2}$ and the nucleus of ^{17}O a spin of $\frac{5}{2}$.

In [Figure 2.42](#), the spin of a nucleus is visualized by an arrow (up or down), which indicates that the nuclei are "rotating" around the arrow axis clockwise or counterclockwise, respectively. If an external magnetic field B is applied, the (usually unoriented) spins may align parallel or anti-parallel to this field.

Figure 2.42. Separation of Energy Levels According to Spin Directions



The energy necessary for the transition from the parallel to the anti-parallel state depends on the value of the external magnetic field B and is given as

Equation 2.28

$$\Delta E = h \cdot \nu = \gamma \cdot B \cdot \frac{h}{2\pi} = \gamma \cdot B \cdot \hbar ,$$

with h as Planck's constant and γ as the intrinsic gyromagnetic ratio. Integrated in Eq. (2.28) is the Larmor equation

Equation 2.29

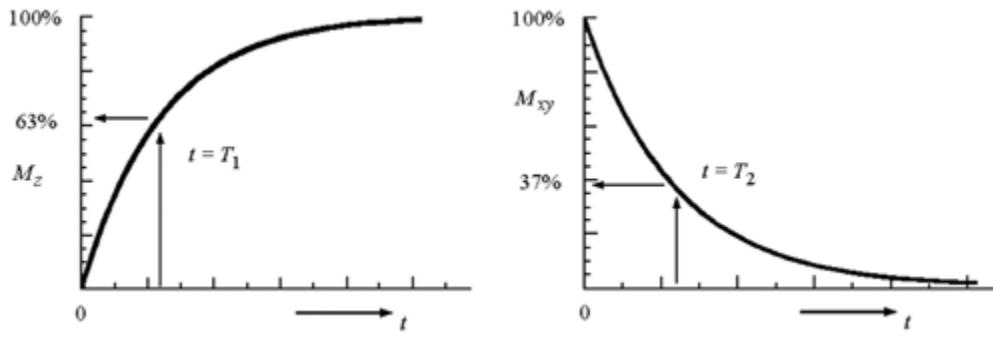
$$\omega = \nu \cdot 2\pi = \gamma \cdot B ,$$

which indicates that the frequency related to the nuclei^[10] depends on the value of the external magnetic field.

[10] This is the so-called precession frequency.

If the energy of Eq. (2.28) is applied to some nuclei, a large number of them are raised from their state of lower energy to the state of higher energy. After a certain time, the nuclei return to their original state, emitting the energy ΔE .

Figure 2.43. Relaxation Times T_1 and T_2



The magnetization M can be split into a component M_{xy} residing in the xy plane and a component M_z in z direction. After excitation of the nuclei with a certain frequency pulse, both magnetization components follow the equations

Equation 2.30

$$M_z = M_0 \left(1 - e^{-t/T_1}\right)$$

and

Equation 2.31

$$M_{xy} = M_0 e^{-t/T_2}$$

(see also [Figure 2.43](#)) with the specific relaxation times T_1 and T_2 . [Table 2.7](#) shows some different values for some important tissue types.

If the Larmor equation ([2.29](#)) can be modified to

Equation 2.32

$$\omega(r_i) = \gamma \left(B_0 + \vec{G} \cdot \vec{r} \right) ,$$

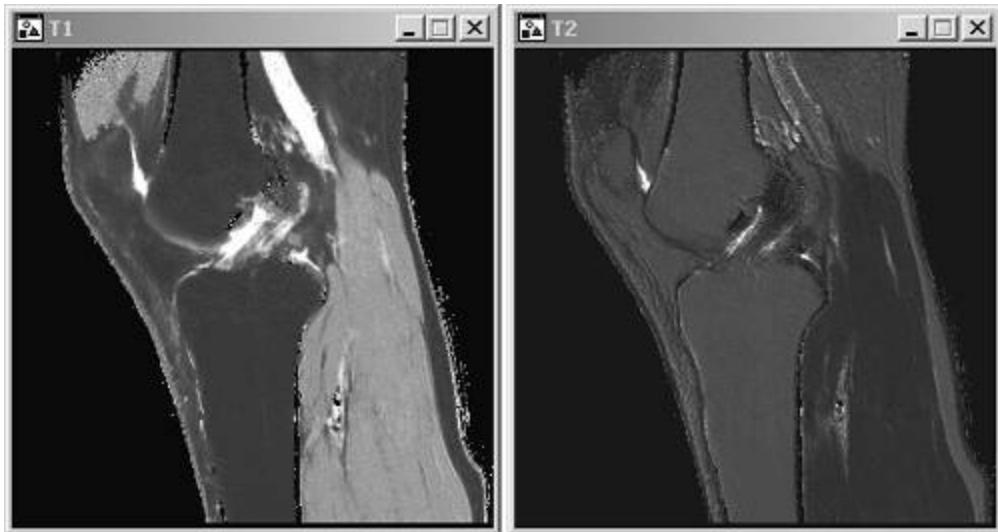
with $\omega(r)$ as the frequency of a nucleus at the location r , and \vec{G} describing the gradient location-dependent amplitude and direction, the location r of the nucleus can be specified. The final imaging techniques are quite similar to those used in computer tomography but offer a number of additional possibilities:

Table 2.7. Typical Values of Relaxation Times T_1 and T_2 at 1 Tesla

Tissue Type	Relaxation Time T_1	Relaxation Time T_2
<i>Muscle:</i>	730 ± 130 ms	47 ± 13 ms
<i>Heart:</i>	750 ± 120 ms	57 ± 16 ms
<i>Liver:</i>	420 ± 90 ms	43 ± 14 ms
<i>Kidney:</i>	590 ± 160 ms	58 ± 24 ms
<i>Spleen:</i>	680 ± 190 ms	62 ± 27 ms
<i>Fat:</i>	240 ± 70 ms	84 ± 36 ms

- It is possible to distinguish more precisely between tissue types, compared with methods, such as CT, that use only the attenuation coefficient. Different tissue types can have identical attenuation coefficients but totally different relaxation times T_1 and T_2 because of their different chemical structure.
- Another possibility is to scale the gray scales of the image according to the respective relaxation times. As can be seen in [Table 2.7](#), T_1 and T_2 do not have a direct relationship. [Figure 2.44](#) shows an example for a T_1 and a T_2 weighted image.

Figure 2.44. MRI Images: Based on T_1 (left) and T_2 (right) of a Knee Joint



[Team LiB]

◀ PREVIOUS ▶ NEXT ▶

Chapter 3. Image Distribution

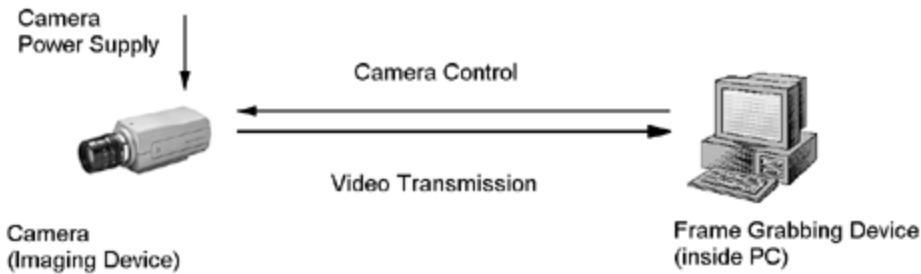
Now that our images are generated, we have to transfer them from the location where they are generated to the location where they can be displayed, processed, and analyzed. In the section *Frame Grabbing* we talk about "the other end of the camera cable"; that means principles regarding capturing camera signals, digital as well as analog. To cover the digital part, we discuss some important details about *bus systems and protocols*. Naturally, it is not possible to cover all of them, so we focus on the more important ones.

The second part of this chapter deals with *compression techniques* in general first, and then with some *image and video standards*, which usually make use of compression. Finally, we learn about a medical standard covering this field, *DICOM (Digital Imaging and Communications in Medicine)*.

Frame Grabbing

The term *frame grabbing* describes the method that is used for capturing an image from the imaging device, including all transport and protocol issues. [Figure 3.1](#) shows the relevant subjects.

Figure 3.1. Getting an Image into a PC



In general, we discuss the following items:

- the camera itself (or another imaging device, such as an ultrasound imager);
- the transfer from the imaging device to the PC, including physical (cable) and protocol aspects;
- the power supply of the imaging device (especially of interest if small cameras are used);
- the frame grabbing device inside the PC; mostly a plug-in PC card;
- if possible, the camera control functions, which can be sent directly from the PC.

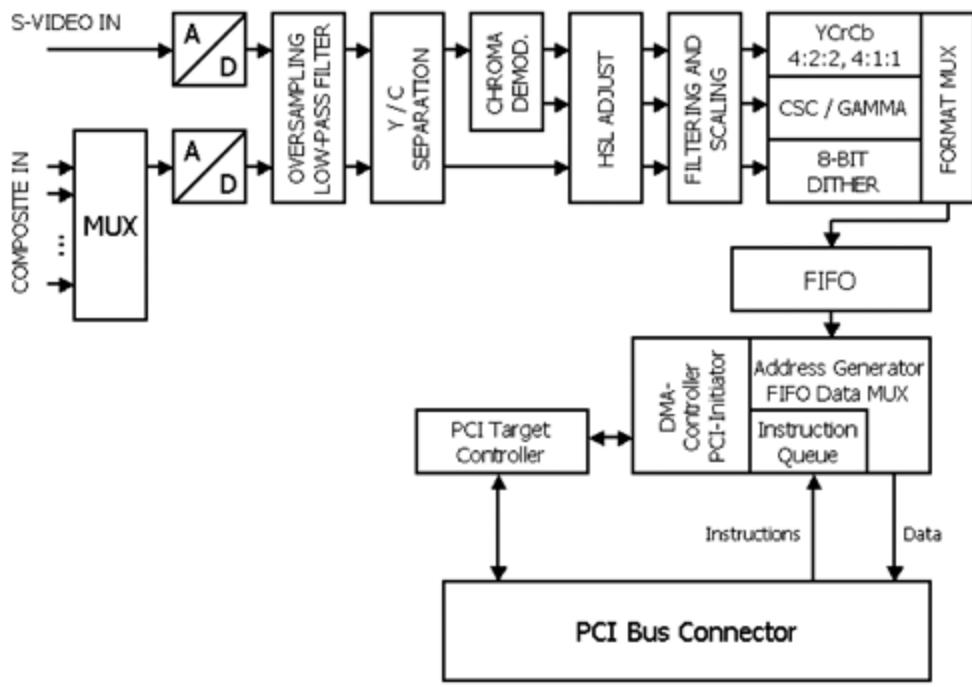
[Table 3.1](#) describes some examples.

Table 3.1. Frame Grabbing Methods

	IEEE 1394	USB	Camera Link	Analog
<i>Imaging device:</i>	digital	digital	digital	analog
<i>Transfer protocol:</i>	1394 camera	USB camera	Camera Link	analog video
<i>Frame grabber:</i>	1394 PCI	USB controller	CL video	analog FG
<i>Power supply:</i>	bus cable	bus cable	external	external
<i>Cable:</i>	1394 serial	USB serial	CL parallel	analog video

The three digital transfer methods are discussed later in this chapter. [Figure 3.2](#) shows a typical block diagram of a PCI frame grabber card. The card offers multiple video input possibilities, in-detail S-video, and composite signals. In the top row of blocks the separation of the luminance and chrominance signals into a HSL color model is clearly visible.

Figure 3.2. Typical Block Diagram of a PCI Frame Grabber



After some adjusting, filtering, and scaling, the signal is formed to a YC_bC_r model with the possibility of 4:2:2 or 4:1:1 color subsampling. The lower area of the block diagram describes the interfacing with the PC, in this case, via a PCI bus connector. Other possible interfacing standards are CompactPCI or PC104.

Camera Interfaces and Protocols

Here, we talk about three bus systems that combined cover a wide field in the digital camera world:

- *IEEE 1394*, or FireWire, or i.Link™, obviously now and in the future the standard for high-speed *serial* bus systems;
- *Universal Serial Bus* (or USB), which is very common with Web cameras;
- *Camera Link*, a high-end parallel camera connection.

IEEE 1394 (FireWire)

IEEE 1394 is a serial bus system; it got its name from the standard IEEE 1394-1995 and was originally developed by Apple Computers under the name ADB (Apple Desktop Bus). Later it was called FireWire and, finding its way into Sony Notebooks and Desktop computers, i.Link.

A number of main advantages, such as hot plugging or bus self-configuration, were the reasons that 1394 established itself in (at least) the multimedia world. In particular, the combination camcorder and multimedia notebook is very common.

In this section, we discuss the fundamentals of the 1394 standard, as well as the 1394 digital camera specification, which naturally is very important for this image distribution chapter. Finally, we discuss how to get images into LabVIEW and IMAQ; by the way, we also do this for similar bus systems in the next sections. For more details see [\[26\]](#).

Fundamentals

The following enumeration summarizes the most outstanding features of 1394 (taken from [\[26\]](#) and slightly modified). Of these, the most important are discussed in the sections below.

- *Scalable performance*: IEEE 1394 currently supports speeds of 100, 200, 400, and 800 Mbit/s; an increase towards 1,600 and 3,200 Mbit/s is planned.
- *Hot insertion and removal*: Devices can be dynamically attached or removed from the bus without the system being powered down.
- *Plug-and-play*: Each time a device is attached or detached, the bus is reconfigured and reenumerated. This configuration does not require intervention from a host system (PC).
- *Transaction types*: Support is provided for isochronous and asynchronous transfers.
- *Layered model*: Communications are based on a transaction layer, link layer, and physical layer protocols, realized both in hardware and software.
- *Support for 64 nodes*: Sixty-four node addresses (0-63) are supported on a single serial bus. The node address 63 is used as a broadcast address.
- *Peer-to-peer transfer support*: Serial bus devices can perform transactions among themselves, without the intervention of a host CPU.

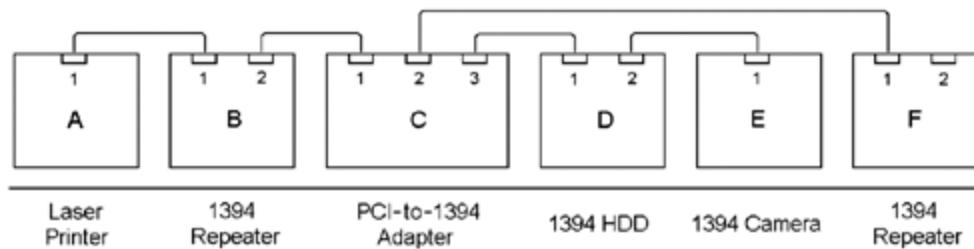
- *Fair arbitration support:* Arbitration ensures that isochronous applications are guaranteed a constant bus bandwidth, while asynchronous applications are permitted access to the bus according to a fairness algorithm.
- *Cable power:* Power available from the bus can be either sourced or sunk by a given node.

[Figure 3.3](#) shows a typical 1394 bus structure, illustrating some important issues regarding the bus topology:

- Though the logical structure of the entire system is a bus, it is realized by a number of point-to-point connections. Nevertheless, all devices on the bus can be accessed by all the others.
- Usually, the number of ports of a single device varies from 1 to 3 so that devices can be connected to each other and no further devices, such as hubs, are needed.
- Repeaters, as shown in [Figure 3.3](#), can be used to lengthen the bus cables.^[1] Moreover, every device with more than one port can be used as a repeater, even when the device is powered off.

[1] The cable length, as specified in the standard, has a maximum of 4.5 meters. However, there are cables on the market with a length of 10 meters and they work fine.

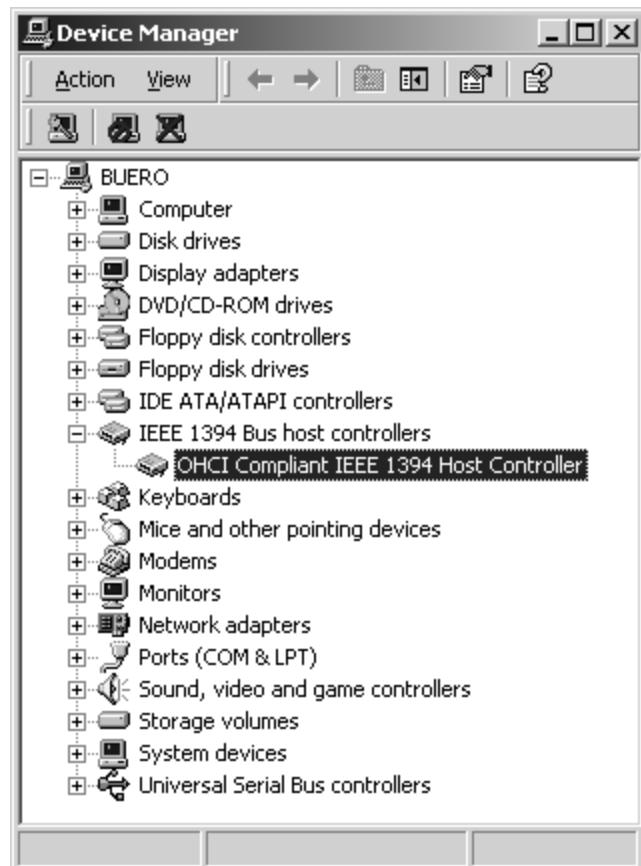
Figure 3.3. Typical 1394 Bus Structure with Single- and Multiport Devices



1394 Devices

[Figure 3.4](#) shows how a 1394 PCI card is listed in the Windows Device Manager. It depends on the high-level protocol where other devices appear; for example, a camera may be listed under Imaging Devices. Repeaters are not listed in the Windows Device Manager.

Figure 3.4. Windows Device Manager Listing 1394 Devices



The next figures show some 1394 devices that use different high-level protocols. For example, [Figure 3.5](#) shows a Zip100 drive and a hard drive that use the SBP2 protocol for mass storage devices. Both drives have two 1394 ports each.

Figure 3.5. 1394 Zip100 Drive and 1394 Hard Drive



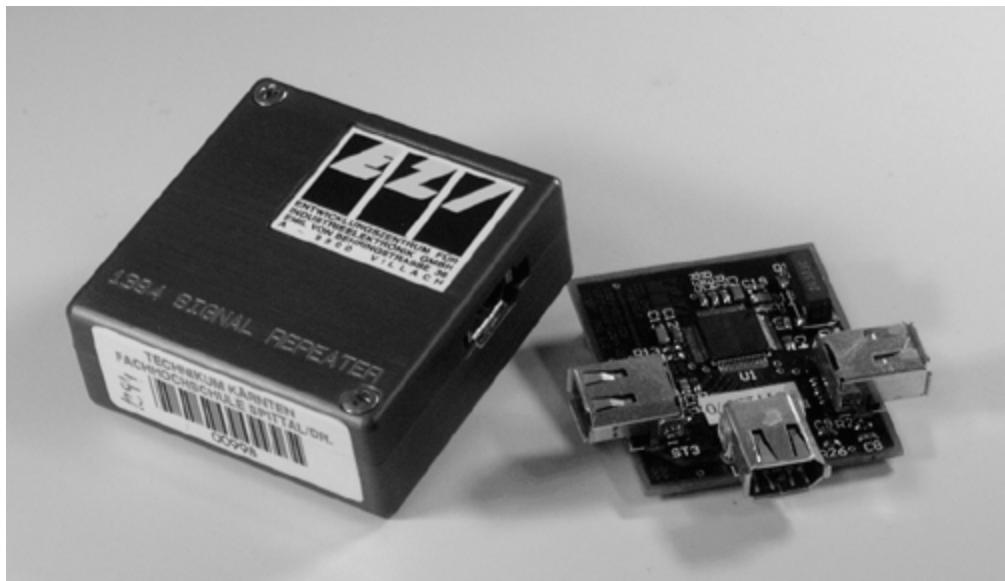
The 1394 cameras are naturally of interest for this book. [Figure 3.6](#) shows an example of a Sony 1394 digital camera, which also was used for an example in this book. The digital camera specification and protocol are discussed later in this chapter.

Figure 3.6. 1394 Video Camera



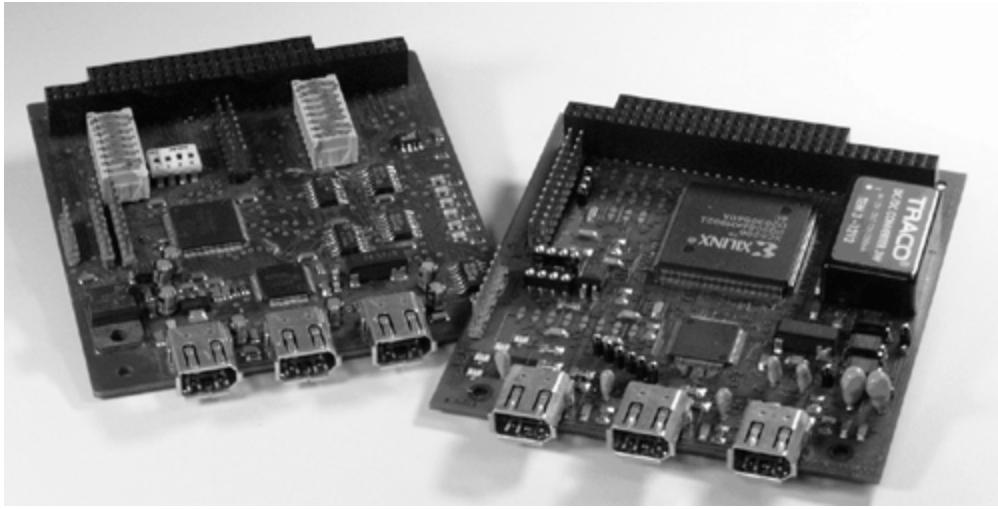
[Figure 3.7](#) shows two 1394 repeaters, one of them without housing. The integrated circuit on the circuit board is the *phy chip*, responsible for the 1394 physical layer. Real 1394 devices need at least one additional IC, the *link chip*, responsible for the 1394 link layer. Important for the functionality of the bus is that the phy chip can be powered over the bus, thus enabling repeating functionality even when the device is powered off.

Figure 3.7. 1394 Repeater (1STT Components; www.1stt.com)



The link chip is shown in [Figure 3.8](#) on the left circuit board. On the right board, the link layer as well as the transaction layer were integrated in an FPGA. Both boards represent examples of 1394 interface boards using the PC104 form factor.

Figure 3.8. 1394 PC104 Boards (1STT Components; www.1stt.com)



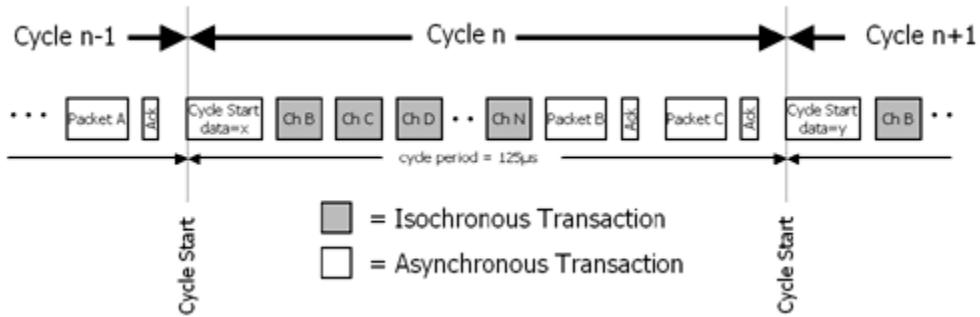
1394 Communications

As mentioned before, IEEE 1394 supports two data transfer types:

- *Asynchronous transfers* do not require delivery at a constant data rate. They do not require a constant bus bandwidth and therefore do not need regular use of the bus, but must be given fair access over time. Moreover, they require a response from the target node, resulting in an additional transaction. Typical examples for asynchronous transactions are control commands to a device or data to be stored on mass storage devices, such as hard drives.
- *Isochronous transfers* require data delivery at constant intervals. Here, not a unique device address but a channel number is defined, and the data is streamed to one or more nodes responding to this channel number. These transfers require a guaranteed bandwidth and therefore regular bus access, and they have higher bus priority than that of asynchronous transfers. Typical examples are audio or video data streams.

The bus bandwidth allocation is based on a 125 µs interval called the cycle period. In this interval, a mix of isochronous and asynchronous transactions can share the overall bus bandwidth. As shown in [Figure 3.9](#), isochronous transfers are based on a 6-bit channel number, which are used by isochronous listeners to access reserved areas of memory buffer.

Figure 3.9. Isochronous and Asynchronous Transactions [[26](#)]



Asynchronous transfers target a particular node and are all together guaranteed a minimum of 20% of the overall bus bandwidth. This implies that a particular node is not guaranteed a particular bandwidth but is guaranteed fair access to the bus; this mechanism is explained later.

The resulting maximum data payload size for both asynchronous and isochronous transfers is

shown in [Table 3.2](#).

Table 3.2. Maximum Data Payload Size for Asynchronous and Isochronous Transfers

Cable Speed	Maximum Data Payload Size (Bytes)	
	Asynchronous Transfers	Isochronous Transfers
100 Mbit/s	512	1024
200 Mbit/s	1024	2048
400 Mbit/s	2048	4096
800 Mbit/s [†]	4096	8192
1.6 Gbit/s [‡]	8192	16384
3.2 Gbit/s [‡]	16384	32768

[[†]] announced

[[‡]] planned

As indicated above, the hardware and software implementation is realized by four protocol layers, each equipped with a set of services for communication purposes with the application:

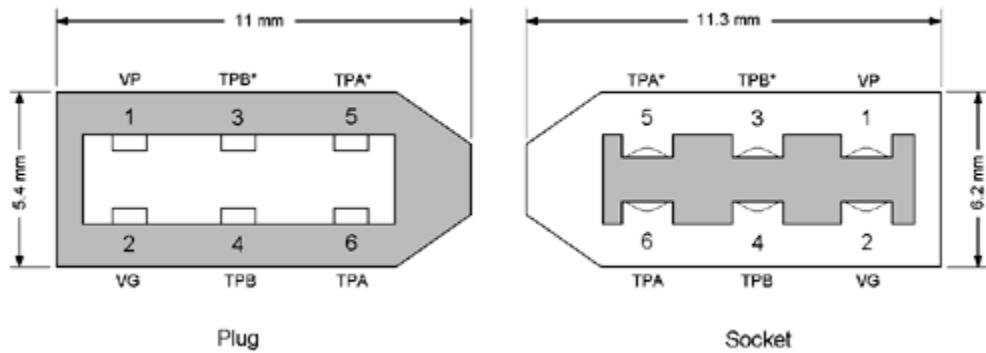
- *Bus management layer*: It provides bus configuration and management activities.
- *Transaction layer*: Here, only asynchronous transfers are processed. Each transaction consists of a request subaction and a response subaction.
- *Link layer*: It provides translation of transaction layer request or response, address and channel number decoding, and CRC error checking.
- *Physical layer*: It provides the electrical and mechanical interface for the transmission of the raw data transferred over the bus.

Read more about 1394 communication in [\[26\]](#).

1394 Cables, Connectors, and Signals

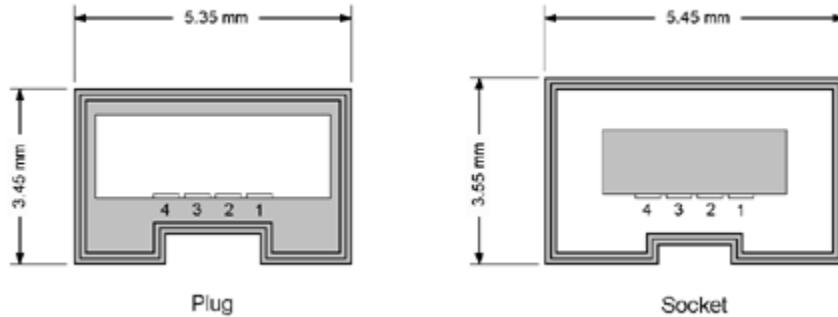
The IEEE 1394 standard cable consists of two 6-pin connectors and a single 6-pin cable. The connectors on both sides of the cable are identical, unlike other systems, e.g., USB.

Figure 3.10. 1394 6-Pin Connector (Plug and Socket)



Cross sections of this 6-pin connector (plug and socket) are shown in [Figure 3.10](#). Two of the six wires carry bus power (VP and VG); the other four form two twisted wire pairs named TPA/TPA* and TPB/TPB*.

Figure 3.11. 1394 4-Pin Connector (Plug and Socket)



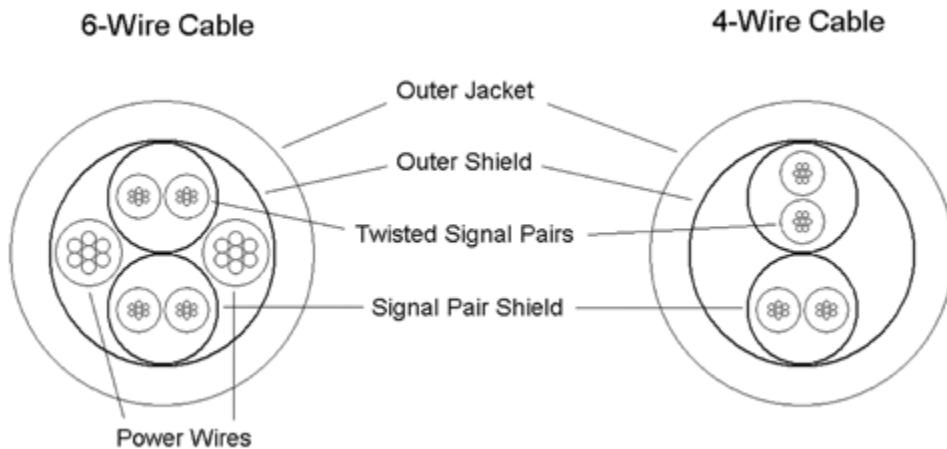
Later, a 4-pin connector was designed by Sony especially for hand-held or battery operated devices, for example, camcorders, since such devices do not need to draw power over the bus ([Figure 3.11](#)). These cables consist only of the two twisted wire pairs. Moreover, the small 4-pin connector fits better into small designs.

So, 1394 supports three different types of cables:

- standard cables with identical 6-pin connectors on both ends;
- cables with identical 4-pin connectors on both ends, for example, for connecting a camcorder to a notebook PC;
- cables with a 4-pin connector on one end and a 6-pin connector on the other end.

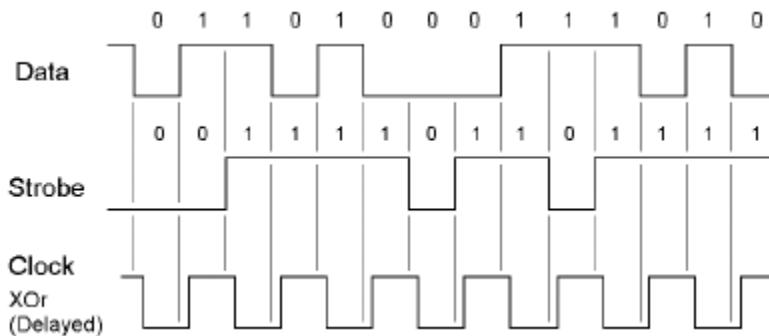
[Figure 3.12](#) shows cross sections of the required cable types; naturally, the cables with 4-pin connectors and 6-pin connectors require only four conductors.

Figure 3.12. Cross Sections of 4-Conductor and 6-Conductor Cables



The signaling over the two twisted wire pairs is based on data strobe encoding (DSE); see [Figure 3.13](#). If one of the twisted pairs carries the data, the other one carries a strobe signal, which can be used to derive a clock signal by an XOR operation. The main advantage of this encoding is that only one of the two signals changes state at a time, thereby reducing electromagnetic interference.

Figure 3.13. Data Strobe Encoding



1394 Bus Configuration

The main difference from other bus systems like USB is that 1394 can work without a host processor. This enables a number of configurations, such as direct data streaming from a camera to a hard drive if an appropriate protocol is installed on both devices. On the other hand, the bus has to perform a number of procedures:

- bus initialization (bus reset),
- tree identification, and
- self-identification.

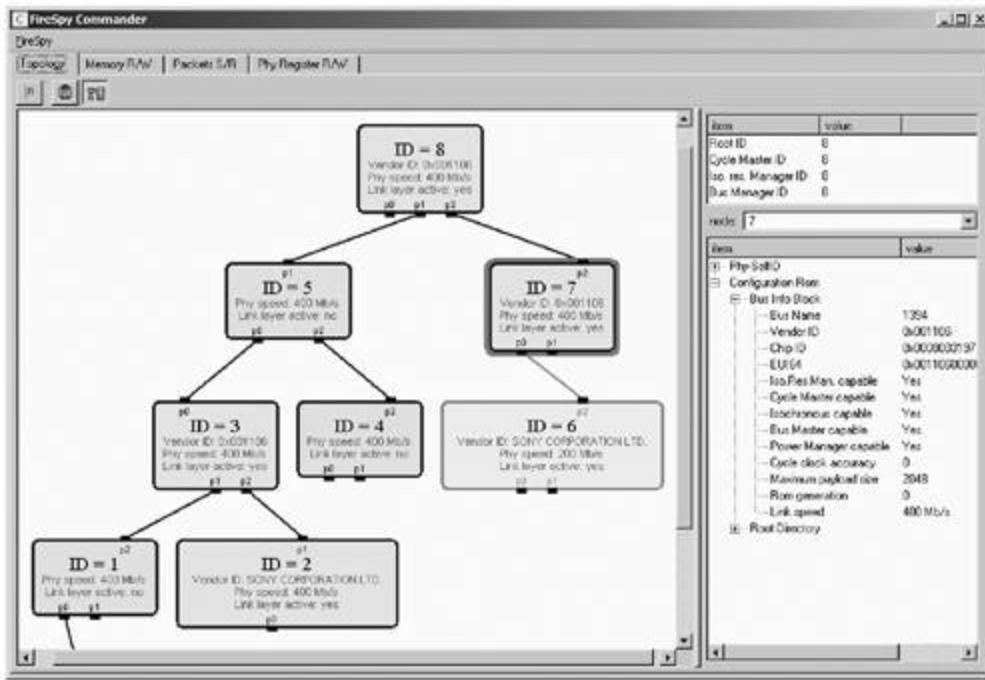
A *bus initialization* or bus reset occurs when power is applied to a node or removed from a node or when a node is attached to or removed from the bus; these occurrences force all nodes to return to their initial states. This reset clears all topology from the nodes and prepares all nodes to begin the tree identification process.

The *tree identification* process redefines the topology of the bus in accordance with the new device situation. After the process, a single node will be the root node and therefore each port will be identified as a parent or a child.[\[2\]](#) Any node can become the root node regardless of where it connects into the network of nodes.

[2] A port identified as a child node points to a node that is farther away from the root.

After that, the *self-identification* process begins, during which all nodes select a unique physical ID and return self-ID packets that identify themselves and parameters that describe their capabilities. [Figure 3.14](#) shows an example of a 1394 bus topology using a FireWire data analyzer.

[Figure 3.14. Example of a 1394 Bus Topology](#)



1394 Bus Management

After the completion of the configuration process, nodes have to be found to take over certain roles in the bus management process. The following tasks are defined in the 1394 specification:

- *Cycle Master* initiates the start of isochronous transactions at regular 125 μ s intervals.
- *Isochronous Resource Manager* tracks the number of isochronous channels available and the amount of isochronous bus bandwidth available and allocates these resources when requested by serial bus nodes.
- *Bus Manager* publishes maps that specify the topology of the serial bus and that indicate the maximum packet speed supported by each cable segment in the system and other tasks [\[26\]](#).

Another process described here among the bus management features is *arbitration*, which means that each node is guaranteed fair access to the bus. Both isochronous and asynchronous arbitration processes are possible; the isochronous process simply guarantees a certain amount of bus bandwidth; therefore, we talk only about the asynchronous process.

The process of asynchronous fair arbitration is based on the following three items:

- The *fairness interval* ensures that each node needing to initiate a transaction gets fair access to the bus. It has to be ensured that each node that has an asynchronous transaction pending is permitted to obtain bus ownership once during each fairness interval.

- The *arbitration enable bit* of a node must be cleared after its asynchronous transaction has started. The reason is that once a node has performed an asynchronous transaction, it is not allowed to request bus ownership until the next fairness interval begins.
- The *arbitration reset gap* follows the fairness interval and sets all arbitration enable bits again. Note that the arbitration reset gap occurs automatically after all nodes have finished their transactions and all arbitration enable bits are cleared.

Power Management

The power distribution part of the 1394 specification defines four different types of node power classes:

- power providers,
- alternate power providers,
- power consumers, and
- self-powered nodes.

For example, an industrial camera or a repeater is a power consumer; a video camcorder is a self-powered node.

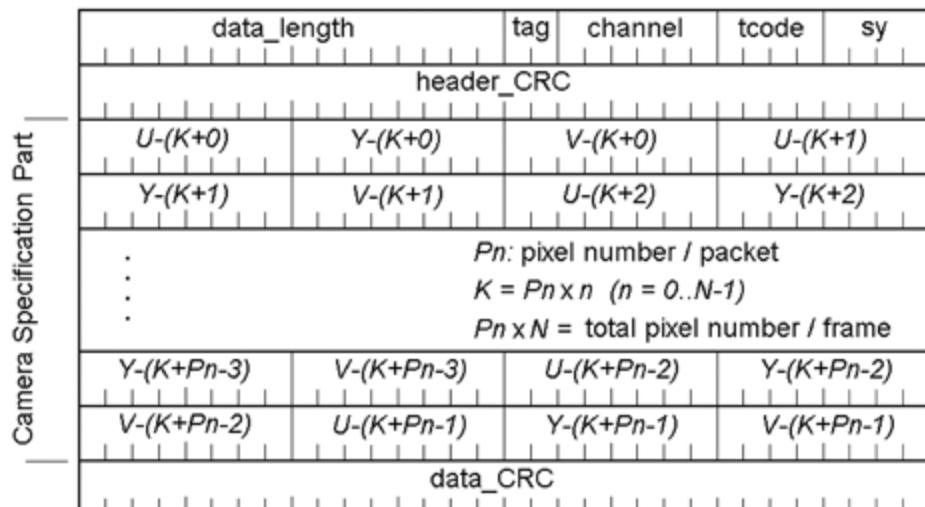
1394 Digital Camera Specification

Especially interesting for our application field is the specification for 1394 cameras. You can download the entire document from www.baslerweb.com or www.1394ta.org.

[Figure 3.15](#) shows an example of an isochronous stream packet carrying 4:4:4 YUV video data. The first two lines, containing the header and the header CRC, are identical in every isochronous stream packet; [3] the rest of the lines contain the video data payload.

[3] The content of the fields is not discussed here; read more about it in the specifications or in [\[26\]](#).

Figure 3.15. Isochronous Stream Packet (Video Data YUV 4:4:4)



Another important field is the setting of some important camera values (here called features),

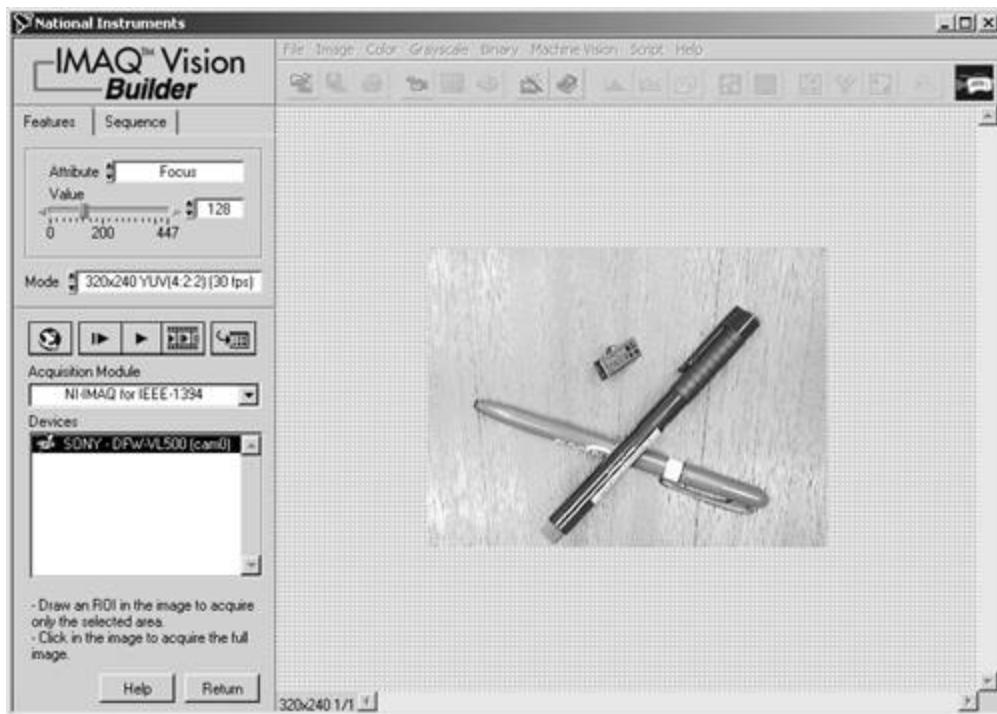
such as Brightness, Sharpness, Hue, Saturation, Focus, Zoom, and some more, which are located in the register area. Again, details can be obtained from the specifications; but we see in the next section that a detailed knowledge of these issues is not important, because LabVIEW performs a number of tasks for us.

1394 Images in LabVIEW

Using images from 1394 cameras in LabVIEW and IMAQ Vision is easy since National Instruments released a special plug-in for IMAQ Vision called "IMAQ Vision for 1394 Cameras." If this tool is installed, the IMAQ Vision Builder offers an additional acquisition module called "NI-IMAQ for IEEE-1394." If this module is selected, images can be acquired from the 1394 camera just as from any other vision device.

Definitely more interesting is that some properties of the camera, such as Zoom, Focus, Brightness, and others, can directly be set by a control in the IMAQ Vision Builder (see [Figure 3.16](#)). The camera then reacts immediately to the property change. Of course, this is also possible directly from LabVIEW; we try it in the following exercise.

Figure 3.16. 1394 Camera Image and Properties in IMAQ Vision Builder

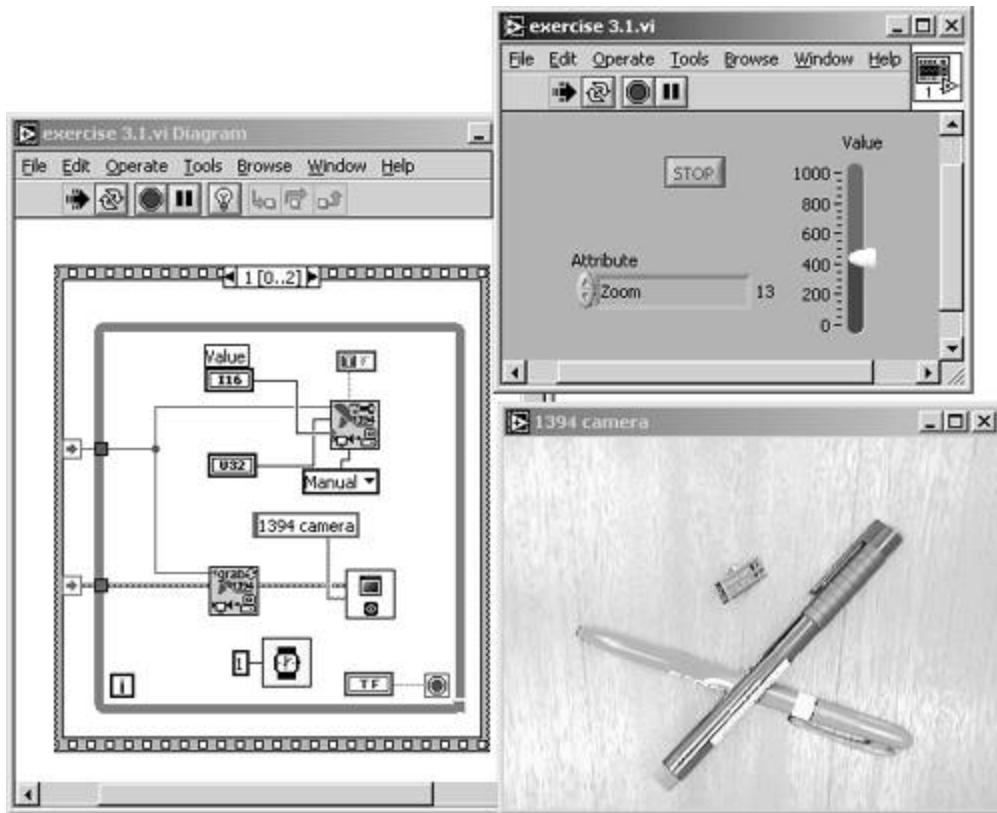


Exercise 3.1: 1394 Camera Images (1394 camera required).

Create a LabVIEW program that captures an image from a 1394 camera and changes some camera attributes. For the image capturing use the functions [IMAQ1394 Grab Setup](#), [IMAQ1394 Grab Acquire](#), and [IMAQ1394 Close](#) (see [Figure 3.17](#)).

The properties of the camera can be changed with the function [IMAQ Vision Attribute](#). The camera property is selected with the Attribute control and the value can be transmitted by the respective slider control.

Figure 3.17. Setting 1394 Camera Properties in LabVIEW



In general, 1394 cameras are a great opportunity for vision engineers, especially if you consider the digital image transfer, the asynchronous transmission of camera properties, and the camera power supply, all of them using a single 6-wire serial cable.

Universal Serial Bus (USB)

The Universal Serial Bus (USB) was intended to make an end with the number of PC interfaces: There were (and still are!) the serial (or RS-232) port, the parallel port, PS/2 ports for keyboard and mouse, and some other interfaces inside the PC itself. The main idea was to use only a single interface (e.g., USB) for all these connections.

The reason why this vision has not come true was the relatively low data rate of 12 Mbit/s, which has turned out to be a major disadvantage of USB, especially if mass storage devices (hard drives or CD writers) are connected to a system.

Anyway, USB is established with a certain number of devices, such as mice, some disk drives, keyboards, and (of interest for us) cameras. In the following sections, I explain some fundamentals and key data of USB, some examples of USB devices, and some details, which include the physical, the signaling, and the data environment. Finally, I mention the new standard USB 2.0 and what it means to the existing USB world.

Fundamentals

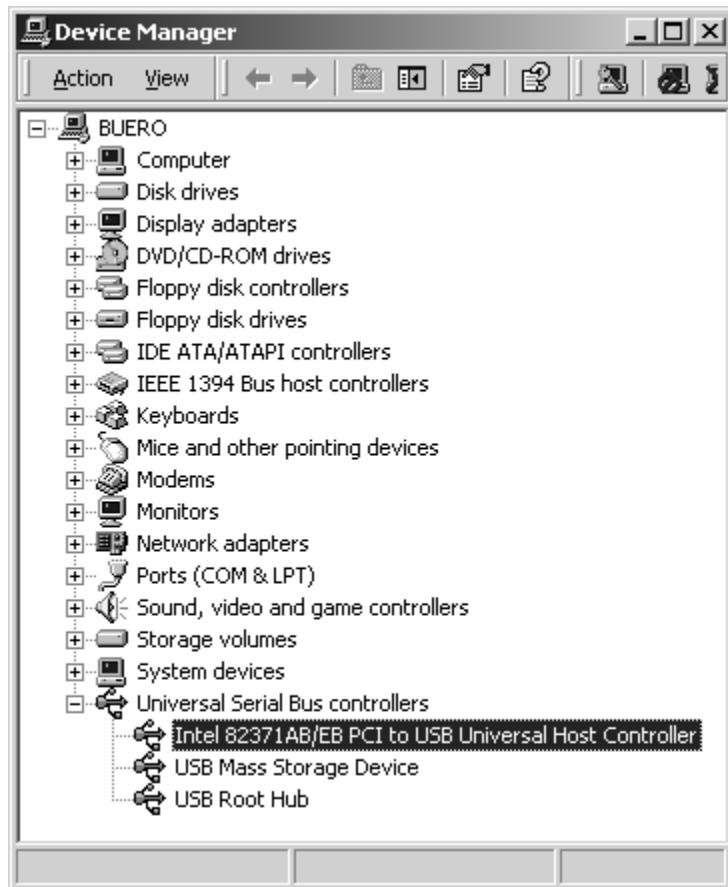
USB can connect up to 127 devices with a host. The USB standard 1.1 distinguishes between two types of devices:

- devices such as mice, keyboards, cameras, which are called *functions*;
-

- USB hubs, which provide one or more ports for the connection of other devices.

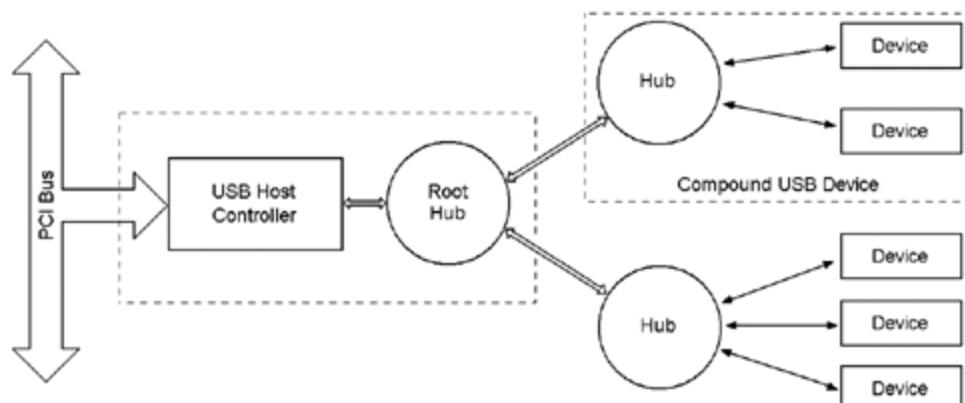
[Figure 3.18](#) shows how these devices are listed in the Windows Device Manager: the USB controller itself, which today is usually integrated in the PC main board, the USB root hub, and the devices (here a mass storage device) appear in the same submenu.

Figure 3.18. Windows Device Manager Listing USB Devices



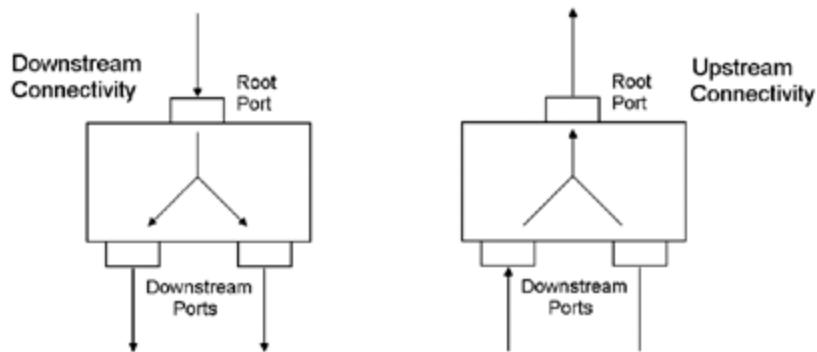
This structure can again be seen in [Figure 3.19](#). The controller, usually a PCI card, integrates the USB host controller and the USB root hub, to which all other devices or hubs must be connected.

Figure 3.19. USB Hub Types [[27](#)]



[Figure 3.19](#) also shows that the USB structure is a tree type; starting at the host (root), the additional hubs or devices form branches or leaves. Therefore, as shown in [Figure 3.20](#), two data directions can be specified:

Figure 3.20. USB Hub Performing Downstream and Upstream Connectivity [27]



- upstream (to the host);
- downstream (from the host).

Two different types of plugs ensure that these directions cannot be mixed; mixing them would result in a forbidden loop structure: in the upstream direction the well-known (flat) A-plug, and downstream the almost quadratic B-plug. Read more about cables and plugs in *Physical Environment*.

Another limitation (besides the maximum of 127 devices) is the number of tree levels, which is limited to seven, including the root hub. This implies that the number of hubs connected logically in series is limited to five.

USB Devices

Any USB communication can be performed at one of two (three with USB 2.0) speed levels:

- low speed (1.5 Mbit per second);
- full speed (12 Mbit per second);
- high speed (only USB 2.0; 480 Mbit per second).

This results in a number of tasks that USB hubs have to perform. ([Figure 3.21](#) shows a typical USB hub.) Usually a hub has to do the following tasks:

Figure 3.21. USB Hub with Four Ports



- repeat data arriving at its ports in upstream or downstream direction; [\[4\]](#)

[4] A device that performs only this function is called a "repeater".

- recognize devices attached to its ports;
- intercede between the different bus speeds;
- distribute and control power to its ports. This is an important issue and results in the big advantage that devices connected to a USB port need not have their own power supply. Therefore, the hubs are divided into two types:
 - bus-powered hubs, which receive their own energy from the bus. They are limited by the amount of power they can get from the bus and can therefore supply a maximum of four ports.
 - self-powered hubs (no limitations concerning power).

Moreover, the USB specification defines a number of device classes, which are listed below. This list is taken from [\[27\]](#) and does not give any information about the definition state of the device classes; that means that some specifications are not yet released.

- *Audio Device Class*: Devices that are the source of real-time audio information.
- *Communications Device Class*: Devices that attach to a telephone line (not local area networks).
- *Display Device Class*: Used to configure monitors under software control.
- *Human Interface Device Class (HID)*: Devices (mice, keyboards) manipulated by end users.
- *Mass Storage Device Class*: Devices (floppy drives, hard drives, and tape drives) used to store large amounts of information.
- *Image Device Class*: Devices that deal with images (either still or moving).
- *Physical Interface Device Class (PID)*: Devices that provide tactile feedback to the operator. For example, a joystick with variable resistance for simulating increased stick forces and turbulence.

- *Power Device Class:* Devices that provide power to a system or to peripherals. For example, uninterruptable power supplies and smart batteries, which can be either stand-alone devices or integrated into the interface.
- *Printer Device Class:* May be merged with storage device class.

[Figure 3.22](#) shows an example for a USB camera and for a USB mass storage device.

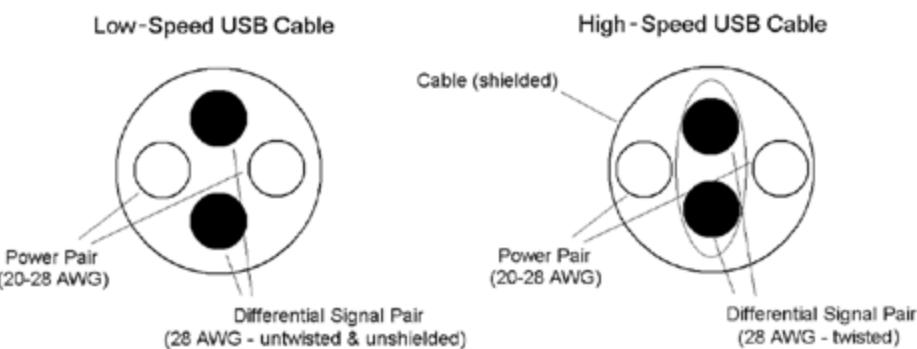
Figure 3.22. USB Mass Storage Device and USB Camera



Physical Environment

The cross section of USB cables is shown in [Figure 3.23](#). The cable on the left is intended only for low speed (1.5 Mbit/second) and does not require shielding. For low speed, the cable length should not exceed three meters.

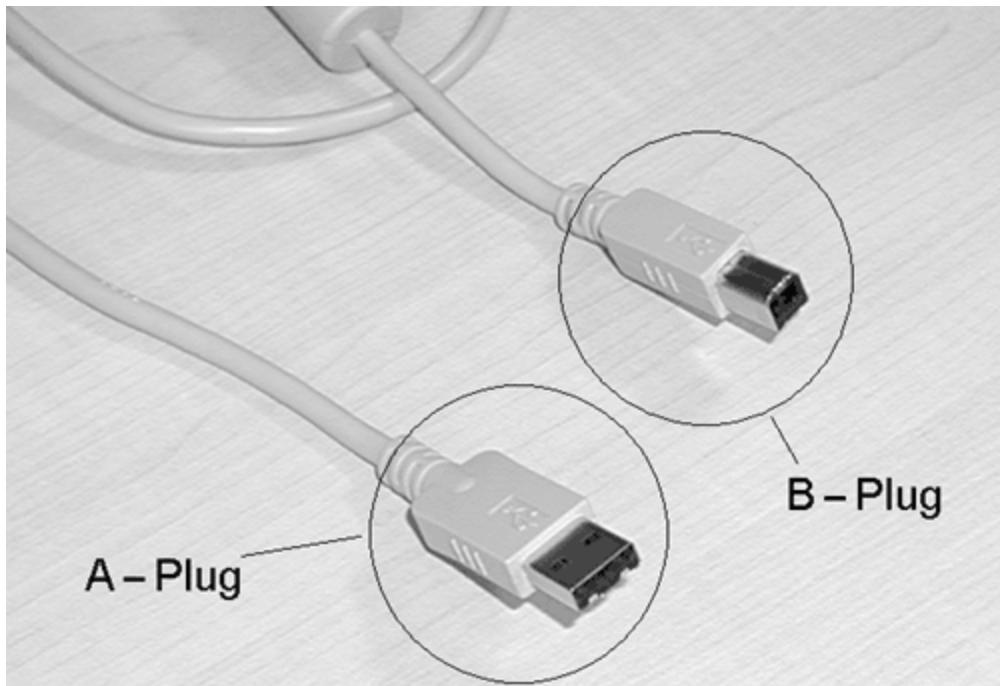
Figure 3.23. Cross Sections of Low-Speed and High-Speed USB Cables [27]



Full speed (12 Mbit/second) USB cables must be both shielded and twisted pair for the differential data lines (right side of [Figure 3.23](#)). The maximum cable length for full speed is five meters if the signal propagation delay is equal to or less than 30 ns over the entire cable length. If the propagation delay is higher, the cable must be shortened [27].

The other two wires that can be seen in the cross sections carry cable power, which is 5 Vdc and can provide from 100 to 500 mA. A number of peripherals do not need this cable power, for example, printers. Finally, [Figure 3.24](#) shows a USB cable illustrating both an A- and a B-plug.

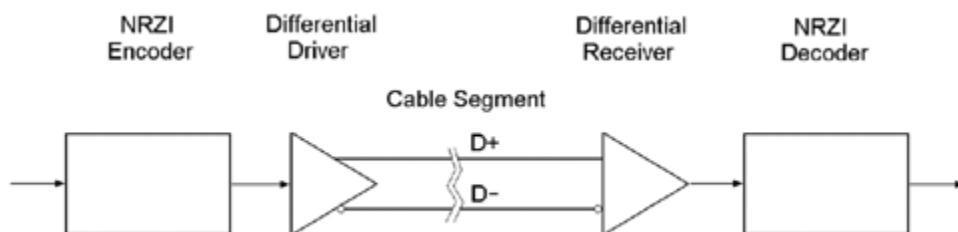
Figure 3.24. USB Cable with A- and B-Plug



Signaling Environment

The USB standard uses a special type of data encoding called NRZI (non-return to zero, inverted). This encoding and the use of differential signaling, which is shown in [Figure 3.25](#), ensure data integrity and significantly decreases noise problems.

Figure 3.25. USB Cables Using NRZI Encoding and Differential Signaling [27]



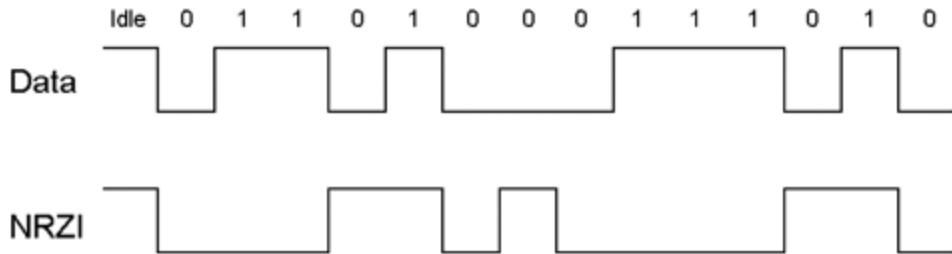
Other tasks are performed by the monitoring of the differential data lines:

- detection of device attachments and detachments by monitoring the voltage levels of the differential data lines;
- detection of the possible device speed (low or full speed; full speed devices have a pull-up resistor on the D+ line, low speed devices have it on the D- line).

You can find more details about NRZI in [27].

Figure 3.26 shows the NRZI data encoding itself. The Data line shows the original data, and the NRZI line contains the encoded data; the NRZI signal remains at its status when data is 1 and changes status when data is 0.

Figure 3.26. NRZI Encoding [27]



A problem may occur if the data contains a large number of consecutive 1s. Therefore, after six consecutive 1s, a transition of the NRZI signal is forced, thereby ensuring that the receiver can detect a transition in the NRZI signal at least at every seventh bit. The transition is forced by additional bits in the data stream, resulting in a *stuffed data* stream. This method is called *bit stuffing*.

Data Environment

The USB specification defines four transfer types [27]:

- *Interrupt transfer*: Interrupt transfers are linked to devices usually using interrupts for their communication, such as mice and keyboards. Because USB does not support hardware interrupts, these devices must be polled periodically.
- *Bulk transfer*: Sending data to a printer is a good example of a bulk transfer: It is a large amount of data, but it does not have a periodic or transfer rate requirement.
- *Isochronous transfer*: Isochronous transfers require a constant delivery rate (similar to IEEE 1394). They are used, for example, for video or audio data.
- *Control transfers*: Control transfers are used for the transfer of specific requests to USB devices. They are mostly used during the device configuration process.

Individual transfers consist of a number of packets, which are themselves combined with transactions. Packets are categorized into four types:

- *Token packets* are sent at the beginning of a USB transaction to define the transfer type.
- *Data packets* follow token packets in transactions with data payload to or from USB devices.
- *Handshake packets* are typically returned from the receiving agent to the sender, thus providing feedback relative to the success or the failure of the transaction. In some cases, the USB device being requested to send data to the system may send a handshake packet to indicate that it currently has no data to send [27].
- *Special packets*: An example of a special packet is the preamble packet, which enables low-speed ports.

The transactions are divided into three categories:

- IN transactions,
- OUT transactions, and
- Setup transactions.

Moreover, the communication at the application level takes place over *pipes*, which act as logical channels and connect the application with a virtual end point at the USB device, no matter whether the device is connected directly to the root hub or to one or more additional hubs.

You can read more about pipes, transfers, transactions, and packets in [27].

USB 2.0

Version USB 2.0 of the Universal Serial Bus adds the following amendments to the standard:

- The (already discussed) high-speed standard with 480 Mbit per second;
- The additional task for 2.0 hubs to negotiate between different speed modes;
- The necessity for split transactions if a USB 2.0 host has to communicate with a slow full-speed device.

The other principles of the standard as well as the cables remain unchanged.

USB Images in LabVIEW

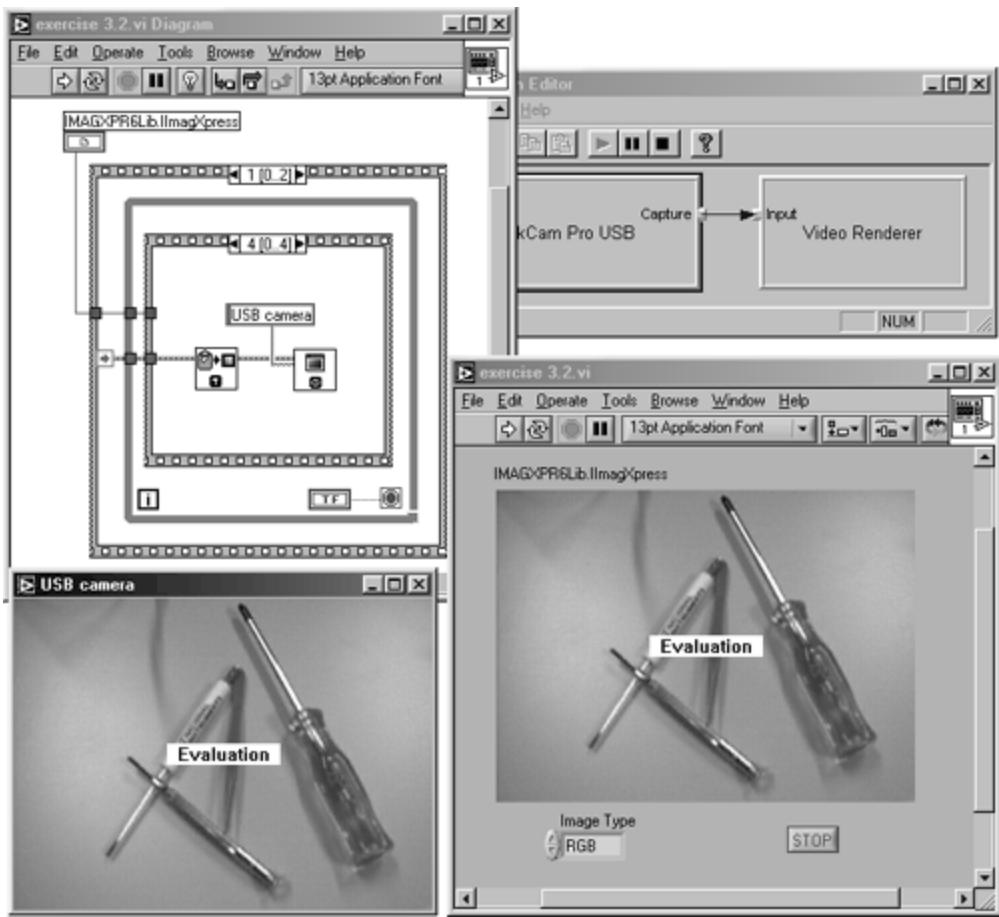
Getting images from USB cameras into LabVIEW and IMAQ Vision is not as easy as it is with 1394 cameras; the reason is simple: IMAQ Vision does not support USB. Here, we have to use some tricks, providing the following functions:

1. Display the image from the USB camera in a resizeable window.
2. Somehow capture the image from this window and copy it to the PC's clipboard.
3. Read the image from the clipboard and display it in an IMAQ Window.

For functions 1 and 2, we need additional software:

- For displaying the image from the USB camera, you can use any proprietary software that came with your camera; or, you can also use a tool like `graphedt.exe`, which is part of Microsoft's DirectX SDK and can be downloaded from <http://www.microsoft.com>. This tool lists all of your imaging devices and displays an image in a video renderer window (see also [Figure 3.27](#)).

Figure 3.27. Importing USB Camera Images in LabVIEW



- Transferring the image into the clipboard is not easy. I suggest that you use a software tool like ImageXpress from Pegasus Software (<http://www.pegasustools.com>). The trial version will work here. After the package is installed, you can import an additional ActiveX control in LabVIEW; for example, if you downloaded version 6 of ImageXpress, it is called **IMAGXPR6Lib.IImagXpress.ctl**.

With this equipment, we can do the following exercise.

Exercise 3.2: USB Camera Images (USB camera and additional software required).

Create a LabVIEW VI that captures an image from a USB camera (Figure 3.27). Use a special software tool (e.g., graphedt) for the image display and another (e.g., ImageXpress) for the capture.

After importing the ImageXpress ActiveX control into LabVIEW, use the methods Capture and Copy through two LabVIEW Invoke Nodes. Note that the image, which should be captured and copied, must be placed over the ActiveX control in the LabVIEW Front Panel window. For optimization, the control itself can be resized to fit the display window.

Once the image is stored in the clipboard, it can be transferred into an IMAQ Window with the function **IMAQ_ClipboardToImage**. See [Figure 3.27](#) for further details.

Camera Link

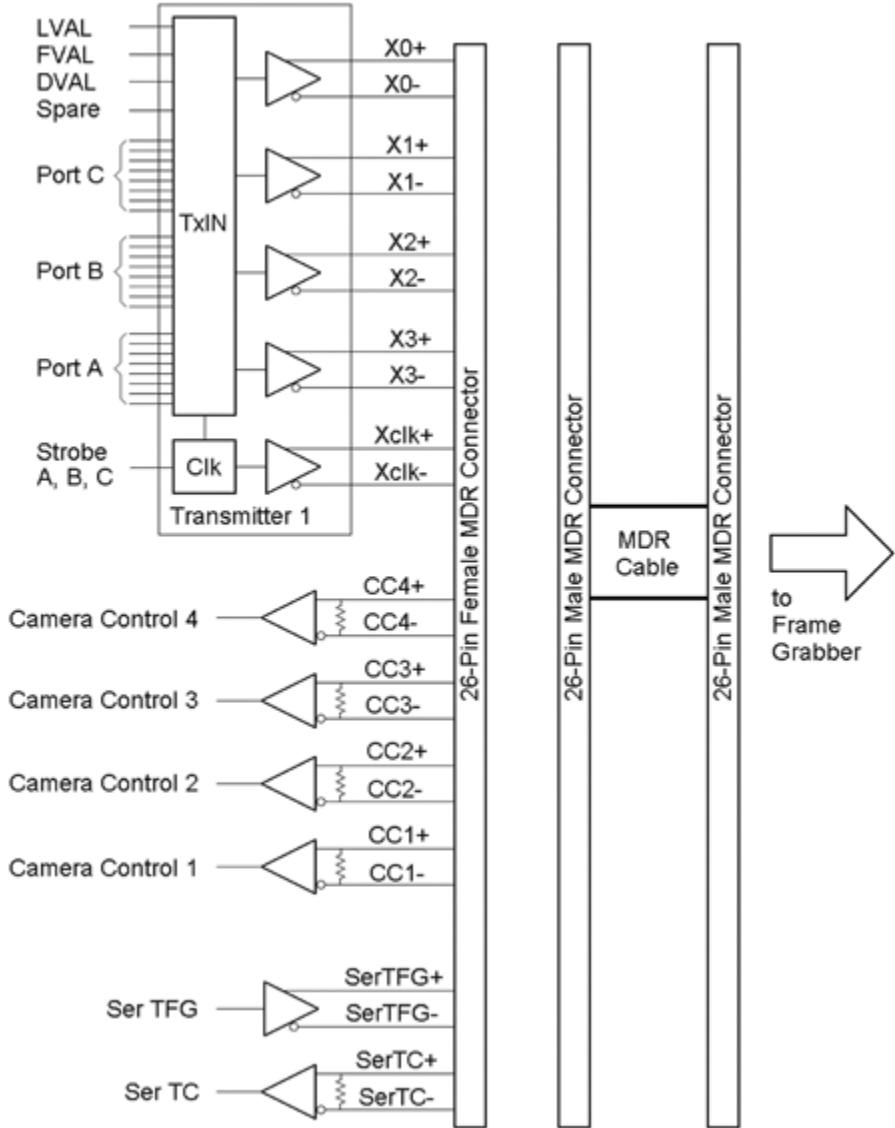
Although Camera Link is not really a bus system—because it is only used for connecting a digital camera to a frame grabber—it is discussed. The Camera Link interface is important for digital video systems because it allows really high data rates.

[Figure 3.25](#) on page 102 shows the use of differential signaling for USB purposes. The main advantage of this technology is immunity from disturbance and interference signals from outside of the cable area. Imagine a disturbance signal that influences both of the signal wires; because only the signal *difference* of these two wires is used, the disturbance signal is (more or less) deleted in the subtraction result.

Camera Link uses this technology as well. Here, a standard called *low voltage differential signaling* (LVDS) is used for the cable transmission. National Instruments introduced a system called *Channel Link*, consisting of a transmitter (converting 28 bits of CMOS/TTL data into four LVDS streams) and a receiver (converting the streams back to CMOS/TTL). In addition, a phase-locked clock is transmitted over a fifth LVDS stream. The major advantage is the significant reduction of the number of wires from the transmitter to the receiver, resulting in smaller connectors and cable outlines.

[Figure 3.28](#) shows the base configuration of the Camera Link interface, which is based on Channel Link. In this configuration, a single transmitter/receiver pair is used together with four separated LVDS streams reserved for general-purpose camera control and two LVDS pairs reserved for serial communication between camera and frame grabber.

Figure 3.28. Camera Link Block Diagram (Base Configuration)



The base configuration, using a single transmitter/receiver pair, is limited to 28 bits of video data, which may not be sufficient for all purposes. [Figure 3.29](#) shows the extension of the Camera Link interface to a medium configuration, which includes another transmitter/receiver pair, and to a full configuration, using in total three Channel Link transmitter/receiver pairs.

Figure 3.29. Camera Link Block Diagram (Extension for Medium and Full Configuration)

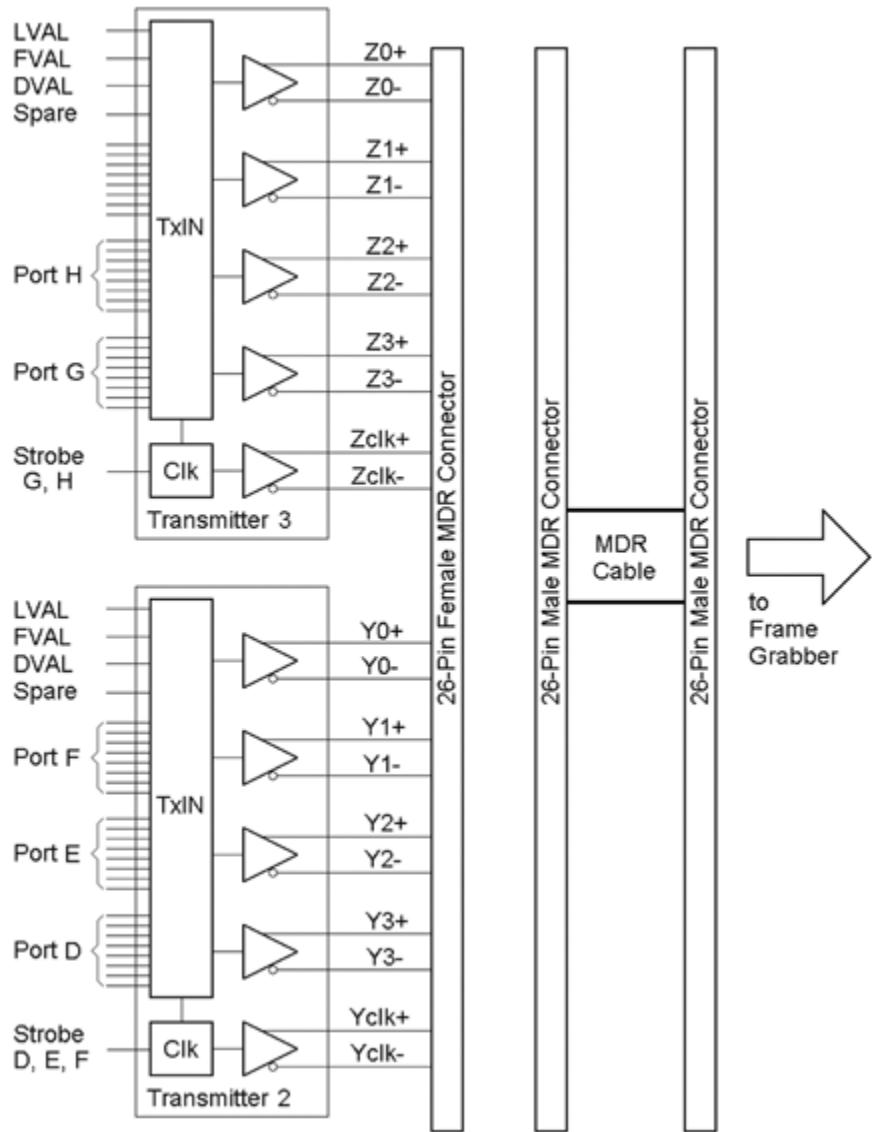


Table 3.3. Camera Link Configurations

	Camera	Ports A,B,C	Ports D,E,F	Ports G,H
Control Lines	(X0-X3, Xclk)	(Y0-Y3, Yclk)	(ZO-Z3, Zclk)	
<i>Base configuration:</i>	Yes	Yes		
<i>Medium configuration:</i>	Yes	Yes	Yes	
<i>Full configuration:</i>	Yes	Yes	Yes	Yes

Table 3.3 is an overview of all three possible configurations. The base configuration requires one single MDR cable; medium and full configuration both require two MDR cables.

Camera power is not provided through the Camera Link connectors, but it can be supplied to the camera through a separate connector. In fact, most of the camera manufacturers use their own standards.

You can get more information about Camera Link from a number of suppliers, for example, from www.basler-mvc.com.

Comparison of Digital Camera Interfaces

Now that we have discussed the most important bus systems for imaging applications in detail, we can extend [Table 3.1](#) and compare in [Table 3.4](#) the most interesting features of the three camera interfaces.

[[Team LiB](#)]

 PREVIOUS  NEXT 

Compression Techniques

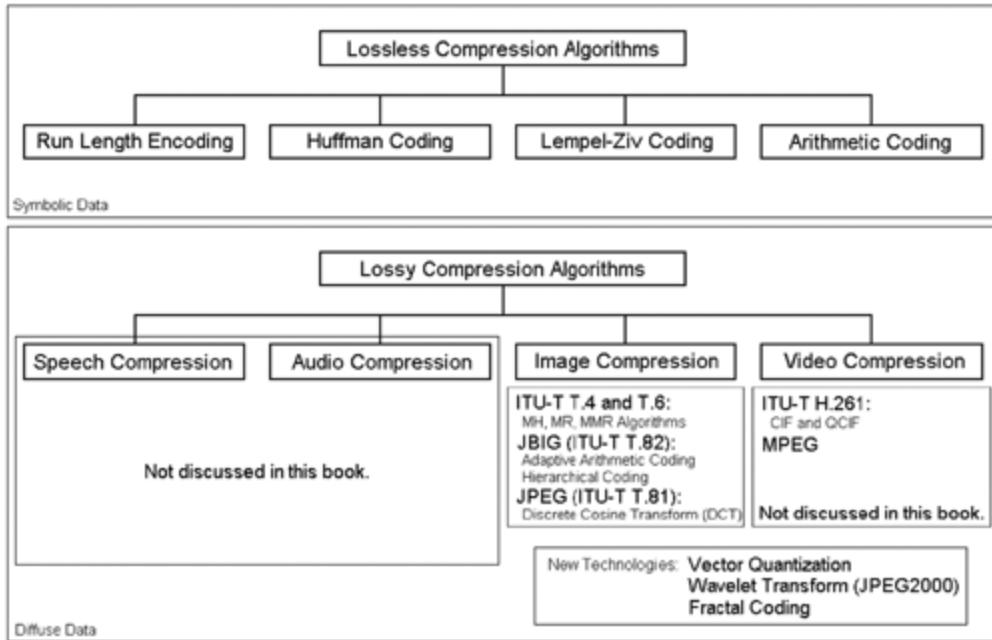
The following section discusses some techniques that make transport and storage of data, especially of image data, more effective. It distinguishes between lossless compression techniques, which enable the reconstruction of the image without any data loss and *lossy* compression techniques. Two good books for the details, from which most of the algorithm descriptions are taken, are [28] and [29].

Table 3.4. Comparison of Digital Camera Interfaces

	IEEE 1394	USB 2.0	Camera Link
<i>Max. data rate:</i>	800 Mbit/s	480 Mbit/s	~ 7.14 Gbit/s
<i>Architecture:</i>	Peer-to-peer	Host oriented	Peer-to-peer
<i>Max. devices:</i>	63	127 (incl. hubs)	—
<i>Max. cable length:</i>	4.50 m	5 m	~ 10 m
<i>Total poss. distance:</i>	72 m	30 m	~ 10 m
<i>Wires for 8 bit/pixel:</i>	4	2	10
<i>Bidirectional comm.:</i>	Yes (asynchronous)	Yes (asynchronous)	Additional channel
<i>Power supply:</i>	8-40 V, 1.5 A	5 V, 500 mA	—

[Figure 3.30](#) illustrates the most common compression techniques and algorithms. According to [29], algorithms for compressing symbolic data such as text or computer files are called *lossless* compression techniques, whereas algorithms for compressing diffuse data like image and video content (when exact content is not important) are called *lossy* compression techniques.

Figure 3.30. Compression Techniques and Algorithms



Lossless Compression

This section deals with methods and standards by which uncompressed data can be exactly recovered from the original data. Medical applications especially need lossless compression techniques; for example, consider digital x-ray images, where information loss may cause diagnostic errors.

The following methods can also be used in combination with other techniques, sometimes even with lossy compression. Some of them we discuss in the section dealing with image and video formats.

Run Length Encoding (RLE)

Consider a data sequence containing many consecutive data elements, here named *characters*. A block of consecutive characters can be replaced by a combination of a byte count and the character itself. For example:

AAAAAAAAAAAAAA (uncompressed)

is replaced with

12A (compressed).

Of course, this is an example in which the data compression by the RLE algorithm is obvious. In the following example, the data compression is significantly lower:

AAAAAABBBCCCCD (uncompressed)

is replaced with

5A3B4C1D (compressed).

The RLE algorithm is very effective in binary images containing large "black" and "white" areas, which have long sequences of consecutive 0s and 1s.

Huffman Coding

The Huffman coding technique^[5] is based on the idea that different symbols of a data source s_i occur with different probabilities p_i , so the *self-information* of a symbol s_i is defined as

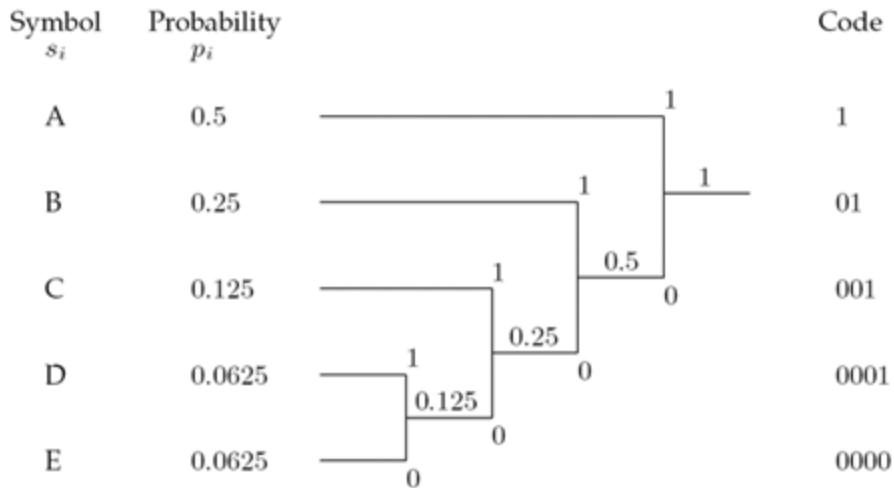
[5] Developed by D. A. Huffman in 1952.

Equation 3.1

$$I(s_i) = \log \frac{1}{p_i} = -\log p_i .$$

If we assign short codes to frequently occurring (more probable) symbols and longer codes to infrequently occurring (less probable) symbols, we obtain a certain degree of compression. [Figure 3.31](#) shows an example.

Figure 3.31. Example of Huffman Coding



The Huffman code is generated in the following four steps:

1. Order the symbols s_i according to their probabilities p_i .
2. Replace the two symbols with the lowest probability p_j by an intermediate node, whose probability is the sum of the probability of the two symbols.
3. Repeat step 2 until only one node with the probability 1 exists.
4. Assign codewords to the symbols according to [Figure 3.31](#).

Obviously, the code can be easily decoded and no prefix for a new code word is needed. For example, we can decode the code sequence

10000001010001

(Huffman coded)

to

AECBD

(decoded)

by simply looking for the 1s and counting the heading 0s. If no 1 appears, then four 0s are grouped to E.

By the way, the principle of the Huffman coding is used in the Morse alphabet; the shortest symbols, the dot (.) and the dash (-) represent the most common letters (at least, the most common letters in Samuel Morse's newspaper) "e" and "t," respectively.

Lempel-Ziv Coding

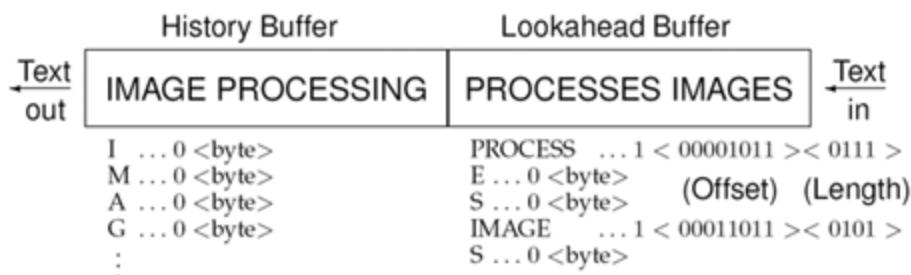
This is another group of compression methods, invented by Abraham Lempel and Jacob Ziv in the late 1970s, consisting of algorithms called LZ77, LZ78, and some derivatives, for example, LZW (from Lempel-Ziv-Welch), and some adaptive and multidimensional versions. Some lossy versions also exist. Here, I describe the principle of LZ77, which is also used in some text-oriented compression programs such as *zip* or *gzip*.

[Figure 3.32](#) shows the principle. The most important part of the algorithm is a sliding window, consisting of a *history buffer* and a *lookahead buffer* [6] that slide over the text characters. The history buffer contains a block of already decoded text, whereas the symbols in the lookahead buffer are encoded from information in the history buffer.[7]

[6] The sizes of the buffers in [Figure 3.32](#) are not realistic. Typically, the history buffer contains some thousand symbols, the lookahead buffer 10–20 symbols.

[7] Therefore, such methods sometimes are called *dictionary compression*.

Figure 3.32. Lempel-Ziv Coding Example (LZ77 Algorithm)



The text in the windows in [Figure 3.32](#) is encoded as follows:

1. Each of the symbols in the history buffer is encoded with 0 <byte>, where 0 indicates that no compression takes place and <byte> is the ASCII character value.
2. If character strings that match elements of the history buffer are found in the lookahead buffer (here: PROCESS and IMAGE), they are encoded with a pointer of the format 1 <offset><length>. For PROCESS, this is 1 <11><7> (decimal).
3. E and S are encoded according to item 1.
4. The resulting pointer for IMAGE is 1 <27><5> (decimal).
5. Finally, S is encoded according to items 1 and 3.

The compression rate in this example is not very high; but if you consider that usually the

history buffer is significantly larger than in our example, it is easy to imagine that compression can be much higher.

Arithmetic Coding

The codes generated in the previous sections may become ineffective if the data contains one or more frequently occurring symbols (that means, with a high probability ρ). For example, a symbol S occurs with the probability $\rho_S = 0.9$. According to Eq. (3.1), the self-information of the symbol (in bits) is

Equation 3.2

$$I(S) = -\log_2 0.9 = -\frac{\log_{10} 0.9}{\log_{10} 2} = 0.152 \quad ,$$

which indicates that only 0.152 bits are needed to encode S . The lowest number of bits a Huffman coder^[8] can assign to this symbol is, of course, 1. The idea behind arithmetic coding is that sequences of symbols are encoded with a single number. [Figure 3.33](#) shows an example.

[8] This is why Huffman coding works optimally for probabilities that are negative powers of 2 (1/2, 1/4, ...).

Figure 3.33. Arithmetic Coding Example

Symbol	Probability	Range
I	0.4	0.0 – 0.4
O	0.2	0.4 – 0.6
S	0.2	0.6 – 0.8
V	0.2	0.8 – 1.0
<hr/>		Total: 1.0
Symbol	Range	Interval Width
		0.0 – 1.0 (Start)
V	0.8 – 1.0	1.0
I	0.0 – 0.4	0.2
S	0.6 – 0.8	0.08
I	0.0 – 0.4	0.016
O	0.4 – 0.6	0.0064

The symbol sequence to be encoded in this example is VISIO.^[9] In a first step, the symbols are ordered (here, alphabetically) and their probability in the sequence is calculated. A symbol range of 0.0–1.0 is then assigned to the whole sequence and, according to their probabilities, to each symbol (top-right corner of [Figure 3.33](#)).

[9] I tried to use VISION, but as you will see, sequences with five symbols are easier to handle.

Then the entire coding process starts with the first symbol, V. The related symbol range is 0.8–1.0; multiplied by the starting interval width of 1.0 a new message interval is generated. The next symbol is I; it has a symbol range of 0.0–0.4. Multiplication by the previous message interval width 0.2 and addition to the previous message interval left border lead to the new message interval 0.80–0.88. This procedure is repeated until all of the symbols have been processed.

The coded result can be any number in the resulting message interval and represents the message exactly, as shown in the decoding example in [Figure 3.34](#). (In this example, I chose 0.85100.)

[Figure 3.34. Arithmetic Decoding Example](#)

(New) Codeword	In Range	Symbol	Probability (Range Interval)
0.85100	0.8 – 1.0	V	0.2
0.25500	0.0 – 0.4	I	0.4
0.63750	0.6 – 0.8	S	0.2
0.18750	0.0 – 0.4	I	0.4
0.46875	0.4 – 0.6	O	0.2

To get the first symbol, the entire code number is checked for the symbol range it is in. In this case, it is the range 0.8–1.0, which corresponds to V. Next, the left (lower) value from this range is subtracted from the codeword, and the result is divided by the symbol probability, leading to

$$\frac{0.85100 - 0.8}{0.2} = 0.25500 \quad ,$$

the new codeword, which is in the range 0.0–0.4 corresponding to I, and so on.

[Table 3.5](#) compares the lossless compression algorithms discussed above.

[Table 3.5. Comparison of Lossless Compression Algorithms](#)

Coding:	RLE	Huffman	LZ77	Arithmetic
<i>Compression ratio:</i>	very low	medium	high	very high
<i>Encoding speed:</i>	very high	low	high	very low
<i>Decoding speed:</i>	very high	medium	high	very low

MH, MR, and MMR Coding

These methods are especially used in images. They are closely related to the techniques already discussed, so we discuss them only briefly here. See [\[28\]](#) [\[29\]](#).

- *Modified Huffman* (MH) coding treats image lines separately and applies an RLE code to each line, which is afterwards processed by the Huffman algorithm. After each line an EOL (end of line) code is inserted for error detection purposes.
- *Modified READ* (MR; READ stands for relative element address designate) coding uses pixel values in a previous line as predictors for the current line and then continues with MH coding.
- *Modified modified READ* (MMR) simply uses the MR coding algorithm without any error mechanism, again increasing the compression rate. MMR coding is especially suitable for noise-free environments.

Lossy Compression

The borders between lossy compression methods and image standards are not clear. Therefore, I focus on the relevant compression standard for still images, JPEG, together with a short description of its main method, discrete cosine transform, and on its successor, JPEG 2000, which uses wavelet transform.

Discrete Cosine Transform (DCT)

Most lossy image compression algorithms use the DCT. The idea is to transform the image into the frequency domain, in which coefficients describe the amount of presence of certain frequencies in horizontal or vertical direction, respectively.

The two-dimensional (2-D) DCT we use here is defined as

Equation 3.3

$$y_{kl} = \frac{c(k)c(l)}{4} \sum_{i=0}^7 \sum_{j=0}^7 x_{ij} \cos\left(\frac{(2i+1)k\pi}{16}\right) \cos\left(\frac{(2j+1)l\pi}{16}\right) ,$$

with $k, l = 0, 1, \dots, 7$ for an 8×8 image area, x_{ij} as the brightness values of this area, and

Equation 3.4

$$c(k) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } k = 0 \\ 1 & \text{if } k \neq 0 \end{cases} .$$

The resulting coefficients y_{kl} give the corresponding weight for each discrete waveform of Eq. (3.3), so the sum of these results is again the (reconstructed) image area.

An important drawback of this calculation is that the effort rises with $(n \cdot \log(n))$, therefore more than linear. This is why the original image is structured in 8×8 pixel areas, called *input blocks*. Figure 3.35 shows this procedure for an image from [Chapter 1](#).

Figure 3.35. 8×8 DCT Coefficients

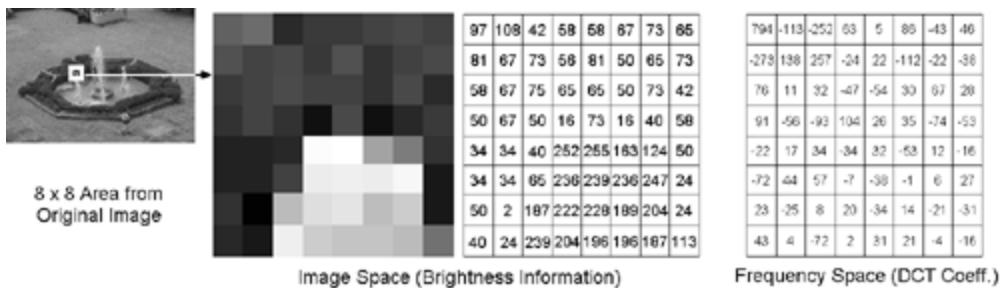


Figure 3.35 also shows the brightness information of the input block (x_{ij}) and the result of the DCT coefficient calculation (y_{kl}) according to Eq. (3.3). This calculation can be done by the VI created in [Exercise 3.3](#).

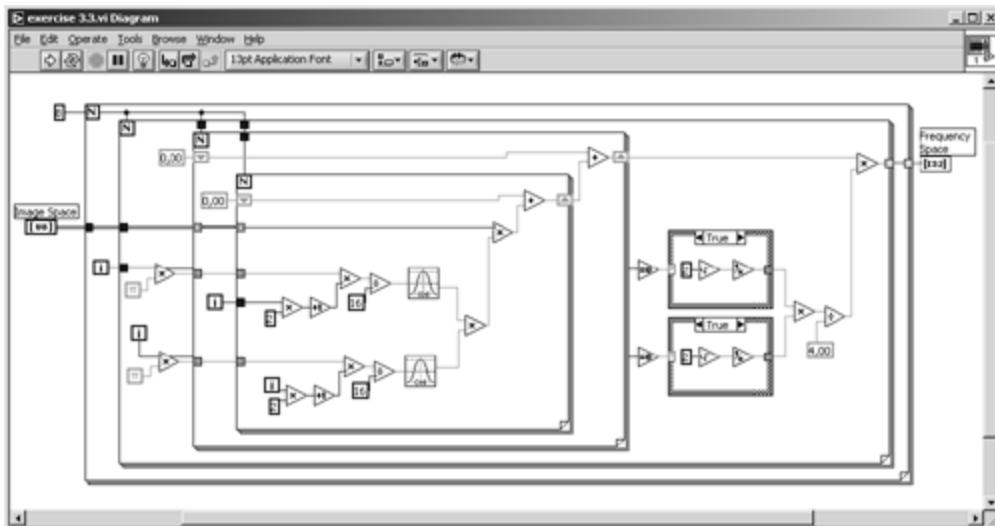
Exercise 3.3: Calculating DCT Coefficients.

Create a LabVIEW VI that calculates the 8×8 DCT coefficients according to Eq. (3.3) and back. (The inverse DCT is described in the following lines.) [Figures 3.36](#) and [3.37](#) show the solution of the first part.

Figure 3.36. Calculating 8×8 DCT Coefficients with LabVIEW



Figure 3.37. Diagram of [Exercise 3.3](#)



The inverse 2-D DCT (8×8) is defined as

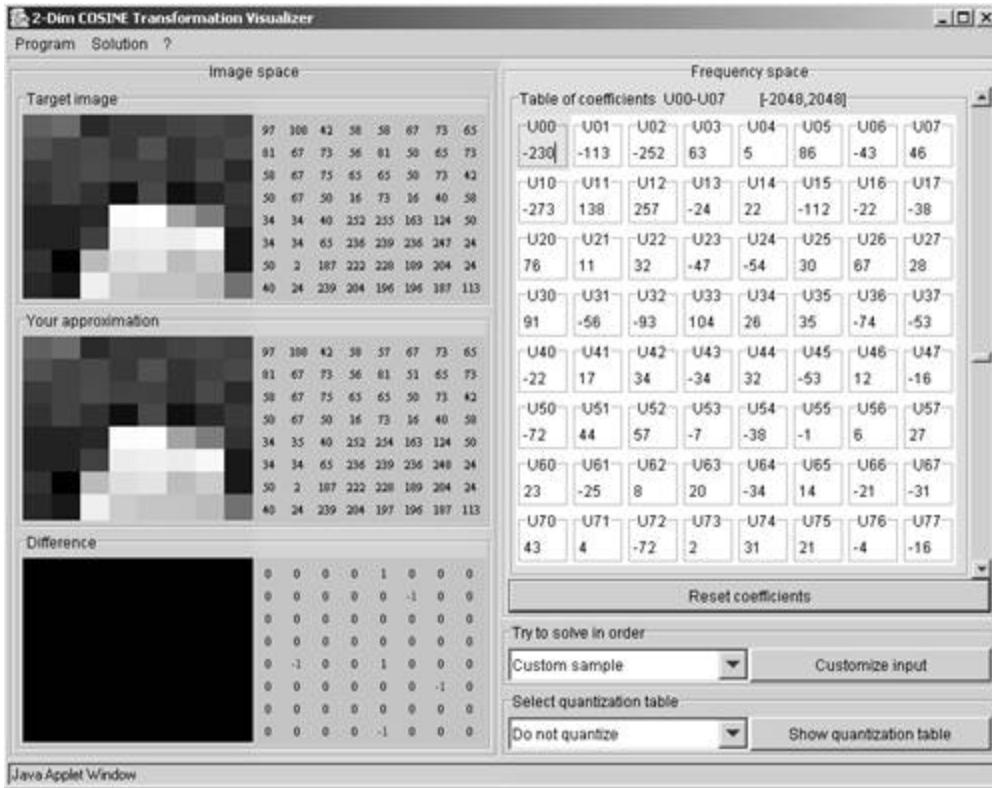
Equation 3.5

$$x_{i,j} = \sum_{k=0}^7 \sum_{l=0}^7 y_{kl} \frac{c(k)c(l)}{4} \cos\left(\frac{(2i+1)k\pi}{16}\right) \cos\left(\frac{(2j+1)l\pi}{16}\right) ,$$

with $k, l = 0, 1, \dots, 7$, resulting in the original image. An important issue is that the formulas for the DCT and the inverse DCT are more or less the same; this makes it easy for hardware implementation solutions to use specific calculation units.

You can easily find tools on the Internet for the calculation of the DCT coefficients and the inverse DCT; a very good one is at www-mm.informatik.uni-mannheim.de/veranstaltungen/animation/multimedia/2d_dct/, shown in [Figure 3.38](#).

[Figure 3.38. DCT and Inverse DCT Calculation](#)



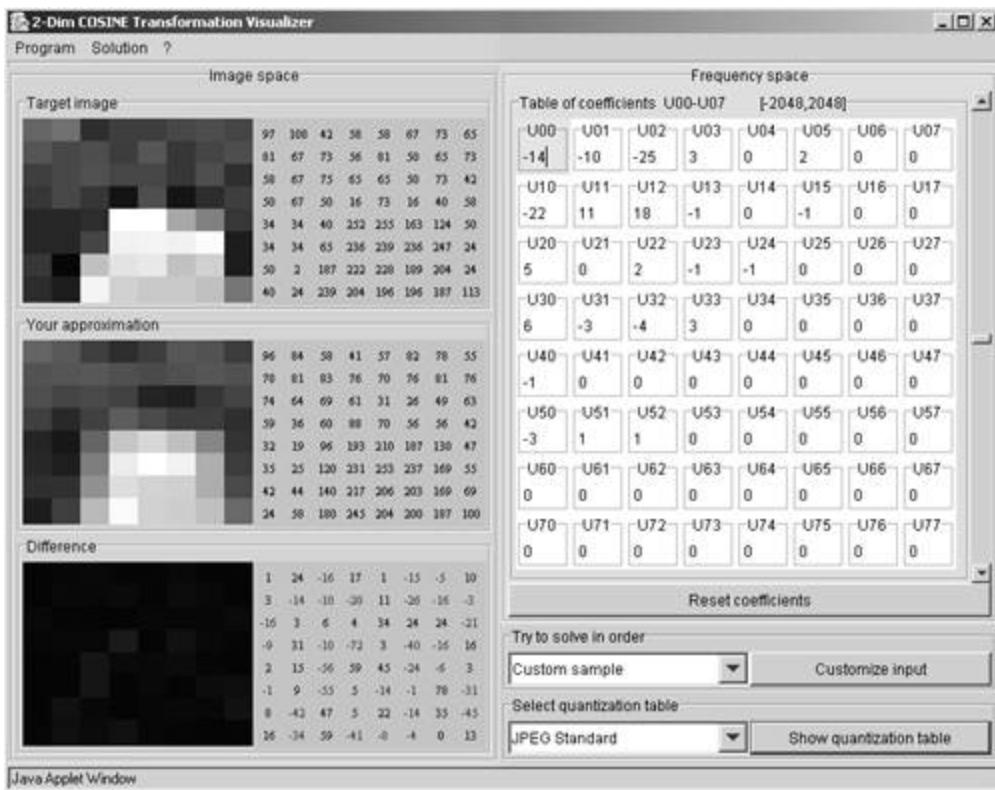
JPEG Coding

You may argue that up to now there has not been any data reduction at all, because the 64 brightness values result in 64 DCT coefficients, and you are right. On the other hand, it is easy to reduce the number of DCT coefficients without losing significant image content.

For example, take a look at the coefficients in [Figure 3.35](#) or [Figure 3.36](#). The most important values, which contain most of the image information, are in the upper-left corner. If coefficients near the lower-right corner are set to 0, a certain degree of compression can be reached. Usually, the coefficient reduction is done by division by a *quantization table*, which is an 8 x 8 array with certain values for each element.

In the 1980s, the JPEG (Joint Photographic Experts Group) started the development of a standard for the compression of colored photographic images, based on the DCT. [Figure 3.39](#) shows the result for the quantization of the DCT coefficients in our example and the reconstructed image.

[Figure 3.39. DCT and Inverse DCT Calculation with JPEG Quantization](#)

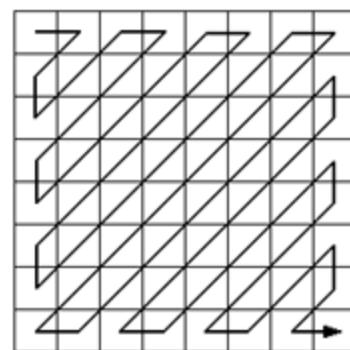


[Figure 3.40](#) shows on its left side the respective JPEG quantization table. The right half of [Figure 3.40](#) shows, that if the resulting coefficients are grouped in a specific order (top left to bottom right), long blocks of consecutive 0s can be found. These blocks are RLE- and Huffman-coded to obtain a compression optimum.

Figure 3.40. JPEG Quantization Table and Coefficient Reading

The used quantize table							
16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

JPEG Quantization Table



JPEG Coefficient Reading

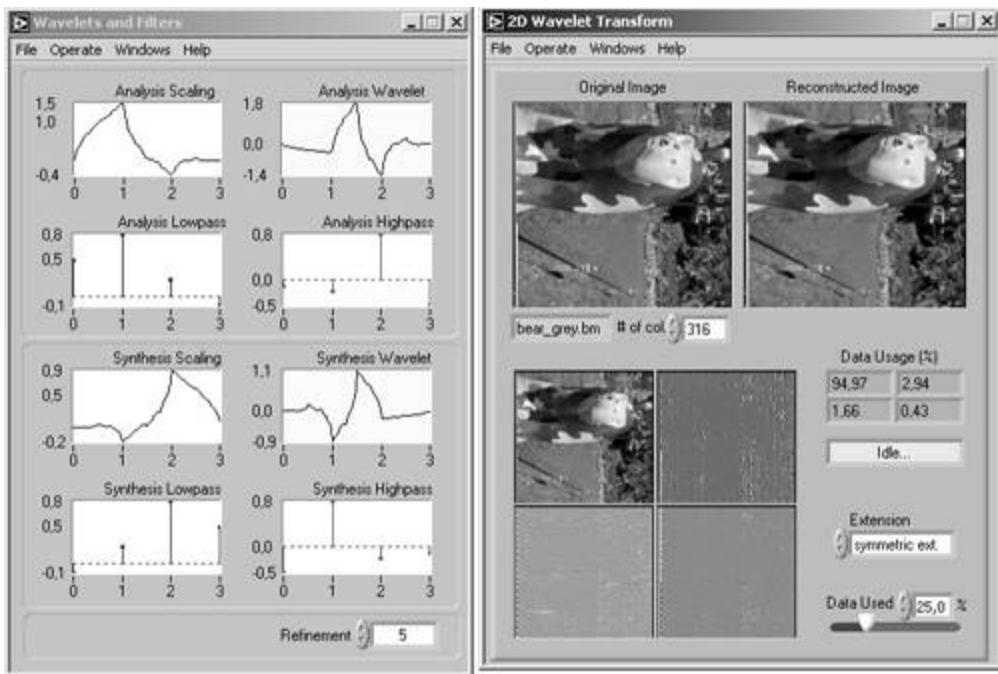
Discrete Wavelet Transform (DWT)

A significant disadvantage of DCT and JPEG compression is that the DCT coefficients describe not only the local image content but also the content of a wider range, because of the continuous cosine functions. The results are the well-known JPEG blocks (8 x 8 pixels, as we already know) surrounded by strong color or brightness borders.

Another possibility is the use of *wavelets* instead of cosine functions. The name *wavelet* comes from "small wave" and means that the signals used in this transformation have only a local impact. [Figure 3.41](#) shows an example from the LabVIEW Signal Processing Toolkit (available

as an add-on).

Figure 3.41. 2D Wavelet Transform Example (LabVIEW Signal Processing Toolkit)



The DWT does the following: it splits the image, by using filters based on wavelets, into two frequency areas—high-pass and low-pass—which each contain half of the information and so, if combined, result in an "image" of the original size. This process is repeated. The equations used for this transformation are

Equation 3.6

$$f_i^{p-1} = \sum_k f_k^p l_{k-2i} ,$$

Equation 3.7

$$w_i^{p-1} = \sum_k f_k^p h_{k-2i} ;$$

where f and w are the filter coefficients and p is an integer giving the number of filter applications. w and f are the results of high-pass and low-pass filter application, respectively. The inverse transform is performed by

Equation 3.8

$$f_i^p = \sum_k (f_k^{p-1} l_{i-2k} + w_k^{p-1} h_{i-2k}) .$$

JPEG2000 Coding

The JPEG2000 group, starting with its work in 1997, developed the new standard for image compression that uses the DWT, resulting in images with the extension **.j2k**. After compression with DWT, JPEG2000 uses arithmetic coding instead of Huffman^[10] for the coefficients. You can compare the results of JPEG and JPEG2000 by using, for example, a tool from www.aware.com; the results are also evident in [Figure 3.42](#).

^[10] Huffman coding is used in JPEG.

Figure 3.42. JPEG2000 Generation Tool (www.aware.com)



In this example, a section of about 35 x 35 pixels is enlarged in [Figure 3.43](#). The 8 x 8 pixel areas in the JPEG version are clearly visible, whereas the JPEG2000 version shows a significantly better result. Both compressed images were generated with a compression ratio of about 60:1.

Figure 3.43. Comparison of JPEG and JPEG2000 Image Areas

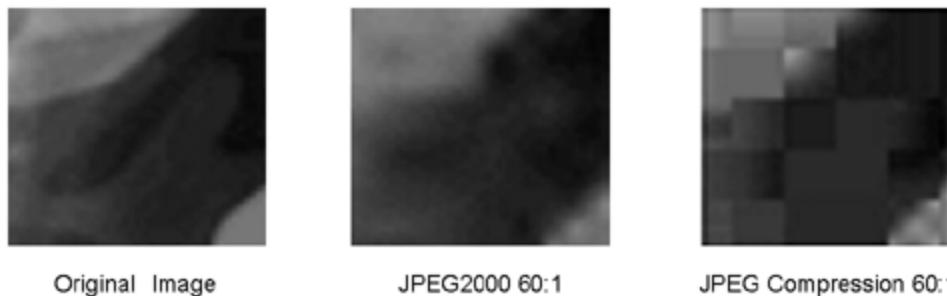


Image Standards

This section discusses the most important image standards, most of them using the previously mentioned compression techniques. Most of them are also file types LabVIEW and IMAQ Vision can handle.

Windows Bitmap Format (BMP)



This device-independent format is primarily used by Windows operating systems. In particular, programs like Paintbrush or Microsoft Paint generate BMP files. The main parts of the BMP format are:

- BITMAP file header;
- BITMAP-INFO
 - BITMAP-INFO header
 - RGB-QUAD;
- BITMAP image data.

The BITMAP file header is structured according to [Table 3.6](#). It is important that the first word of the header contains "BM" in order to indicate a valid BMP file.

Table 3.6. Structure of the BMP File Header

Offset	Bytes	Name	Description
00H	2	<code>bfType</code>	file ID (<code>BM</code>)
02H	4	<code>bfSize</code>	file length (byte)
06H	2		reserved (must be 0)
08H	2		reserved (must be 0)
0AH	4	<code>bfOffs</code>	offset to data area

Following the BITMAP file header, which describes only the file itself, the BITMAP info header gives information about the image data ([Table 3.7](#)). Here, the field `biBitCnt` is of extra importance; it defines the number of colors in the image, using the following values:

- 1: defines 1 bit per pixel (binary image);
- 4: defines an image with 16 colors;
- 8: defines an image with 256 colors;
- 24: defines an image with 16 million colors. The colors are coded directly in the image data area.

Table 3.7. Structure of the BMP Info Header

Offset	Bytes	Name	Description
0EH	4	<code>biSize</code>	length of header (byte)
12H	4	<code>biWidth</code>	width of bitmap in pixels
16H	4	<code>biHeight</code>	height of bitmap in pixels
1AH	2	<code>biPlanes</code>	color planes
1CH	2	<code>biBitCnt</code>	number of bits per pixel
1EH	4	<code>biCompr</code>	compression type
22H	4	<code>biSizeIm</code>	image size (byte)
26H	4	<code>biXPels</code>	horizontal resolution (pixels/m)
2AH	4	<code>biYPels</code>	vertical resolution (pixels/m)
2EH	4	<code>biClrUsed</code>	number of used colors
32H	4	<code>biClrImp</code>	number of important colors

The field `biCompr` defines the type of compression used:

- 0: no compression;
- 1: RLE coding for 8 bit per pixel;
- 2: RLE coding for 4 bit per pixel.

[Table 3.8](#) shows the RGB-QUAD block, which gives additional information about the image color.

Windows NT uses only the three fields from `36H` to `3EH`. Starting with Windows 95, a new format called BMP4 was introduced; it uses additional fields (second part of [Table 3.8](#)).

With BMP4 color spaces other than RGB can be used. One of them is the CIE model (Commission Internationale de l'Eclairage), which is the basis for all of the color models we discussed in [Chapter 2](#).

The following data area contains the image data itself. Usually, the data is not compressed and is stored line by line. The compression options described above are seldom used.

Table 3.8. Structure of the BMP RGB-QUAD Block

Offset	Bytes	Name	Description
36H	4	redMask	mask of red color part
3AH	4	greenMask	mask of green color part
3EH	4	blueMask	mask of blue color part
42H	4	only BMP4	mask of alpha channel
46H	4	only BMP4	color space type
4AH	4	only BMP4	x coordinate red CIE end point
4EH	4	only BMP4	y coordinate red CIE end point
52H	4	only BMP4	z coordinate red CIE end point
56H	4	only BMP4	x coordinate green CIE end point
5AH	4	only BMP4	y coordinate green CIE end point
5EH	4	only BMP4	z coordinate green CIE end point
62H	4	only BMP4	x coordinate blue CIE end point
66H	4	only BMP4	y coordinate blue CIE end point
5EH	4	only BMP4	z coordinate blue CIE end point
62H	4	only BMP4	gamma red coordinate
66H	4	only BMP4	gamma green coordinate
6AH	4	only BMP4	gamma blue coordinate

Graphics Interchange Format (GIF)

This format was developed by CompuServe and first released under the standard GIF87a in 1987 and two years later extended to GIF89a. Interesting features of the GIF format are the capabilities for animations (using multiple images) and for transparency. The typical structure of a GIF file is as follows:

- GIF header block;
- logical screen descriptor block;
- global color map (optional);
- extension blocks (optional);
- image area (repeated for each image):
 - image descriptor block,
 - local color map (optional),
 - extension block (optional),
 - raster data blocks;

- GIF terminator.

A GIF file always starts with the header, using the simple structure shown in [Table 3.9](#). [Table 3.10](#) shows the structure of the subsequent logical screen descriptor block.

Table 3.9. Structure of the GIF Header

Offset	Bytes	Description
00H	3	signature "GIF"
03H	3	version "87a" or "89a"

Table 3.10. Structure of the GIF Logical Screen Descriptor

Bytes	Description
2	logical screen width
2	logical screen height
1	resolution flag →
1	background color index
1	pixel aspect ratio

The diagram shows a bit field with 8 positions, indexed from 0 to 7. Braces group specific bits: {0,1,2} for bits per pixel, {3} for sort flag, {4,5,6} for color resolution, and {7} for the global color table flag.

The content of the logical screen descriptor block is valid for the entire GIF file. Starting from offset 04H, a bit field, called *resolution flag*, describes some definitions: Bit 7 indicates whether a global color map is present. If this bit is true, a global color map follows the logical screen descriptor block.

Bits 4 to 6 define how many bits are used for the color resolution. Bit 3 indicates whether the global color map is sorted (only GIF89a), and bits 0 to 2 define the number of bits per pixel.

The (optional) global color map contains the fundamental color table for all of the subsequent images. It is used if the image itself does not have its own color table. The maximum color resolution for each pixel is 8 bits (256 values).

Table 3.11. Structure of the GIF Image Descriptor Block

Bytes	Description
1	image separator header (ASCII 2CH = ',')
2	coordinate of left margin
2	coordinate of upper margin
2	image width
2	image height
1	flags →

The diagram shows a bit field with 8 positions, indexed from 0 to 7. Braces group specific bits: {0,1,2} for pixel size, {3,4,5} for reserved, {6,7} for sort flag, interlace flag, and local color map flag.

Each image of the GIF file starts with an image descriptor block ([Table 3.11](#)) that contains the

most important image properties. The last byte again contains some flags; one of them is the local color map flag, which specifies whether a local color map block is present and follows the image descriptor block.

The optional extension block, which may follow the local color map, contains some data blocks starting with a byte that defines its length. The entire block starts with a header byte (ASCII **21H** = '!') and ends with a terminator byte (ASCII **00H**).

The image data itself is structured in one or more raster data blocks ([Table 3.12](#)). The first byte defines the minimal code length for the used LZW coding (usually the number of color bits per pixel); the second byte gives the number of bytes in the following data block n .

Table 3.12. Structure of a GIF Raster Data Block

Bytes	Description
1	code size
1	bytes in data block
n	data bytes

The end of a GIF file is defined by a terminator block containing a single byte (ASCII **3BH** = ';) .

Tag Image File Format (TIFF 6.0)



The Tag Image File Format (TIFF; currently available in version 6.0) was defined by a few companies (among them Aldus, Hewlett-Packard, and Microsoft). The fundamental structure of a TIFF file is built by a header ([Table 3.13](#)) and a variable number of data blocks, which are addressed by pointers.

Table 3.13. Structure of the TIFF Header

Offset	Bytes	Description
00H	2	byte order ('II' = Intel, 'MM' = Motorola)
02H	2	version number
04H	4	pointer to first image file directory (IFD)

Another important part of a TIFF file is built by the image file directory (IFD) blocks ([Table 3.14](#)). Their main function is to work as a directory and as a header for the specific data areas. The data block belonging to IFDs is addressed with a pointer.

Table 3.14. TIFF IFD Block Structure

Offset	Bytes	Description
00H	2	number of entries
02H	12	tag 0 (12 bytes)
0EH	12	tag 1 (12 bytes)
...	...	
...	12	tag n (12 bytes)
...	4	pointer to next IFD (0 if last)

The next unit of a TIFF file is a *tag*, which is a 12-byte data structure containing information about the image data. The content of a tag is shown in [Table 3.15](#); it ends with a pointer to the respective image data area. [Table 3.16](#) defines the different data types.

Finally, [Tables 3.17–3.26](#) show the different tag groups. The definition of the tags is beyond the scope of this book; for more details, please read [\[30\]](#).

Table 3.15. TIFF Tag Structure

Offset	Bytes	Description
00H	2	tag type
02H	2	data type
04H	4	length of data area
08H	4	pointer to data area

Table 3.16. Tag Data Types

Code	Type	Remarks
01H	Byte	8-bit byte
02H	ASCII	8-bit ASCII code
03H	SHORT	16-bit (unsigned) integer
04H	LONG	32-bit (unsigned) integer
05H	RATIONAL	2 LONGs: 1: Fraction, 2: Denominator
06H	SBYTE	8-bit (signed) integer
07H	UNDEFINED	8-bit anything
08H	SSHORT	16-bit (signed) integer
09H	SLONG	32-bit (signed) integer
0AH	SRATIONAL	2 SLONGS: 1: Fraction, 2: Denominator

0BH	FLOAT	4-byte single precision IEEE floating point
0CH	DOUBLET	8-byte double precision IEEE floating point

Table 3.17. Image Organization Tags

Code	Tag Group	Data Type	Values
0FEH	New subfile	LONG	1
0FFH	SubfileType	SHORT	1
100H	ImageWidth	SHORT/LONG	1
101H	ImageLength	SHORT/LONG	1
112H	Orientation	SHORT	1
11AH	XResolution	RATIONAL	1
11BH	YResolution	RATIONAL	1
11CH	PlanarConfiguration	SHORT	1
128H	ResolutionUnit	SHORT	1

Table 3.18. Image Pointer Tags

Code	Tag Group	Data Type	Values
111H	StripOffsets	SHORT/LONG	StripPerImage
117H	StripByteCounts	SHORT/LONG	StripPerImage
116H	RowsPerStrip	SHORT/LONG	1
142H	TileWidth	SHORT/LONG	1
143H	TileLength	SHORT/LONG	1
144H	TileOffsets	SHORT/LONG	TilesPerImage
145H	TileByteCounts	SHORT/LONG	TilesPerImage

Table 3.19. Pixel Description Tags

Code	Tag Group	Data Type	Values
102H	BitsPerSample	SHORT	SamplesPerPixel
106H	PhotometricInterpretation	SHORT	1
107H	Thresholding	SHORT	1
108H	CellWidth	SHORT	1
109H	CellLength	SHORT	1
115H	SamplesPerPixel	SHORT	1
118H	MinSampleValue	SHORT	SamplesPerPixel
119H	MaxSampleValue	SHORT	SamplesPerPixel
122H	GrayResponseUnit	SHORT	1
123H	GrayResponseCurve	SHORT	2 ^{BitsPerSample}
12CH	ColorResponseUnit	SHORT	1
12DH	ColorResponseCurves	SHORT	1 or N
131H	Software	ASCII	
132H	DateTime	ASCII	
13BH	Artist	ASCII	
13CH	HostComputer	ASCII	
13DH	Predictor	SHORT	1
13EH	WhitePoint	RATIONAL	2
13FH	PrimaryChromatics	LONG	2 x SamplesPerPixel
140H	ColorMap	SHORT	3 x 2 ^{BitsPerSample}

Table 3.20. Data Orientation Tags

Code	Tag Group	Data Type	Values
10AH	FillOrder	SHORT	1

Table 3.21. Data Compression Tags

Code	Tag Group	Data Type	Values
103H	Compression	SHORT	1
124H	T4Options	SHORT	1
125H	T6Options	SHORT	1
152H	ExtraSamples	BYTE	/
153H	SampleFormat	SHORT	SamplesPerPixel
154H	SMinSampleValue	ANY	SamplesPerPixel
155H	SMaxSampleValue	ANY	SamplesPerPixel
156H	TransferRange	SHORT	6

Table 3.22. Document and Scanner Description Tags

Code	Tag Group	Data Type	Values
10DH	DocumentName	ASCII	
10EH	ImageDescription	ASCII	
10FH	ScannerMake	ASCII	
110H	ScannerModel	ASCII	
11DH	PageName	ASCII	
11EH	XPosition	RATIONAL	
11FH	YPosition	RATIONAL	
129H	PageNumber	SHORT	2
298H	Copyright	ASCII	

Table 3.23. Storage Management Tags

Code	Tag Group	Data Type	Values
120H	FreeOffsets	LONG	
121H	FreeByteCounts	LONG	

Table 3.24. Ink Management Tags

Code	Tag Group	Data Type	Values
14CH	InkSet	SHORT	1
14DH	InkNames	ASCII	
14EH	NumberOfInks	SHORT	1
150H	DotRange	BYTE/SHORT	/n
151H	TargetPrinter	ASCII	

Table 3.25. JPEG Management Tags

Code	Tag Group	Data Type	Values
200H	JPEGProc	SHORT	1
201H	JPEGInterchangeFormat	LONG	1
203H	JPEGInterchangeFormatLength	LONG	1
204H	JPEGRestartInterval	SHORT	1
205H	JPEGLossLessPredictors	SHORT	SamplesPerPixel
206H	JPEGPointTransforms	SHORT	SamplesPerPixel
207H	JPEGQTables	LONG	SamplesPerPixel
208H	JPEGDCTables	LONG	SamplesPerPixel
209H	JPEGACTables	LONG	SamplesPerPixel

Table 3.26. YCbCr Management Tags

Code	Tag Group	Data Type	Values
211H	YCbCrCoefficients	RATIONAL	3
212H	YCbCrSubSampling	SHORT	2
213H	YCbCrPositioning	SHORT	1

The image data of a TIFF file can be uncompressed or compressed with one of the following algorithms:

- *PackBit* compression, which is a proprietary coding technique;
- *Modified Huffman* (MH) coding;
- *LZW* coding by dividing the data into "strips," with a maximum of 8 kbytes and a maximum buffer length of 12 bits;
- *JPEG* coding (since TIFF 6.0). Usually, the TIFF file format is known for its lossless compression techniques; therefore, JPEG does not seem to fit in here. Also, JPEG

compression is not used very often in TIFF files.

Portable Network Graphics Format (PNG)



The PNG file format was originally defined to replace the GIF format because of the patented LZW coding used in GIF. PNG supports images with up to 16 bits per pixel for gray-scale values and up to 48 bits for color values.

The main elements of a PNG file (and also of some other formats) are CHUNKs, which are data blocks of variable length, thus quite similar to tags, with a structure according to [Table 3.27](#).

Table 3.27. CHUNK Structure

Offset	Bytes	Description
00H	4	length of data area (bytes)
04H	4	CHUNK type
08H	n	CHUNK data area
...H	4	CRC value of data area

In general, a PNG file consists of a signature (8 bytes long) and at least three CHUNKs (IDHR, IDAT, and IEND). If the first letter of a CHUNK type is a capital letter, the CHUNK is a critical CHUNK, which must be supported by every program able to read PNG files.[\[11\]](#) All others are ancillary CHUNKs.

[11] The capitalizations of the other letters have specific meanings, but there is no need to discuss them here.

Four critical CHUNKs are possible:

- The *Header CHUNK (IHDR)* ([Table 3.28](#)) contains information about the data stored in the PNG file and immediately follows the PNG signature.
- The *Palette CHUNK (PLTE)* ([Table 3.29](#)) is only used in color images and defines palette values (one for red, green, blue each).
- The *Image Data CHUNK (IDAT)* contains the compressed image data. If the image is large, it is split into more than one IDAT CHUNKs.
- The *Trailer CHUNK (IEND)* is located at the end of the PNG file and does not contain a data area.

Table 3.28. Header (IHDR) CHUNK

Offset	Bytes	Description
00H	4	image width (pixels)
04H	4	image height (pixels)
08H	1	bits per pixel (per sample)
09H	1	color type
0AH	1	compression type
0BH	1	filter type
0CH	1	interlace flag

Table 3.29. Palette (PLTE) CHUNK

Offset	Bytes	Description
00H	3	palette value (1 byte each for red, green, blue)

[Tables 3.30–3.33](#) show examples for ancillary CHUNKs. For more information about them and other possible CHUNKs, please read [\[30\]](#).

Table 3.30. Primary Chromacities and White Point (cHRM) CHUNK

Offset	Bytes	Description
00H	4	x value white point
04H	4	y value white point
08H	4	x value red
0CH	4	y value red
10H	4	x value green
14H	4	y value green
18H	4	x value blue
1CH	4	y value blue

Table 3.31. Physical Pixel Dimension (pHYs) CHUNK

Offset	Bytes	Description
00H	4	pixels per unit (x direction)
04H	4	pixels per unit (y direction)
08H	1	flag: 0 = unit unknown, 1 = unit meter

Table 3.32. Textual Data (tEXt) CHUNK

Offset	Bytes	Description
00H	/	string with key word
...H	1	00H as separator
...H	/	text

Table 3.33. Image Last Modification Time (tIME) CHUNK

Offset	Bytes	Description
00H	2	year
02H	1	month (1 – 12)
03H	1	day (1 – 31)
04H	1	hour (0 – 23)
05H	1	minute (0 – 59)
06H	1	second (0 – 60)

The image data compression is a deflate/inflate compression, based on the LZ77 algorithm using a 32-kbyte sliding window. This algorithm is a license-free technique.

ZSoft Paintbrush File Format (PCX)

This image format is quite old; it was developed by ZSoft and usually used by the imaging software Paintbrush. Because of its great (former) popularity, it is still supported by almost all imaging programs. [Table 3.34](#) shows the structure of a PCX header.

PCX uses the color definition by color planes as shown in [Figure 1.5](#) on page 11. Older PCX versions use only 8 colors (16 with an intensity bit); this use corresponds to the color capabilities of CGA or EGA cards. Starting with version 3.0, PCX supports 256 colors (VGA support).

Table 3.34. PCX File Header

Offset	Bytes	Description
00H	1	identification: 0AH = PCX
01H	1	PCX Version: 0: 2.5, 2: 2.8, 3: 2.8 (w/o palette), 5: 3.0
02H	1	compression flag: 0: no compression, 1: RLE
03H	1	bits per pixel (per plane)
04H	8	coordinates of original image: XMIN, YMIN, XMAX, YMAX
0CH	2	horizontal resolution of a point in dpi (dots per inch)
0EH	2	vertical resolution of a point in dpi (dots per inch)
10H	48	color map with color palette (16 x 3 byte field)
40H	1	reserved
41H	1	number of color planes (max. 4)
42H	2	bytes per image line (even number)
44H	2	palette data: 1: color, 2: gray
46H	58	empty

The image data area itself is either RLE-compressed or uncompressed. The image is divided into its color and intensity information and processed line by line; that means that first the red value of line 1 is compressed and stored, followed by the green value, the blue value, and the intensity value. After that, the processing of line 2 starts.

JPEG/JFIF and JPEG2000 (JPG, J2K)



The JPEG and JPEG2000 standards were already discussed as compression methods. JPEG uses DCT (discrete cosine transform), quantization, and Huffman encoding of the coefficients; JPEG2000 uses DWT (discrete wavelet transform), quantization, and arithmetic coding of the coefficients.

The file format used for JPEG coded files is called JPEG File Interchange Format (JFIF); a JFIF file contains the following:

- *SOI segment* (start of image)
- *APP0 segment* (application marker)
- *Extension APP0 segments* (optional)
- *SOF segment* (start of frame)
- *EOI segment* (end of image)

The entire image data is located in the SOF segments.

A JFIF file may also contain smaller images (called *thumbnails*) for preview use. In this case,

additional (extension) APP0 segments follow the first APP0 segment. A JFIF file always starts with an SOI segment and ends with an EOI segment.

[Tables 3.35, 3.36, and 3.37](#) show the structures of SOI, EOI, and APP0 segments, respectively.

Table 3.35. JFIF SOI Segment

Offset	Bytes	Description
00H	2	SOI signature (FFD8H)

Table 3.36. JFIF EOI Segment

Offset	Bytes	Description
00H	2	EOI signature (FFD9H)

The (optional) extension APP0 segment ([Table 3.38](#)) contains an extension code that defines the structure of the thumbnail image:

- [10H](#): thumbnail JPEG coded;
- [11H](#): thumbnail with 1 byte per pixel;
- [13H](#): thumbnail with 3 bytes per pixel.

Table 3.37. JFIF APP0 Segment

Offset	Bytes	Description
00H	2	APP0 signature (FFE0H)
02H	2	segment length
04H	5	ID "JFIF"
09H	2	version (0102H)
0BH	1	units
0CH	2	x density
0EH	2	y density
10H	1	x thumbnail
11H	1	y thumbnail
12H	3 x n	RGB thumbnail values

Table 3.38. JFIF Extension APP0 Segment

Offset	Bytes	Description
00H	2	APP0 signature (FFE0H)
02H	2	segment length
04H	5	ID "JFXX"
09H	1	extension code
0AH	/	data area

The next tables show some details about the encoding: [Table 3.39](#) describes the DHT segment, which specifies the Huffman table; [Table 3.40](#) shows the DAC segment used for the coding of the color components. Finally, [Table 3.41](#) shows the DQT segment that defines the JPEG quantization table.

Table 3.39. Define Huffman Table (DHT) Segment

Offset	Bytes	Description
00H	2	DHT signature (FFC4H)
02H	2	segment length
04H	1	color component index
05H	16	Huffman table length
15H	/	Huffman table

Table 3.40. Define Arithmetic Coding (DAC) Segment

Offset	Bytes	Description
00H	2	DAC signature (FFCCH)
02H	2	segment length
04H	1	color component index
05H	1	value

Table 3.41. Define Quantization Table (DQT) Segment

Offset	Bytes	Description
00H	2	DQT signature (FFDBH)
02H	2	segment length
04H	1	index
05H	64	quantization table

As mentioned above, the entire data is stored in SOF segments. JFIF allows a number of different SOF segments, each of them specifying a coding algorithm. Until now, only baseline DCT coding has been used.

Comparison of Image Standards

If we compare some of the discussed image standards, we may find file sizes as shown in [Table 3.42](#). All of the files are enclosed on the CD; they were generated with the standard settings of Corel Photo Paint version 8.

Table 3.42. Comparison of Image Standards

	BMP	GIF	TIF	PNG	PCX	JPG	J2K
<i>Compression:</i>	none	none/LZW	none	LZ77	none	DCT	DWT
<i>Monochrome size (kbytes):</i>	76	76	75	67	77	23	2
<i>Color size (kbytes):</i>	223	34	223	207	229	28	6

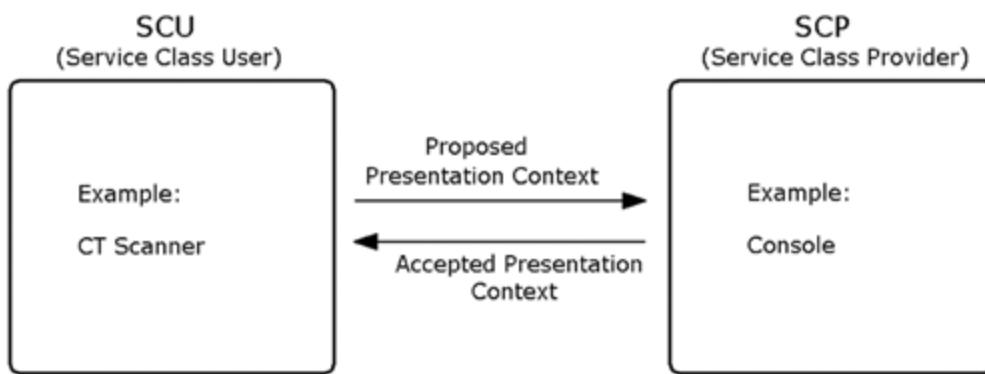
Digital Imaging and Communication in Medicine (DICOM)

DICOM (digital imaging and communication in medicine) is the current standard for imaging formats and data transfer for use in medical applications and devices. In 1994, the ACR-NEMA standard (American College of Radiology, National Electrical Manufacturers Association) was transformed into the DICOM standard; the current version is 3.0.

The DICOM 3.0 Standard

Although sometimes only the imaging standard of DICOM is mentioned, DICOM is actually a network protocol based on a client-server architecture. If two members, which is obviously the minimum, participate on a DICOM communication, the client is called SCU (service class user) and the server SCP (service class provider); [Figure 3.44](#).

Figure 3.44. Simple DICOM Communication



As a first step, the SCU (e.g., a CT scanner) proposes to the SCP a list of services, the SCP (e.g., an imaging console) answers by sending a list of accepted (or possible) services, resulting in an *association negotiation*.

Modalities

Of course, a CT scanner is only one possibility. All methods used for the generation of medical images are called *modalities*. DICOM identifies all modalities by 2-letter acronyms; for example, ultrasound: US; computed tomography: CT; magnetic resonance: MR.

As discussed in [Chapter 2](#), the methods of medical image generation are quite different. DICOM accommodates these different methods and defines the contents and the interpretation of the data.

Conformance

Once a medical equipment manufacturer decides to use DICOM, that manufacturer need not implement the entire standard. In a conformance statement, the manufacturer declares which parts of the standard are incorporated in the product. This makes it easier for the customer to

understand the capabilities of the product and easier for the technician to estimate the communication possibilities with other products.

Compression

In most cases, DICOM images are not compressed in order not to lose any relevant medical information. On the other hand, DICOM provides possibilities for a number of compression techniques; one of them is JPEG. DICOM calls uncompressed images *native* and compressed images *encapsulated*.

DICOM File Format

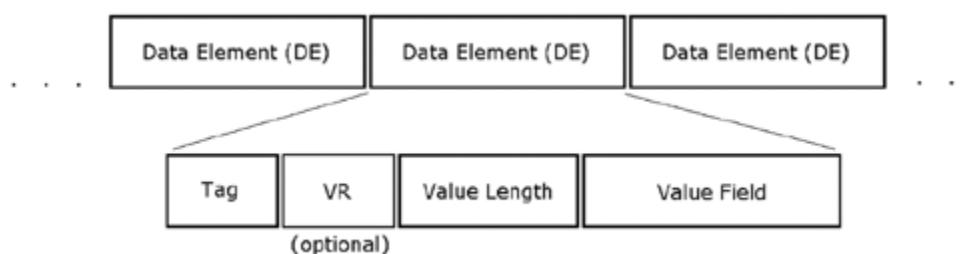
Naturally, for us the DICOM image file format is of major interest. The following sections describe the most important issues.

Data Elements

The most fundamental block of a DICOM file is a data element (DE), which describes certain DICOM file attributes. In general, each DICOM file consists of a number of data elements and nothing else. DEs themselves can be divided into specific components:

- Tag
- Value representation (VR)
- Value length (VL)
- Value field ([Figure 3.45](#))

Figure 3.45. Structure of a DICOM Data Element



Tags

The tag field identifies the type of information that follows in the value field; it consists of two 16-bit numbers: the Group Number and the Element Number. For example, a DICOM image requires the following group numbers:

- 0008: examination data;
- 0010: patient information;
- 0018: image generation data;
- 0020: information of image relationship to examination;

- **0028**: image display data;
- **4000**: overlay information;
- **7FE0**: image data.

[Table 3.43](#) shows an example of the data elements of an image header, sorted by their group and element numbers.

As you can see, DICOM offers a lot of properties for the generation, display, and storage of medical images. The DEs are always sorted in ascending order by their tag numbers.

Value Representations (VRs)

VRs are optional 2-character acronyms for the additional description of the data contained in the DE. For example, the tag (**0010,0010**) has the VR "PN" (patient name).

Because this information is redundant in combination with tags, VRs need not be present. If a data set does not contain VRs, it is referred to as having implicit VRs. An entire data set can only have either VRs or implicit VRs; it is not possible to mix both types.

Value Lengths (VLs)

The VL in the next field is used for two purposes:

- It gives the number of bytes of the following data field.
- It indicates the number of bytes to be skipped in order to jump to the next data element, which is helpful for a quick parsing of the data set.

The subsequent field, called Data Field or Value Field, contains the actual information of the data element.

Table 3.43. Tags of a DICOM Image Header

Group	Element	Group Description	Element Description
0008	0020	examination data	examination date
0008	0030	examination data	examination time
:	:	:	:
0010	0010	patient information	patient name
0010	0030	patient information	date of birth
:	:	:	:
0018	0050	image generation data	slice thickness
:	:	:	:
0020	0020	image relationship	orientation
:	:	:	:

0028	0030	image display data	pixel size
0028	1050	image display data	window center
0028	1051	image display data	window width
:	:	:	:

Nested Data Sets

DICOM can embed data sets within data sets; the unit is called a nested data set or sequence of items (SQ). For example, a DE for modality LuT (look-up table) sequence has the tag (0028, 3000) and contains a set of data elements as its data value, all together forming the modality LuT.

Value Multiplicity (VM)

DE could contain more than one data field. This case is expressed in a VM. The value of a VM can be one of three types:

- a specific number (example: zoom factor (0028,0032));
- a maximum number;
- an unlimited number of values (example: other study numbers (0020,1070)).

DICOM Image Storing

The pixel data of a DICOM image is stored in a DE and has a tag called Pixel Data (7FE0,0010). It is important to know that DICOM images can store 8 to 16 bits per pixel, compared to the usual 8 bits used in regular file formats. Therefore, an image with 8 bits per pixel stores the data in a single byte per pixel; images with 9 to 16 bits need 2 bytes per pixel. The following three data elements specify how the data is stored:

- *Bits allocated* (0028,0100): the number of bits for each pixel, which can be only 8 or 16.
- *Bits stored* (0028,0101): the number of bits containing meaningful image data; for example, 12.
- *High bit* (0028,0102): gives the position of the most significant bit (MSB); for example, if BS = BA, then HB = BA - 1.

The Samples per Pixel DE (0028,0002) defines the number of color channels per pixel. Gray-scale images have a value of 1, RGB images of 3.

The Photometric Interpretation DE (0028,0004) stores information about how the pixel data is to be interpreted. The respective code string can be one of the following:

- MONOCHROME1: gray-scale image with black as the maximum value and white as the minimum.
- MONOCHROME2: gray-scale image with white as the maximum value and black as the minimum.
- PALETTE COLOR: color image using a color LuT (samples per pixel = 1). The respective tags are:

- (0028,1201): red,
 - (0028,1202): green,
 - (0028,1203): blue.
- **RGB**: color image; samples per pixel = 3.
- **HSV**: color image; samples per pixel = 3.
- **ARGB**: RGB image with an additional alpha channel containing more image information or overlays; samples per pixel = 4.
- **CMYK**: color image; samples per pixel = 4.
- **YBR_FULL**: color image using YCbCr channels; samples per pixel = 3.
- **YBR_FULL_422**: same as above with 4:2:2 color sampling.
- **YBR_PARTIAL_422**: same as above with Y restricted to 220 levels, C_b and C_r restricted to 225 levels.

The Planar Configuration DE (0028,0006) specifies the arrangement of the pixels in a color image:

- **000**: pixel by pixel;
- **001**: plane by plane.

Most of this information was taken from the *AccuSoft ImageGear Medical Plug-In, 2001 User's Guide*, which comes with the respective software.

DICOM Functions in LabVIEW

To use DICOM functions in LabVIEW, you have to install a DICOM SDK, which may be available for free or as evaluation versions from the Internet. I used AccuSoft (www.accusoft.com) DICOM Communication SDK in my exercises; it provides a SDK based on DLL32 and OCX32 as well.

Before you can use the ActiveX controls coming with the SDK, you must import them into LabVIEW. Select Tools/Advanced/Import ActiveX Controls... to open a window that lists all of the ActiveX controls registered on your system (see [Figure 3.46](#)). Here, look for DcSdk Control, press OK, and save the control under a specific name. Afterwards, the control appears in the Controls Palette under Custom Controls ([Figure 3.47](#)).

Figure 3.46. ActiveX Control Import List

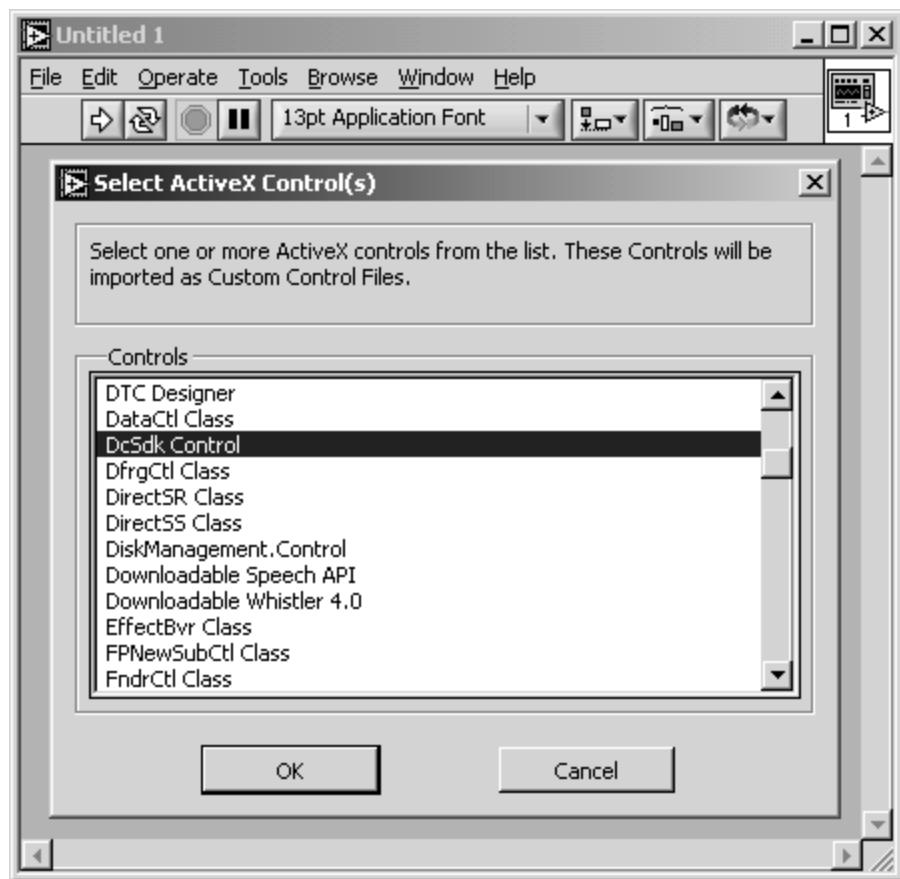
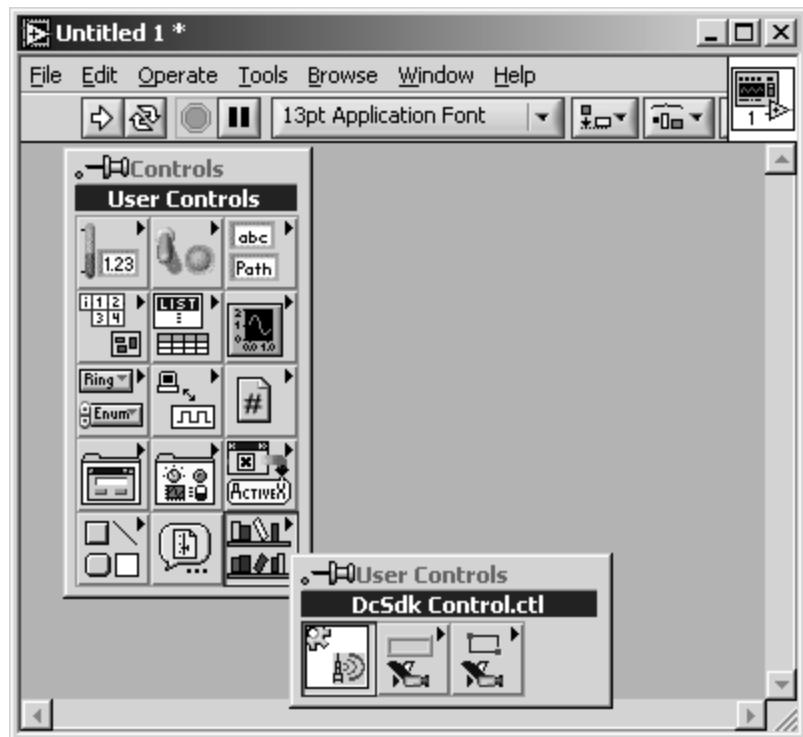


Figure 3.47. Imported ActiveX Control

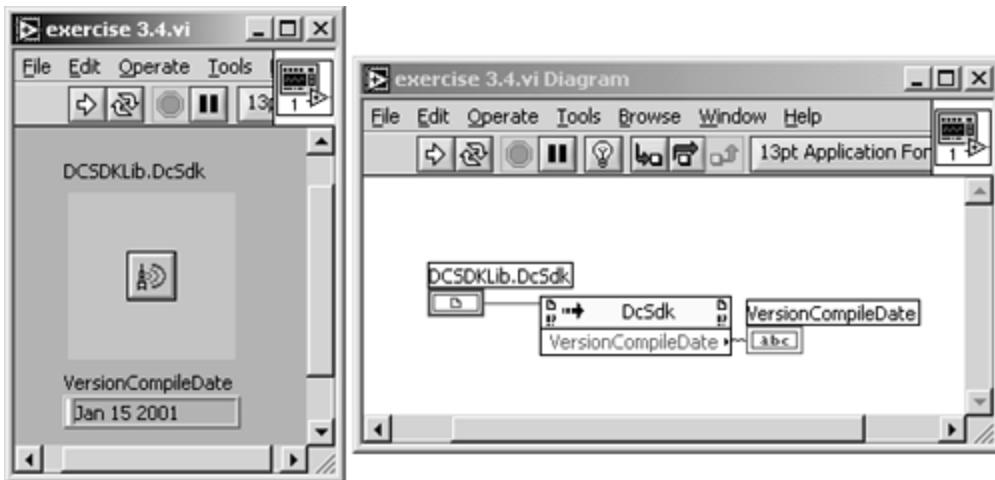


Exercise 3.4: DICOM Communication.

Verify the importation of the AccuSoft DICOM Communication SDK by using an

Invoke Node and select the property VersionCompileDate from the pop-up menu. My version displays "Jan 15 2001" (see [Figure 3.48](#) for further details).

Figure 3.48. Verification of the Correct Import of Accusoft DICOM Comm SDK in LabVIEW



Other tools let you work with DICOM images in LabVIEW and IMAQ Vision; unfortunately, they are not available for free. In the following exercise I worked with the Medical Imaging Toolkit, again to be found at www.accusoft.com. An evaluation version of this software works for five days.

After the toolkit is installed, two additional ActiveX controls must be imported into LabVIEW:

- ImageGear ActiveX (ImageGear is a basic toolkit for image processing by AccuSoft) and
- MEDXCtl Class.

With these two additional controls, you can do the following exercise.

Exercise 3.5: Importing DICOM Images (additional software required).

Using Image Gear and Medical Imaging ActiveX controls (or similar controls), build a VI that imports DICOM images into LabVIEW and IMAQ Vision. If the import is successful, all of the image processing and analysis functions of IMAQ Vision can be applied to DICOM images.

[Figure 3.49](#) shows a possible solution. The image is transferred via the clipboard to an IMAQ window; to get the image there, you need both imported ActiveX controls. First of all, ImageGear needs to know that you intend to use the evaluation version of the Medical Imaging Toolkit; on the other hand, the Medical Imaging Toolkit requires the name of the ImageGear software (frame 0 of the sequence in [Figure 3.50](#)).

As a second step, load the image into ImageGear and copy it onto the clipboard (frame 1). Be careful, and verify the correct implementation of the Medical Imaging Toolkit (frame 0); otherwise, ImageGear will not load DICOM images.

Figure 3.49. Loading DICOM Images into LabVIEW and IMAQ Vision

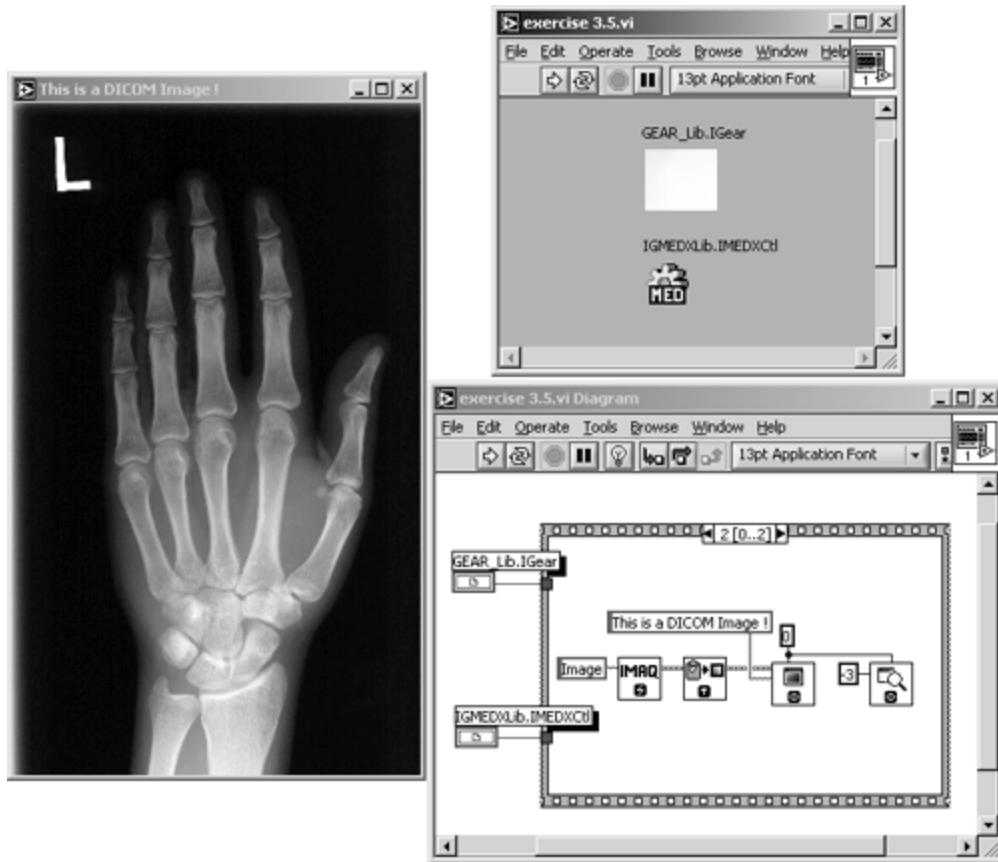
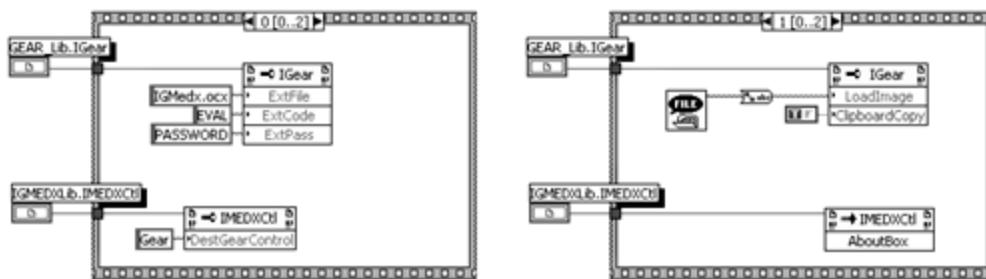


Figure 3.50. Frames 0 and 1 of [Exercise 3.5](#)



The property nodes of frame 0 in [Exercise 3.5](#) can be replaced by the following Visual Basic code in other applications:

```

Gear.ExtFile = "IGMedx.ocx"
Gear.ExtCode = "EVAL"
Gear.ExtPass = "PASSWORD"

MED1.DestGearControl = "Gear"

Gear.LoadImage = "filename.dcm"
Gear.ClipboardCopy = TRUE

```

Chapter 4. Image Processing

Our images have now found their way from the generation device over transport or storage media to the image processing and analysis software, in our case, IMAQ Vision. As defined in [Chapter 1](#), the output of an image *processing* operation is still another image; this chapter, therefore, focuses on methods like image manipulation using *look-up tables*, *filtering*, *morphology*, and *segmentation*.

Here and in [Chapter 5](#), I tried to provide exercises and examples without any additional hardware and software. When that is not possible, I specify exactly the type of hardware equipment and where to get additional software. Moreover, I keep as close as possible to the IMAQ Vision User Manual [\[4\]](#), which means that I use the same symbols and equations but also give more information and examples.

Gray-Scale Operations

The easiest way, of course, to alter a digital image is to apply changes to its (usually 8-bit, from 0 to 255) gray-level values. Before we do this, it might be interesting to learn how the gray-level values of an image are distributed.

Histogram and Histogram

IMAQ Vision provides two simple tools for this distribution; they do almost the same thing:

- The *histogram* gives the numeric (quantitative) information about the distribution of the number of pixels per gray-level value.
- The *histograph* displays the histogram information in a waveform graph.

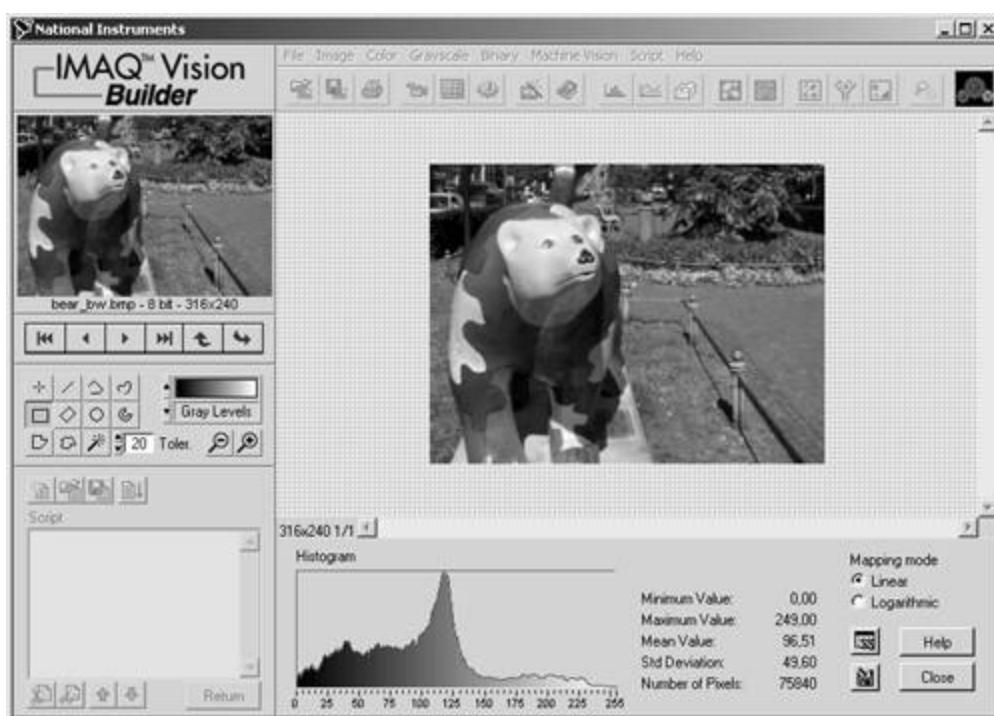
According to [4], the histogram function $H(k)$ is simply defined as

Equation 4.1

$$H(k) = n_k \quad ;$$

with n_k as the number of pixels with the gray-level value k . Figure 4.1 shows the histogram function $H(k)$ for our bear image using IMAQ Vision Builder. In addition, you can try the following exercise.

Figure 4.1. Histogram Function in IMAQ Vision Builder



Exercise 4.1: Histogram and Histogram.

Create a LabVIEW VI that provides the functionality described above directly in IMAQ Vision, using the functions **IMAQ Histogram** and **IMAQ Histogram**. You can find them under Motion and Vision / Image Processing / Analysis.[\[1\]](#)[Figure 4.2](#) shows a possible solution.

[1] Actually, the histogram is an image analysis function according to our definition. On the other hand, we need its results for the following sections, so I explain it here.

From IMAQ Vision Builder, the histogram data can be exported for further processing into MS Excel; [Figure 4.3](#) shows the data from [Figure 4.1](#) and [4.2](#) in an MS Excel diagram.

Figure 4.2. Histogram and Histogram of an Image

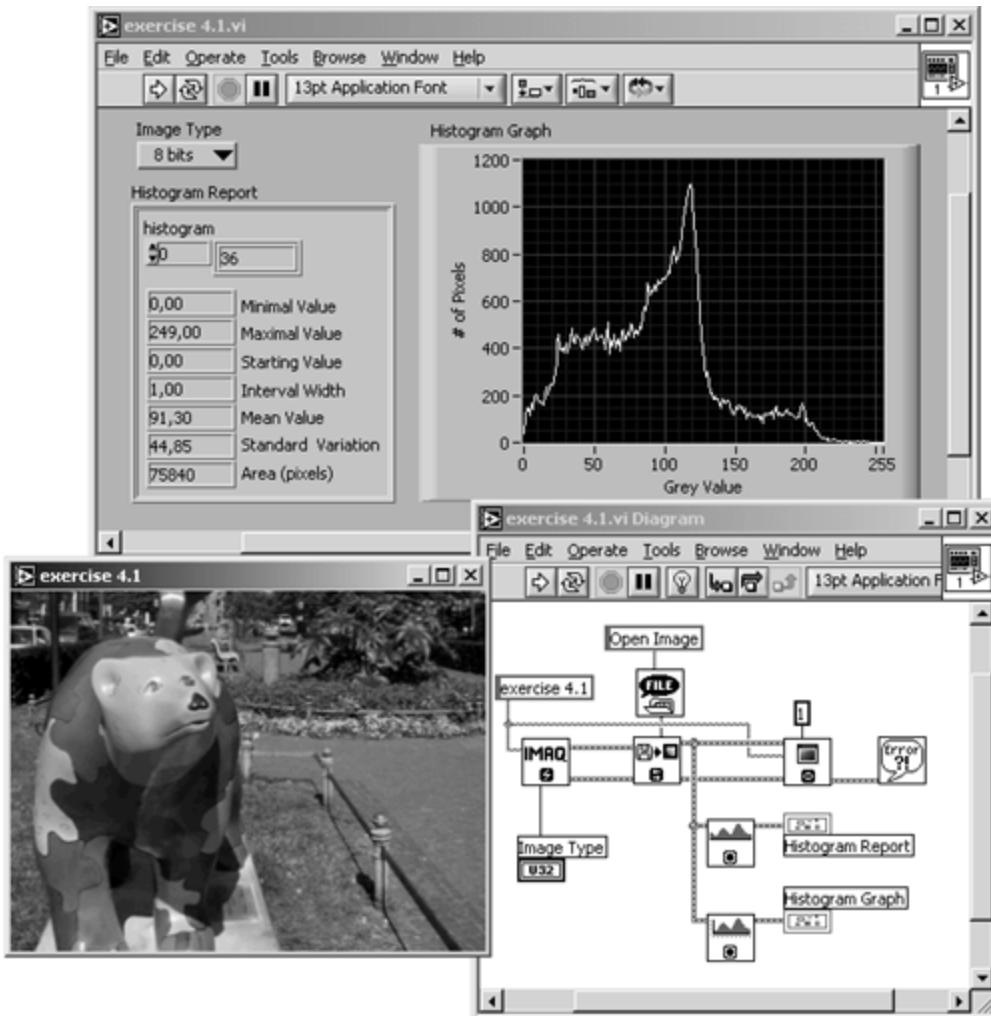
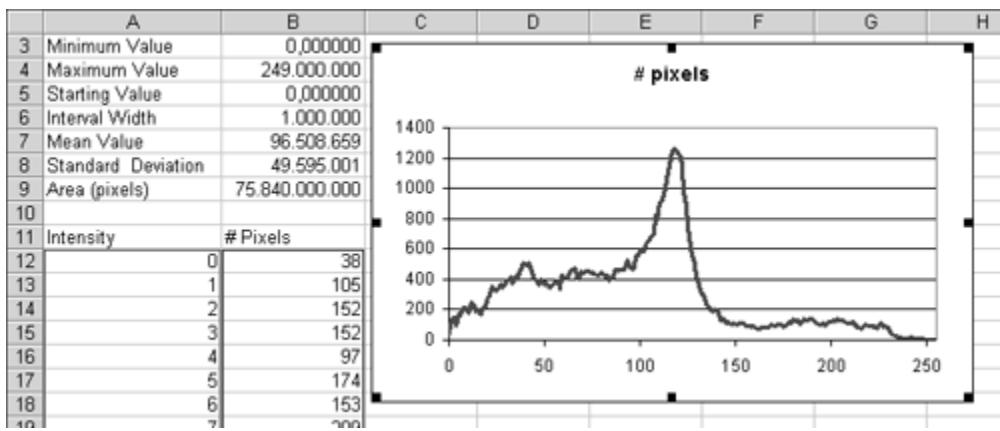
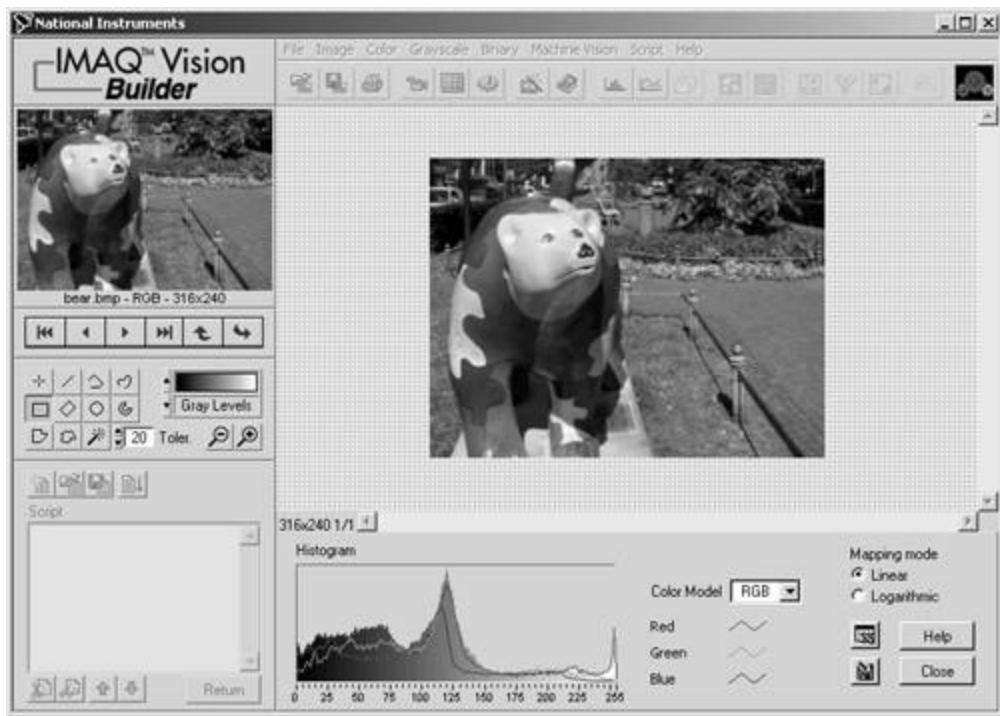


Figure 4.3. Histogram Exported in MS Excel



If the histogram function is applied to a color image, it returns values $H(\lambda)$ for each color plane; for example, three histograms for red, green, and blue in the case of an RGB image. [Figure 4.4](#) shows an example using IMAQ Vision Builder.

Figure 4.4. Color Histogram Function in IMAQ Vision Builder



Using Look-up Tables (LuT)

You know how to adjust brightness and contrast values of your TV set or TV video PC card; the intention is to get a "better" image, what obviously means a better distribution of the gray-level values. This is achieved by a function $\mathcal{A}(g)$ (g are the gray-level values), which assigns a new value to each pixel.

If we consider Eq. (1.1), then our function $\mathcal{A}(g)$ will modify the gray-level values according to

Equation 4.2

$$s_{\text{out}}(x, y) = f(s_{\text{in}}(x, y)) \quad ,$$

with s_{in} as the original values and s_{out} as the resulting values. Because the possible results are limited to 256 values, the function $\mathcal{A}(g)$ usually is realized by a table consisting of 256 values, a *look-up table* (LuT). Therefore, if

Equation 4.3

$$LuT(g) = f(g) \quad ,$$

Eq. (4.2) will change to

Equation 4.4

$$s_{out}(x, y) = LuT(s_{in}(x, y)) \quad .$$

Special LuTs: Math Functions

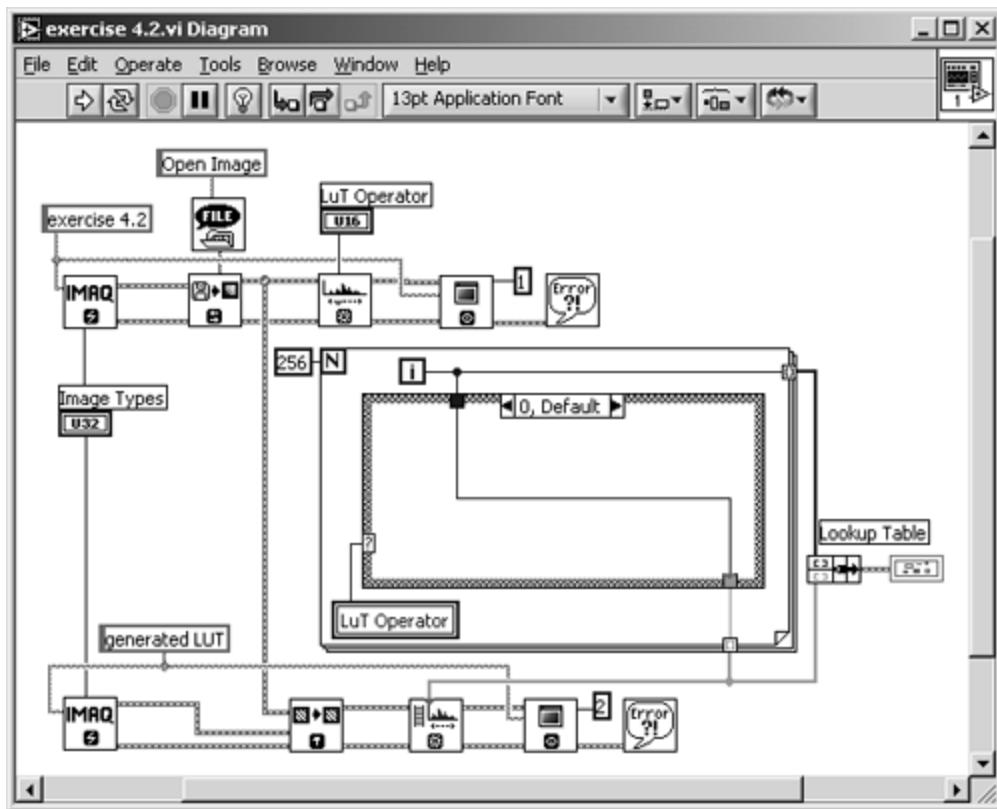
The easiest way to generate a LuT is to use mathematic functions, such as logarithmic, exponential, and power. We try this in the following exercise.

Exercise 4.2: Look-up Tables.

In this and the following exercise we compare the fixed IMAQ Vision functions to the impact of manually generated LuTs. First of all, create the VI shown in [Figure 4.5](#).

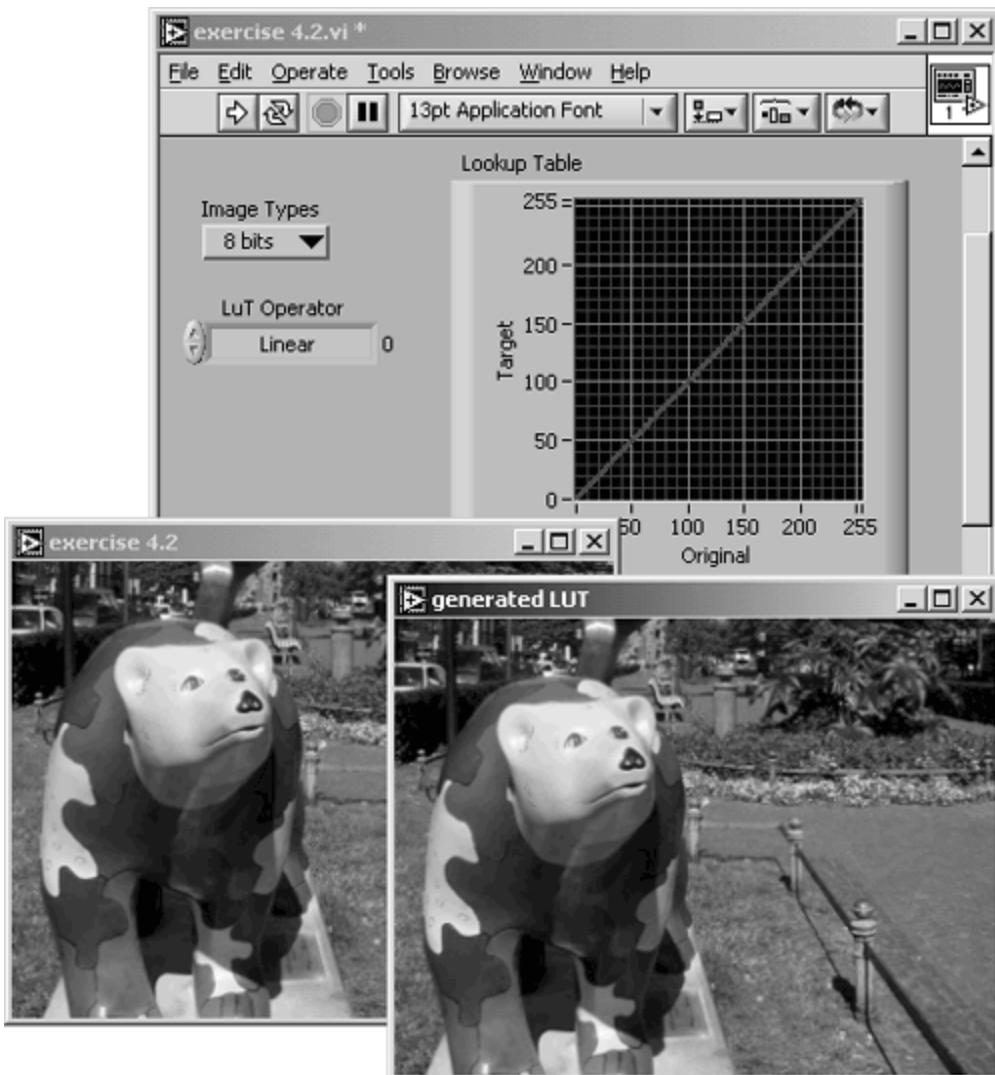
Use the function `IMAQ UserLookup` to generate the second image, which is used for the comparison of the result to the built-in IMAQ function; here, the function `IMAQ MathLookup`. An XY graph displays the entire LuT by a simple line.

Figure 4.5. [Exercise 4.2](#): Creating User LuTs



[Figure 4.6](#) shows the front panel of [Exercise 4.2](#). Case 0 of the VI leaves the gray-level values of the original image unchanged, which means that the location 0 of the LuT gets the value 0, 1 gets 1, and so on until the location 255, which gets 255. The XY graph shows a line from bottom left to top right.

Figure 4.6. Processing Look-up Tables (LuTs)



Another built-in LuT is a logarithmic one; if the function $f(g)$ were $f(g) = \log(g)$, then it would be sufficient to add the logarithm base 10 function of LabVIEW to case 1 and watch the results.

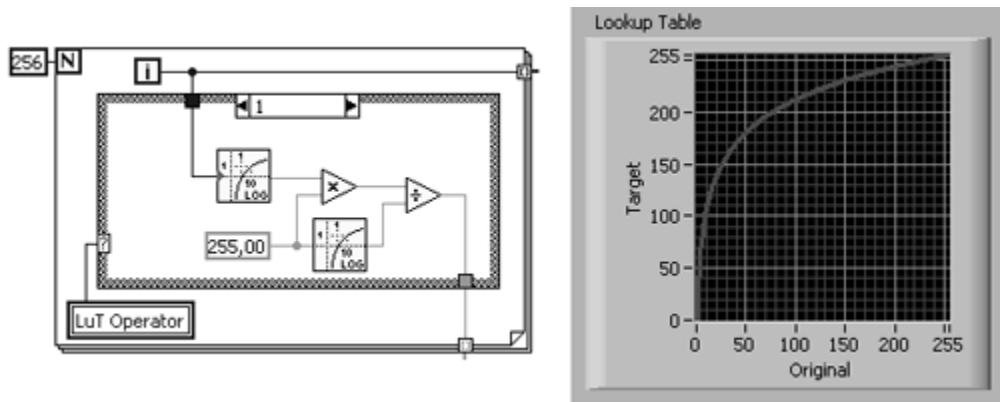
Oops—that did not work. Naturally, the logarithm (base 10) of 255 is about 2.4, so we have to scale the entire range of resulting values to the 8-bit gray-scale set. The resulting function is

Equation 4.5

$$s_{\text{out}}(x, y) = \log(s_{\text{in}}(x, y)) \cdot \frac{255}{\log(255)} = \log(s_{\text{in}}(x, y)) \cdot 105.96$$

and is shown in [Figure 4.7](#).

Figure 4.7. Creating a Logarithmic Look-up Table



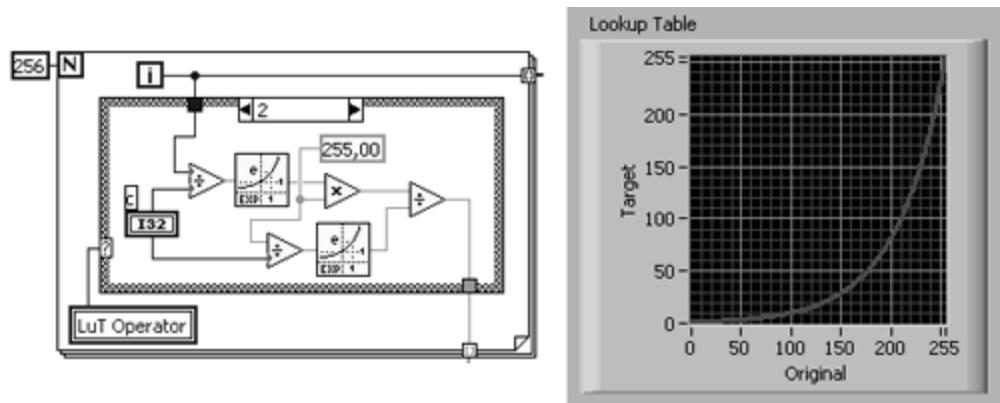
The next function is exponential. If we generate it the way we did above, we see a far too steep function because the values rise too fast. A correction factor c , according to

Equation 4.6

$$s_{\text{out}}(x, y) = \exp(s_{\text{in}}(x, y)/c) \cdot \frac{255}{\exp(255/c)} ,$$

leads to more reasonable results. $c \approx 48$ is the value used in the built-in IMAQ function (see the realization of this equation in [Figure 4.8](#)).

Figure 4.8. Creating an Exponential Look-up Table



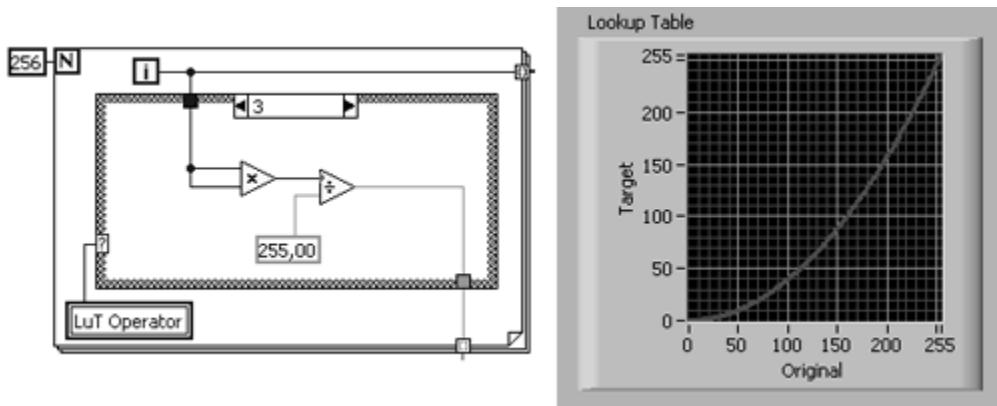
The square function is easy: If we multiply the value location by itself and scale it like we did above, we get

Equation 4.7

$$s_{\text{out}}(x, y) = (s_{\text{in}}(x, y))^2 \cdot \frac{255}{255^2} = (s_{\text{in}}(x, y))^2 \cdot \frac{1}{255} .$$

[Figure 4.9](#) shows the corresponding results.

Figure 4.9. Creating a Square Look-up Table

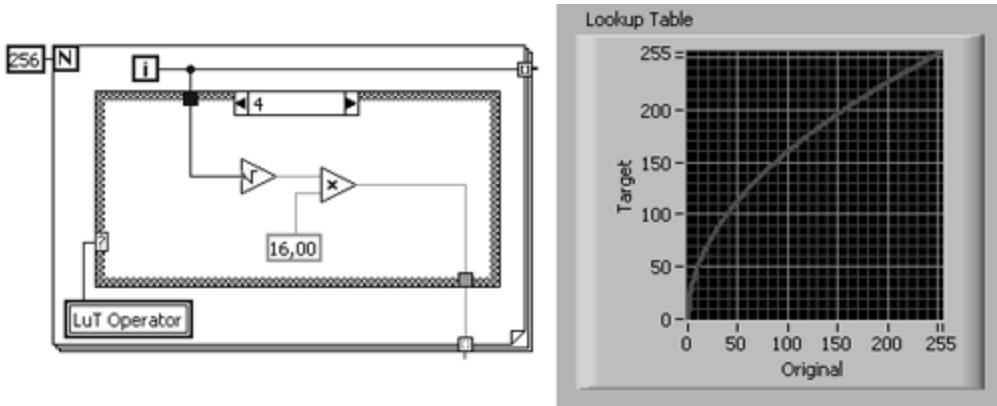


Square root is quite similar. [Figure 4.10](#) shows the results, and here is the function:

Equation 4.8

$$s_{\text{out}}(x, y) = \sqrt{(s_{\text{in}}(x, y))} \cdot \frac{255}{\sqrt{255}} \approx \sqrt{(s_{\text{in}}(x, y))} \cdot 16 \quad .$$

Figure 4.10. Creating a Square Root Look-up Table



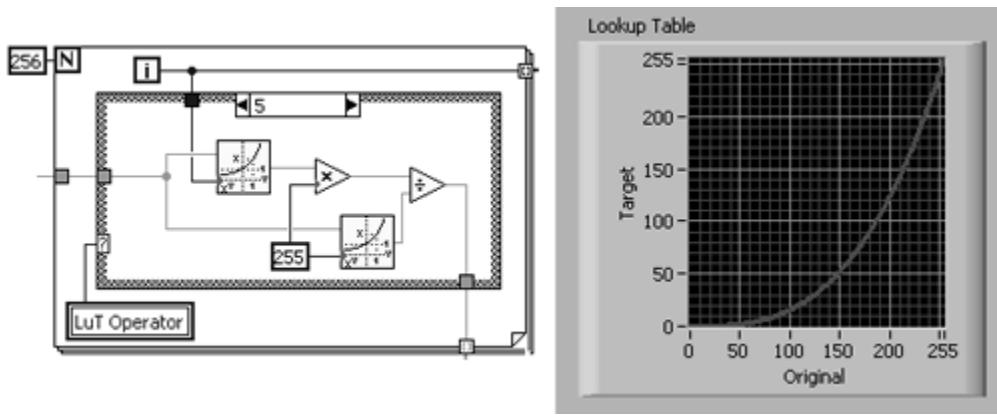
The next two cases need an additional factor called power value ρ : We create the function

Equation 4.9

$$s_{\text{out}}(x, y) = (s_{\text{in}}(x, y))^{\rho} \cdot \frac{255}{255^{\rho}} = (s_{\text{in}}(x, y))^{\rho} \cdot \frac{1}{255^{\rho-1}} \quad .$$

IMAQ Vision calls this function power x , its realization can be seen in [Figure 4.11](#) for a power value of $\rho = 3$.

Figure 4.11. Creating a Power x Look-up Table



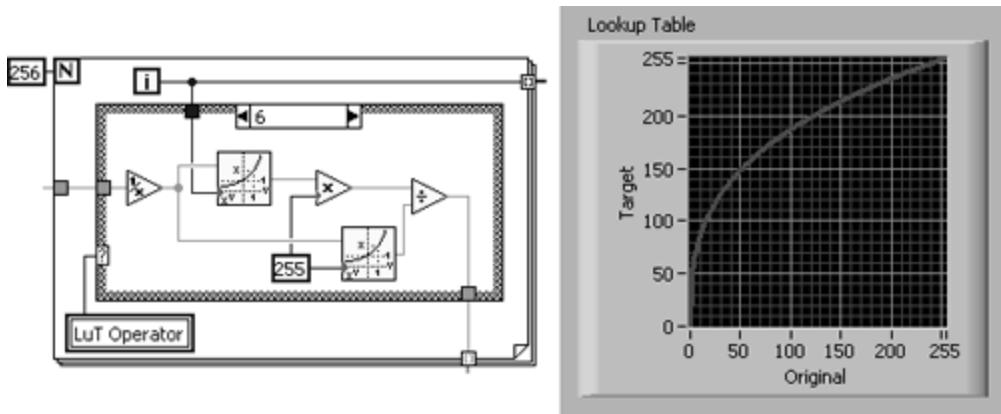
The last function used in this exercise is called power $1/x$ and is simply realized by calculating the reciprocal value of ρ , use it like the power x function:

Equation 4.10

$$s_{\text{out}}(x, y) = (s_{\text{in}}(x, y))^{\frac{1}{p}} \cdot \frac{255^{\frac{1}{p}}}{255^p} .$$

[Figure 4.12](#) shows the implementation of the power $1/x$ function in LabVIEW and IMAQ Vision.

Figure 4.12. Creating a Power $1/x$ Look-up Table



It is easy to see that these functions can be divided into two groups: Functions like logarithmic, square root, and power $1/x$ compress regions with high gray-level values and expand regions with low gray-level values. The exponential, square, and power x functions operate conversely; they expand regions with high gray-level values and compress regions with low gray-level values.

The power value ρ is often called gamma coefficient; it indicates the degree of impact of the power function. We return to it in [Exercise 4.5](#).

Special LuTs: Equalize and Inverse

IMAQ Vision also contains a function called [IMAQ Equalize](#), which distributes the gray-level values evenly over the entire gray-scale set (usually 0 to 255 for 8 bits). If an image does not use all available gray levels (most images do not), the contrast (the difference from one gray level to another) of the image can be increased.

Exercise 4.3: Equalizing Images.

Create a VI that equalizes our bear image. In this case, it is not so easy to create a user defined LuT, so we concentrate on the histograms. Both of them can be seen in [Figure 4.13](#); [Figure 4.14](#) shows the diagram.

The histogram of the equalized image in [Figure 4.13](#) is quite interesting. It is obviously not a solid line like the histogram of the original image. This is because the equalize function distributes an equal amount of pixels over a constant gray-level interval. If not all gray-level values are used in the original image, the histogram of the resulting image contains values without pixels.

Figure 4.13. Image and Histogram Resulting from Equalizing

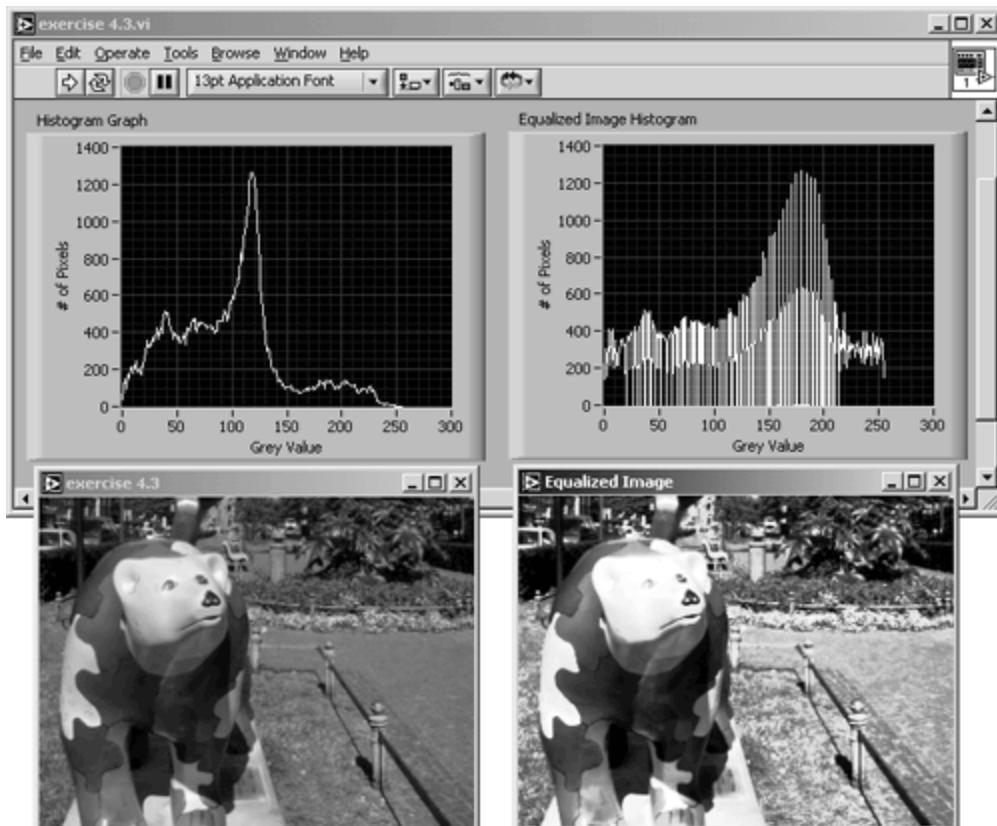
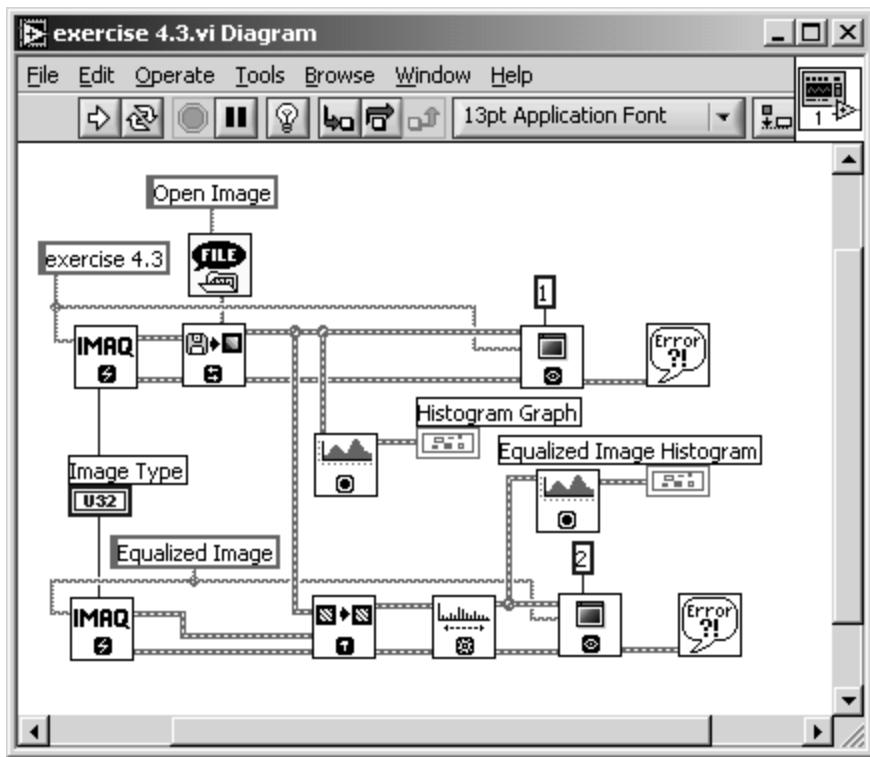


Figure 4.14. Diagram of [Exercise 4.3](#)



The next function is very simple: inverse. High gray-level values are replaced by low gray-level values, and vice versa: 0 becomes 255, 1 becomes 254, and so forth until 255, which becomes 0. This procedure can be described by

Equation 4.11

$$s_{\text{out}}(x, y) = 255 - s_{\text{in}}(x, y) \quad .$$

Exercise 4.4: Manual Creation of LuTs.

In this exercise we create the LuT manually by simply subtracting the actual value from 255, as in Eq. (4.11). The resulting images and the LuT are shown in [Figure 4.15](#); the diagram is shown in [Figure 4.16](#).

Figure 4.15. Inverting the Bear Image

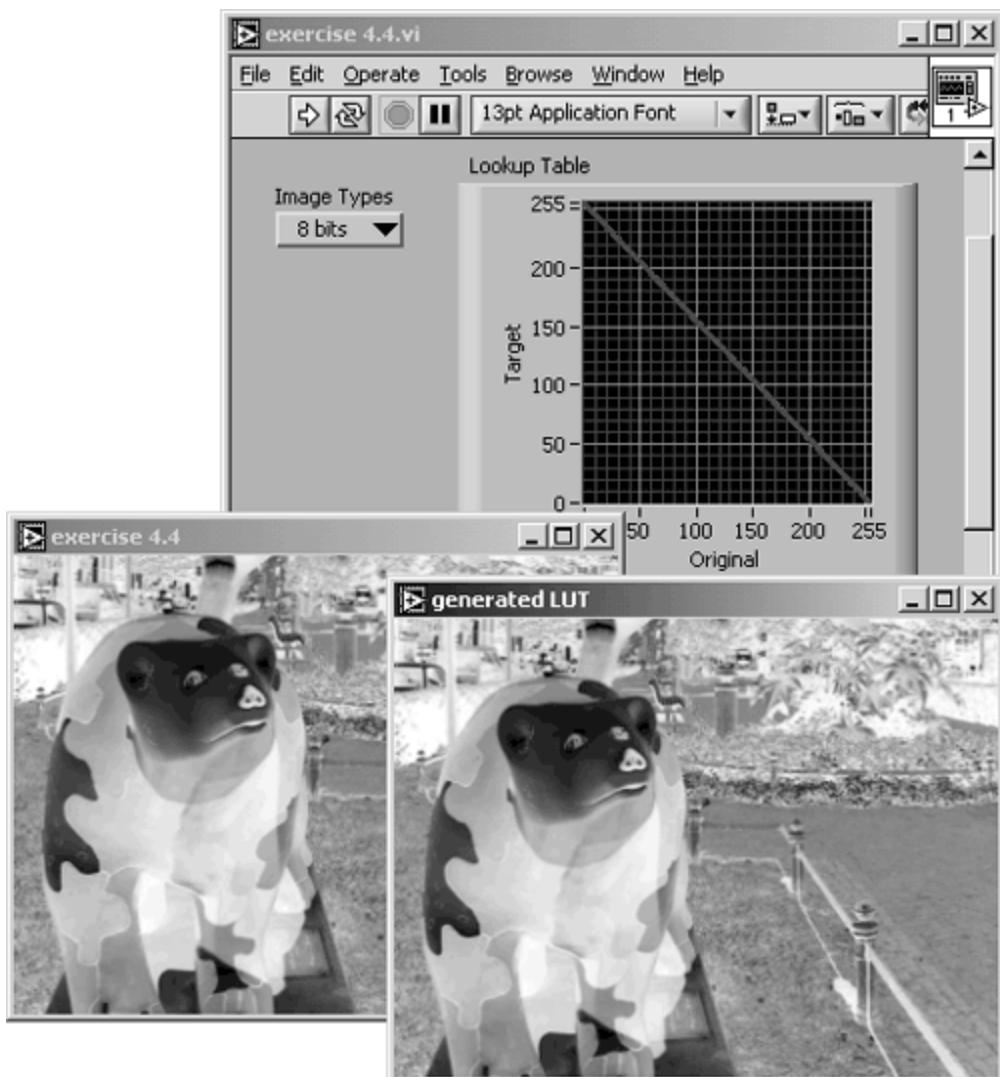
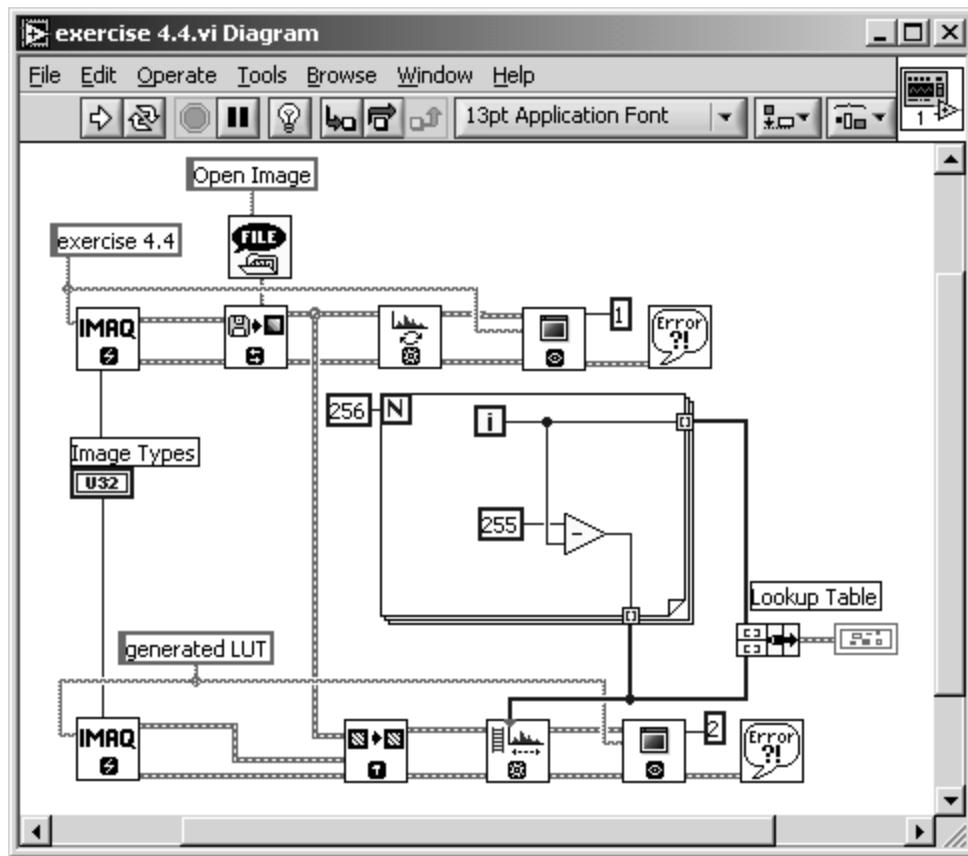


Figure 4.16. Diagram of [Exercise 4.4](#)



Now you know why you should plot the LuTs in an XY graph. In most cases, you can predict the results by simply looking at the plotted LuT. For example, if the LuT graph rises from bottom left to top right, then the resulting image will be not inverted; if it looks like a logarithmic function, the overall brightness of the resulting image will be increased because the log function compresses regions of high gray-level values.

Special LuTs: BCG Values

Another method is shown in [Exercise 4.5](#). Instead of creating the LuT by using mathematical functions, we can use the function **IMAQ_BCGLookup**, which changes the following parameters:

- Brightness: the median value of the gray-level values. If the LuT is drawn as a line as in the previous exercises, changing the brightness value shifts the line to the top (lighter) or to the bottom (darker).
- Contrast: the difference from one gray-level value to another. Varying the contrast value changes the slope of the line in the XY graph.
- Gamma: the gamma coefficient is equal to the power factor ρ we used in the previous exercises. If the gamma value is equal to 1, the graph always remains a straight line.

[Exercise 4.5](#) compares the IMAQ function with a self-constructed one:

Exercise 4.5: BCG Look-up Table.

The IMAQ function **IMAQ_BCGLookup** modifies the image by changing the values of brightness, contrast, and gamma. Create a VI that does the same with a manually generated LuT.

[Figure 4.17](#) shows a possible solution; [Figure 4.18](#) shows the diagram. I did only the brightness and the contrast adjustment; the realization of the gamma coefficient is left for you.

Figure 4.17. Using Special LuTs for Modifying Brightness and Contrast

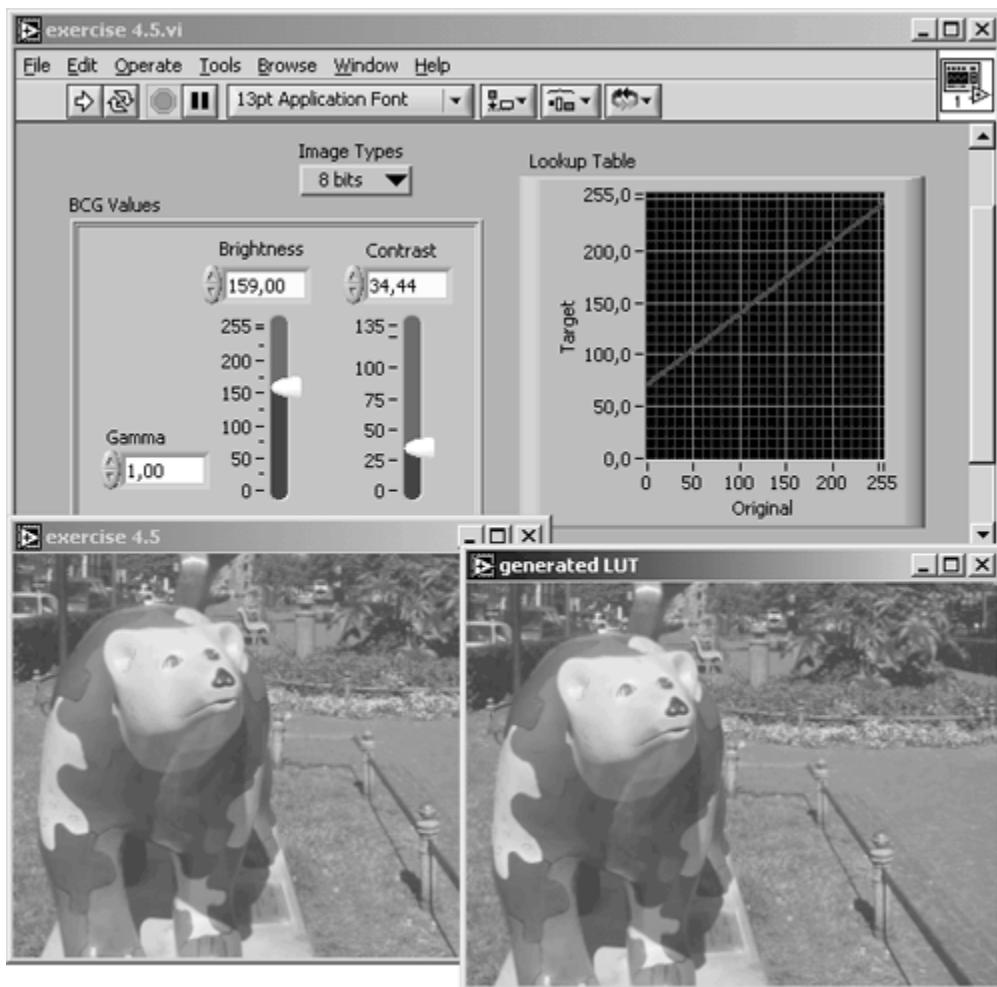
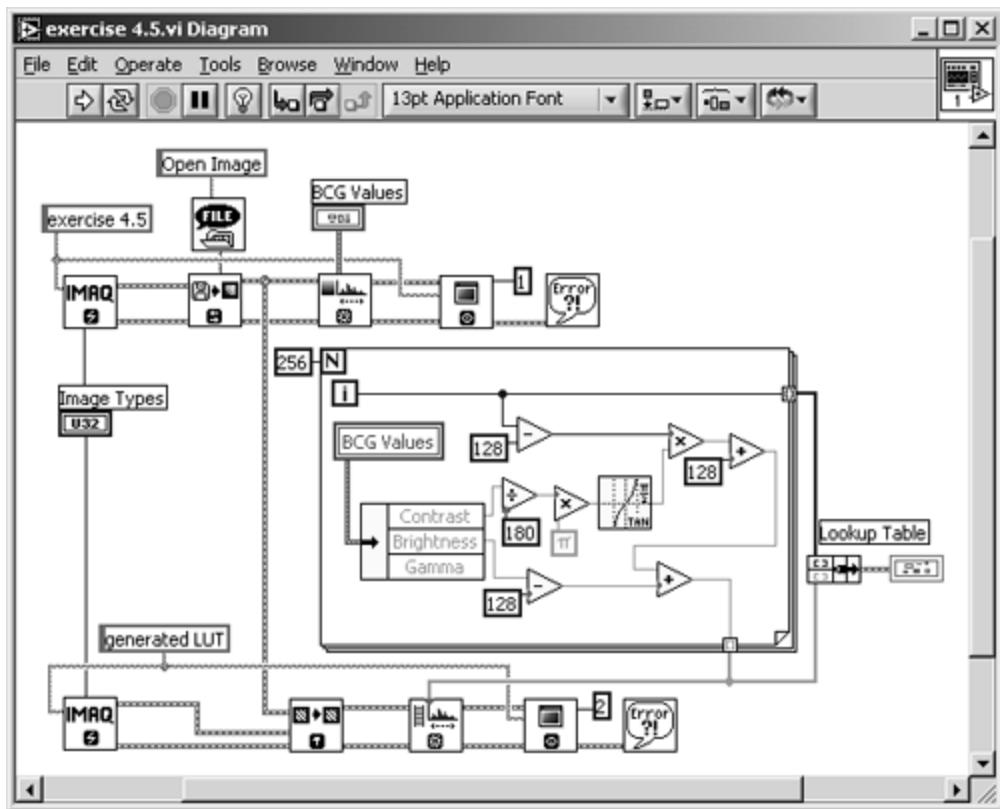


Figure 4.18. Diagram of [Exercise 4.5](#)



[Team LiB]

◀ PREVIOUS NEXT ▶

Spatial Image Filtering

As the word *filtering* suggests, most of the methods described in this section are used to improve the quality of the image. In general, these methods calculate new pixel values by using the original pixel value and those of its neighbors. For example, we can use the equation

Equation 4.12

$$s_{\text{out}}(x, y) = \frac{1}{m^2} \sum_{u=0}^{m-1} \sum_{v=0}^{m-1} s_{\text{in}}(x + k - u, y + k - v) \cdot f(u, v) ,$$

which looks very complicated. Let's start from the left: Eq. (4.12) calculates new pixel values s_{out} by using an $m \times m$ array. Usually, m is 3, 5, or 7; in most of our exercises we use $m = 3$, which means that the original pixel value s_{in} and the values of the eight surrounding pixels are used to calculate the new value. k is defined as $k = (m - 1)/2$.

The values of these nine pixels are modified with a *filter kernel*

Equation 4.13

$$\mathbf{F} = (f(u, v)) = \begin{pmatrix} f(0, 0) & f(0, 1) & f(0, 2) \\ f(1, 0) & f(1, 1) & f(1, 2) \\ f(2, 0) & f(2, 1) & f(2, 2) \end{pmatrix} .$$

As you can see in Eq. (4.12), the indices u and v depend upon x and y with $k = (m - 1)/2$. Using indices x and y , we can write

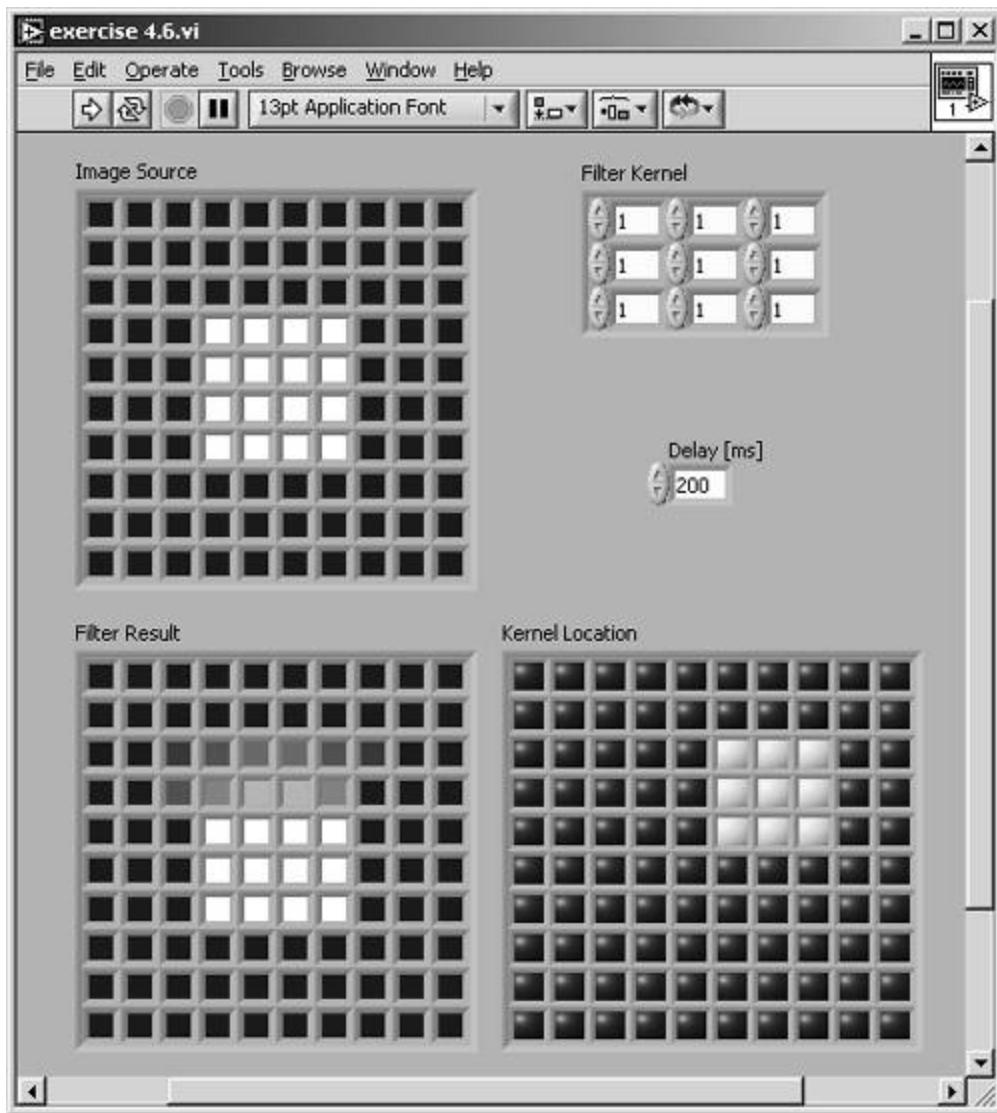
Equation 4.14

$$\mathbf{F} = \begin{pmatrix} f(x - 1, y - 1) & f(x, y - 1) & f(x + 1, y - 1) \\ f(x - 1, y) & f(x, y) & f(x + 1, y) \\ f(x - 1, y + 1) & f(x, y + 1) & f(x + 1, y + 1) \end{pmatrix} ,$$

which specifies the pixel itself and its eight surrounding neighbors.

The filter kernel moves from the top-left to the bottom-right corner of the image, as shown in [Figure 4.19](#). To visualize what happens during the process, do [Exercise 4.6](#):

Figure 4.19. Moving the Filter Kernel

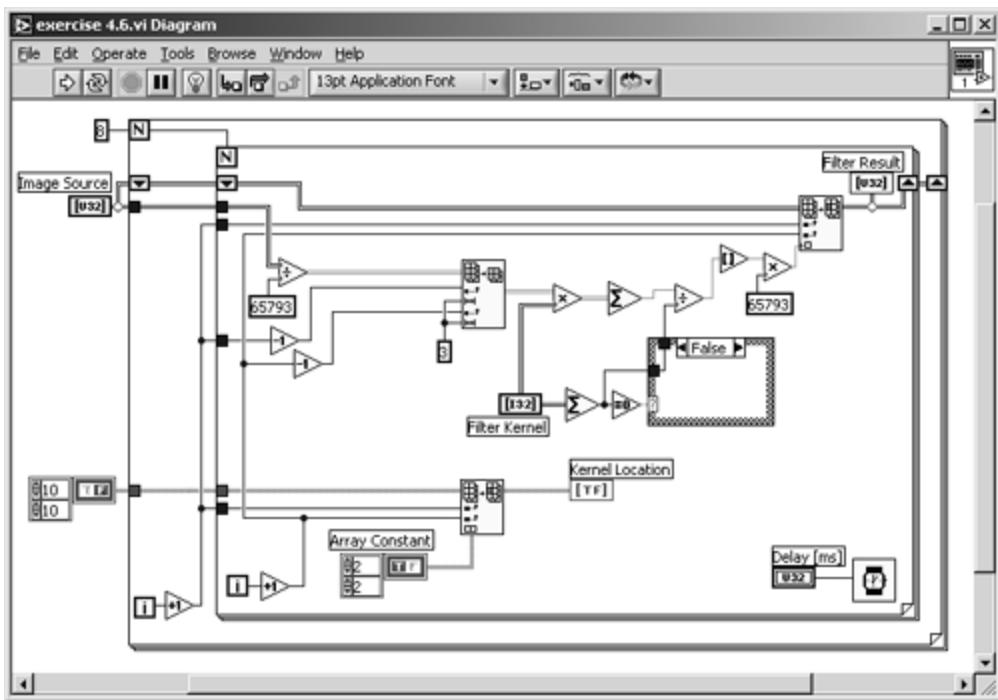


Exercise 4.6: Filter Kernel Movement.

Visualize the movement of a 3×3 filter kernel over a simulated image and display the results of Eq. (4.12). Figures 4.19 and 4.20 show a possible solution.

In this exercise I simulated the image with a 10×10 array of color boxes. Their values have to be divided by 64 k to get values of the 8-bit gray-scale set. The pixel and its eight neighbors are extracted and multiplied by the filter kernel, and the resulting values are added to the new pixel value.

Figure 4.20. Diagram of [Exercise 4.6](#)



The process described by Eq. (4.12) is also called *convolution*; that is why the filter kernel may be called *convolution kernel*.

You may have noticed that I did not calculate the border of the new image in [Exercise 4.6](#); the resulting image is only an 8 x 8 array. In fact, there are some possibilities for handling the border of an image during spatial filtering:

- The first possibility is exactly what we did in [Exercise 4.6](#): The resulting image is smaller by the border of 1 pixel.
- Next possibility: The gray-level values of the original image remain unchanged and are transferred in the resulting image.
- You can also use special values for the new border pixels; for example, 0, 127, or 255.
- Finally, you can use pixels from the opposite border for the calculation of the new values.

Kernel Families

LabVIEW provides a number of predefined filter kernels; it makes no sense to list all of them in this book. You can find them in IMAQ Vision Help: In a LabVIEW VI window, select Help / IMAQ Vision... and type "kernels" in the Index tab.

Nevertheless, we discuss a number of important kernels using the predefined IMAQ groups. To do this, we have to build a LabVIEW VI that makes it possible to test the predefined IMAQ Vision kernels as well as the manually defined kernels.

Exercise 4.7: IMAQ Vision Filter Kernels.

In this exercise VI, you will be able to visualize the built-in IMAQ Vision filters, and this allows you to define your own 3 x 3 kernels. [Figure 4.21](#) shows the front panel: The Menu Ring Control (bottom-left corner) makes it possible to switch between Predefined and Manual Kernel. I used Property Nodes here to make the unused

controls disappear.

Next, let's look at the diagram ([Figure 4.22](#)). The processing function we use here is `IMAQ Convolute`; the predefined kernels are generated with `IMAQ GetKernel`. So far, so good; but if we use manual kernels, we have to calculate a value called Divider, which is nothing more than $1/m^2$ in Eq. ([4.12](#)).

Obviously, $1/m^2$ is not the correct expression for the divider if we want to keep the overall pixel brightness of the image. Therefore, we must calculate the sum of all kernel elements for the divider value; with one exception[\[2\]](#) of course, 0. In this case we set the divider to 1.[\[3\]](#)

$$\mathbf{F} = \begin{pmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

[2] This happens quite often, for example consider a kernel

[3] Note that if you wire a value through a case structure in LabVIEW, you have to click once within the frame.

Figure 4.21. Visualizing Effects of Various Filter Kernels

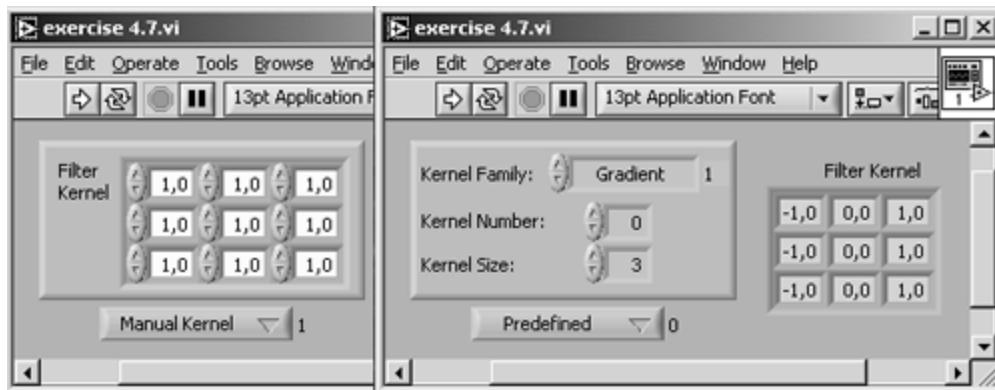
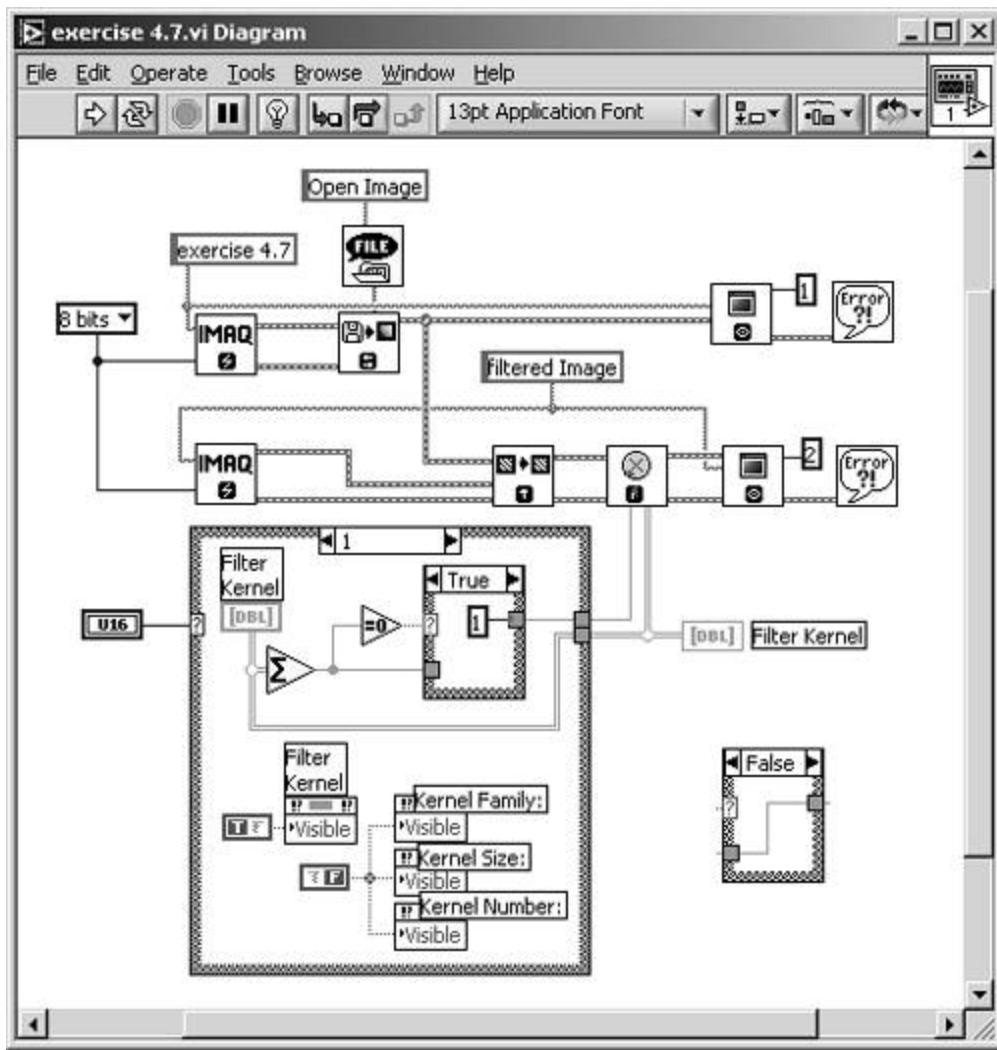


Figure 4.22. Diagram of [Exercise 4.7](#)



The four filter families which you can specify in [Exercise 4.7](#) (as well as in the Vision Builder) are:

- Smoothing
- Gaussian
- Gradient
- Laplacian

Image Smoothing

All smoothing filters build a weighted average of the surrounding pixels, and some of them also use the center pixel itself.

Filter Families: Smoothing

A typical smoothing kernel is shown in [Figure 4.23](#). Here, the new value is calculated as the average of all nine pixels using the same weight:

Equation 4.15

$$s_{\text{out}}(x, y) = \frac{1}{9} \sum_{u=x-1}^{x+1} \sum_{v=y-1}^{y+1} s_{\text{in}}(u, v) .$$

Figure 4.23. Filter Example: Smoothing (#5)



$$\mathbf{F} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} (\#0), \quad \mathbf{F} = \begin{pmatrix} 0 & 2 & 0 \\ 2 & 1 & 2 \\ 0 & 2 & 0 \end{pmatrix} (\#2),$$

Other typical smoothing kernels are

$$\mathbf{F} = \begin{pmatrix} 2 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 2 \end{pmatrix} (\#6).$$

Note that the coefficient for the center pixel is either 0 or 1; all other coefficients are positive.

Filter Families: Gaussian

The center pixel coefficient of a Gaussian kernel is always greater than 1, and thus greater than the other coefficients because it simulates the shape of a Gaussian curve. [Figure 4.24](#) shows an example.

Figure 4.24. Filter Example: Gaussian (#4)



$$\mathbf{F} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{pmatrix} (\#0), \quad \mathbf{F} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{pmatrix} (\#1), \quad \text{or}$$

More examples for Gaussian kernels are

$$\mathbf{F} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 4 & 1 \\ 1 & 1 & 1 \end{pmatrix} (\#3).$$

Both smoothing and Gaussian kernels correspond to the electrical synonym "low-pass filter" because sharp edges, which are removed by these filters, can be described by high frequencies.

Edge Detection and Enhancement

An important area of study is the detection of edges in an image. An *edge* is an area of an image in which a significant change of pixel brightness occurs.

Filter Families: Gradient

A gradient filter extracts a significant brightness change in a specific direction and is thus able to extract edges rectangular to this direction. [Figure 4.25](#) shows the effect of a vertical gradient filter, which extracts dark-bright changes from left to right.

Figure 4.25. Filter Example: Gradient (#0)



Note that the center coefficient of this kernel is 0; if we make it equal to 1 (as in [Figure 4.26](#)), the edge information is added to the original value. The original image remains and the edges in the specified direction are highlighted.

Figure 4.26. Filter Example: Gradient (#1)



[Figure 4.27](#) shows the similar effect in a horizontal direction (dark-bright changes from bottom to top).

Figure 4.27. Filter Example: Gradient (#4)



Gradient kernels like those in [Figure 4.25](#) and [Figure 4.27](#) are also known as *Prewitt filters* or

$$\mathbf{F} = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix},$$

Prewitt kernels. Another group of gradient kernels, for example, called *Sobel filters* or *Sobel kernels*. A Sobel kernel gives the specified filter direction a stronger weight.

In general, the mathematic value *gradient* specifies the amount of change of a value in a certain direction. The simplest filter kernels, used in two orthogonal directions, are

Equation 4.16

$$\mathbf{F}_x = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \quad \text{and} \quad \mathbf{F}_y = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{pmatrix},$$

resulting in two images, $/_x = (s_x(x,y))$ and $/_y = (s_y(x,y))$. Value and direction of the gradient are therefore

Equation 4.17

$$\sqrt{s_x(x,y)^2 + s_y(x,y)^2} \quad \text{and} \quad \tan \varphi = \frac{s_x(x,y)}{s_y(x,y)} .$$

Filter Families: Laplacian

All kernels of the Laplacian filter group are omnidirectional; that means they provide edge information in each direction. Like the Laplacian operator in mathematics, the Laplacian kernels are of a second-order derivative type and show two interesting effects:

- If the sum of all coefficients is equal to 0, the filter kernel shows all image areas with a significant brightness change; that means it works as an omnidirectional edge detector (see [Figures 4.28](#) and [4.30](#)).
- If the center coefficient is greater than the sum of the absolute values of all other coefficients, the original image is superimposed over the edge information (see [Figures 4.29](#) and [4.31](#)).

Figure 4.28. Filter Example: Laplace (#0)



Figure 4.29. Filter Example: Laplace (#1)

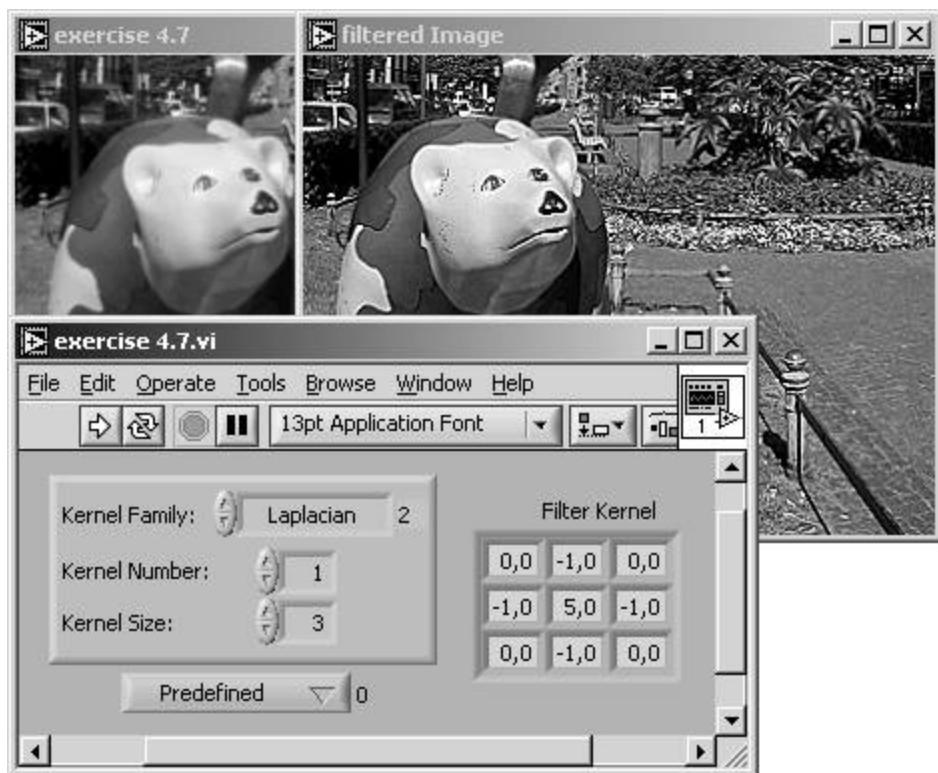
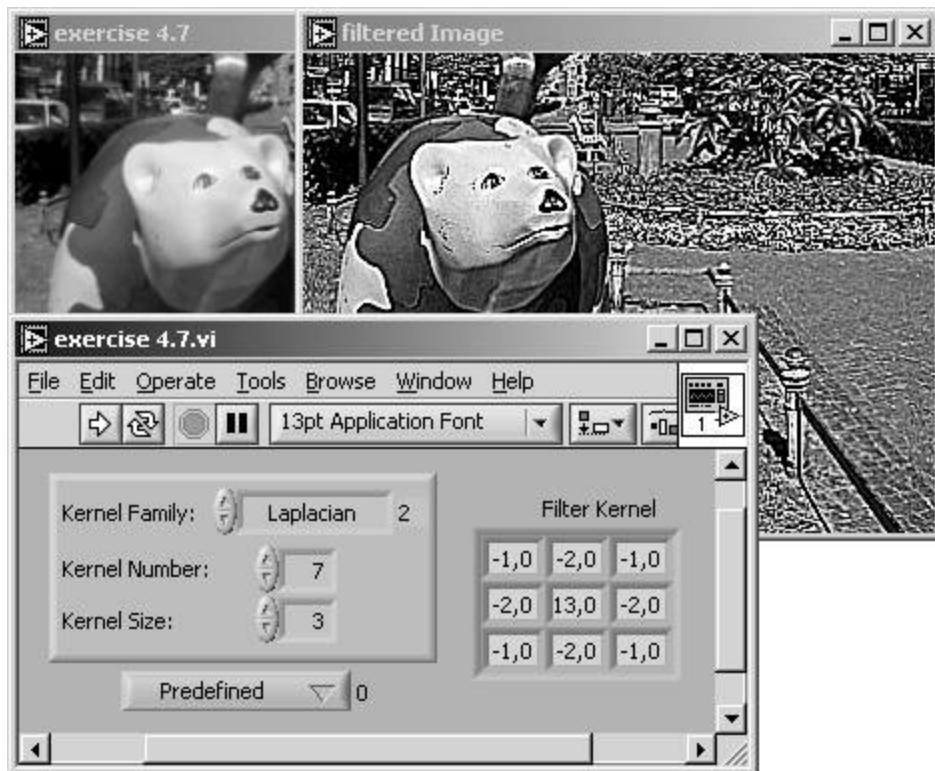


Figure 4.30. Filter Example: Laplace (#6)



Figure 4.31. Filter Example: Laplace (#7)



Both gradient and Laplacian kernels correspond to the electrical synonym "high-pass filter." The reason is the same as before; sharp edges can be described by high frequencies.

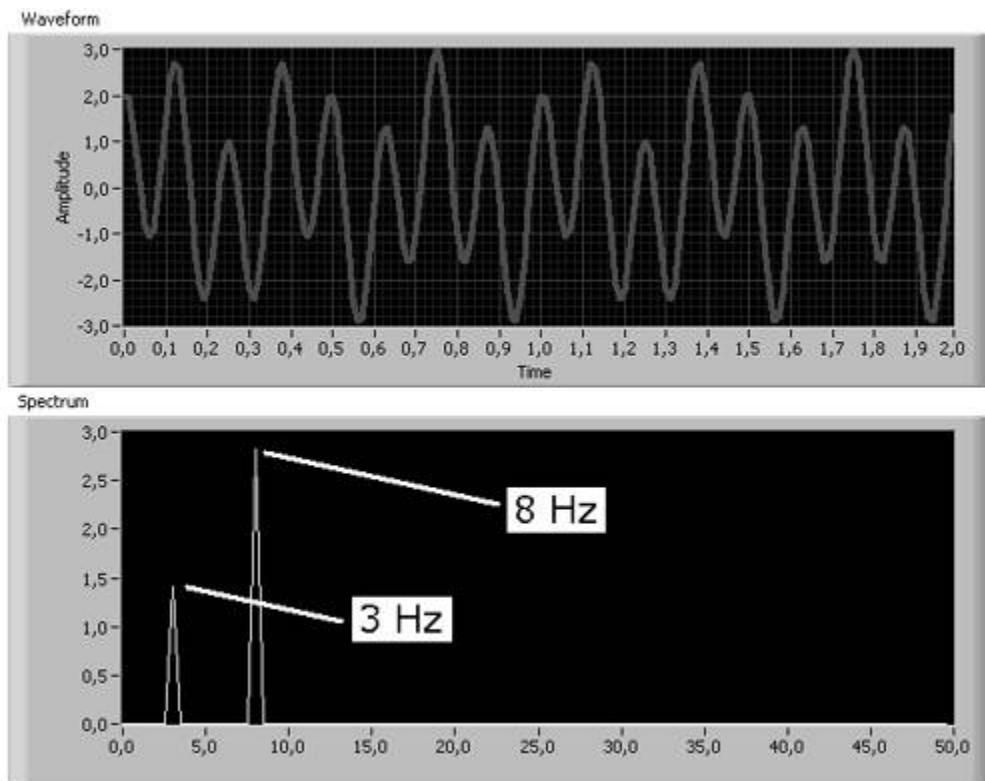
[Team LiB]

◀ PREVIOUS NEXT ▶

Frequency Filtering

If we follow this idea further, we should be able to describe the entire image by a set of frequencies. In electrical engineering, every periodic waveform can be described by a set of sine and cosine functions, according to the *Fourier transform* and as shown in [Figure 4.32](#).

Figure 4.32. Waveform Spectrum



Here, the resulting waveform consists of two pure sine waveforms, one with $f_1 = 3 \text{ Hz}$ and one with $f_2 = 8 \text{ Hz}$, which is clearly visible in the frequency spectrum, but not in the waveform window at the top. A rectangular waveform, on the other hand, shows sharp edges and thus contains much higher frequencies as well.

Coming back to images, the Fourier transform is described by

Equation 4.18

$$S(u, v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} s(x, y) e^{-i 2\pi(xu+yv)} dx dy ,$$

with $s(x, y)$ as the pixel intensities. Try the following exercise.

Exercise 4.8: Frequency Representation of Images.

Create a frequency representation of our bear image by using the function `IMAQ FFT`. You can switch between two display modes, using `IMAQ ComplexFlipFrequency` (both functions are described later). Note that the image type of the frequency image has to be set to Complex (see [Figures 4.33](#) and [4.34](#)).

Figure 4.33. FFT Spectrum of an Image

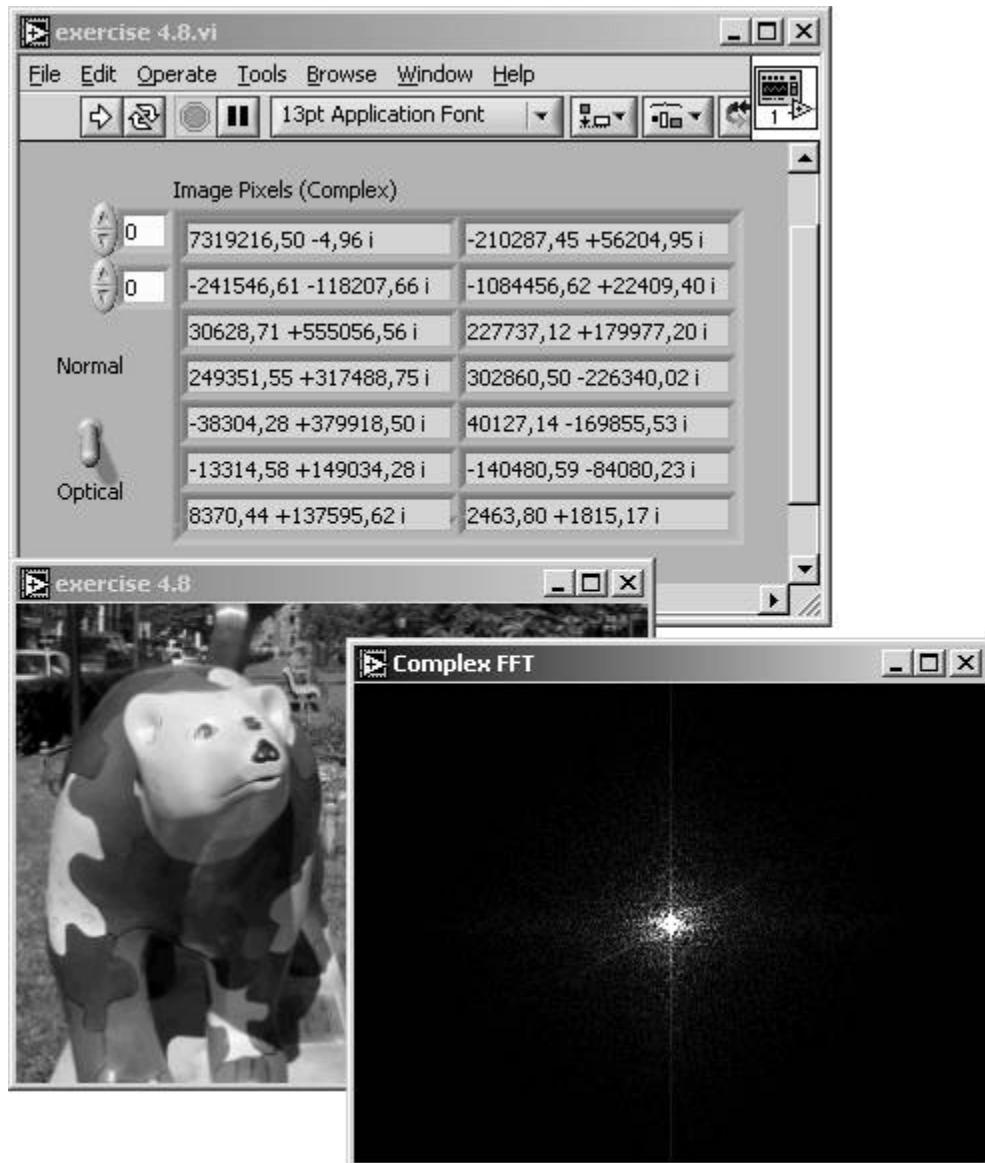
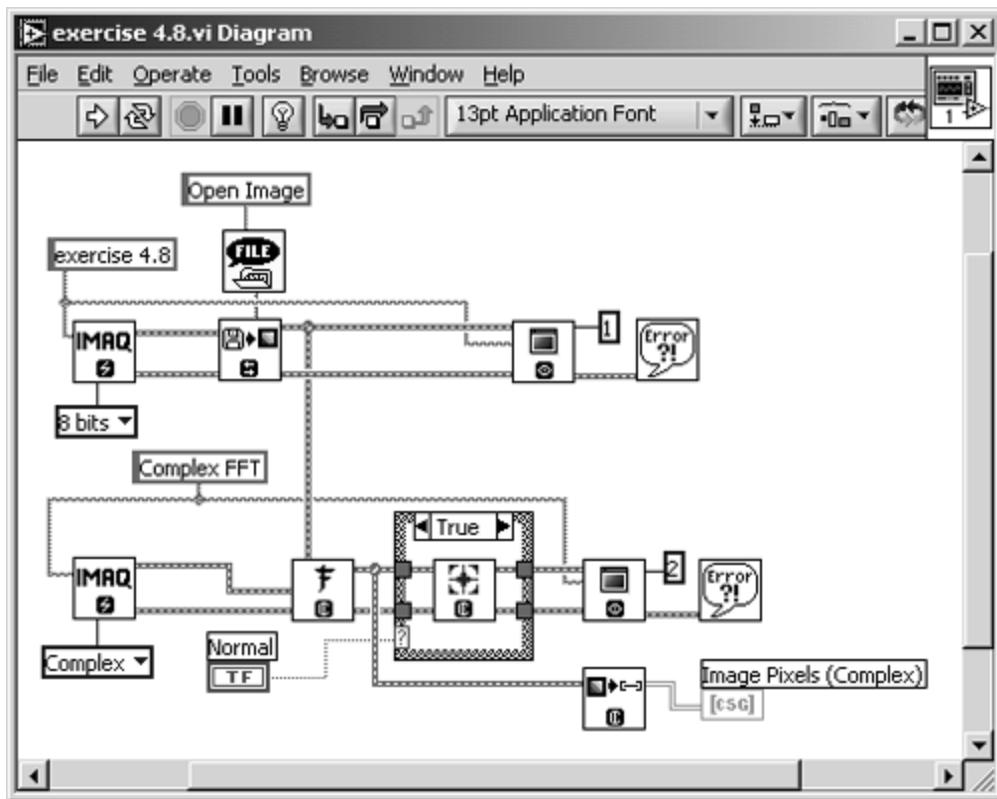


Figure 4.34. FFT Spectrum of an Image



The exponential function of Eq. (4.18) can also be written as

Equation 4.19

$$e^{-i2\pi\alpha} = \cos 2\pi\alpha - i \sin 2\pi\alpha \quad ,$$

and that is why the entire set of pixel intensities can be described by a sum of sine and cosine functions but the result is complex.[\[4\]](#)

[\[4\]](#) On the other hand, the complex format makes it easier to display the results in a two-dimensional image.

IMAQ Vision uses a slightly different function for the calculation of the spectral image, the *fast Fourier transform* (FFT). The FFT is more efficient than the Fourier transform and can be described by

Equation 4.20

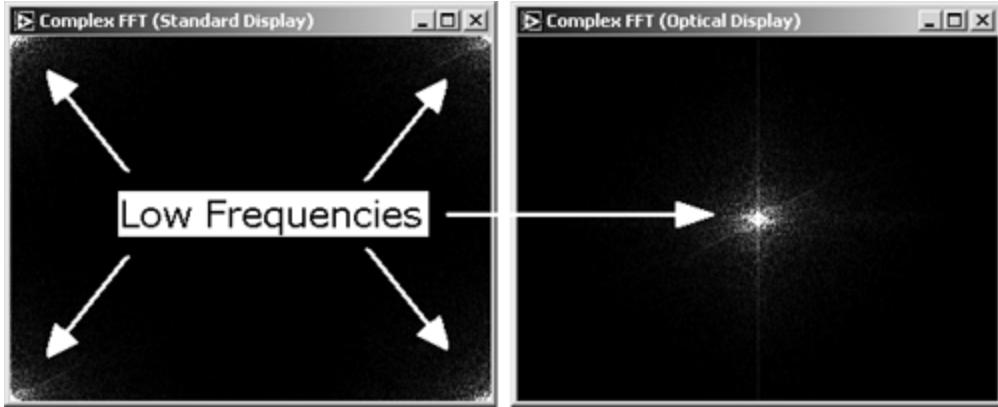
$$S(u, v) = \frac{1}{nm} \sum_{x=0}^{n-1} \sum_{y=0}^{m-1} s(x, y) e^{-i2\pi\left(\frac{ux}{n} + \frac{vy}{m}\right)} \quad ,$$

where n and m are the numbers of pixels in x and y direction, respectively.

[Figure 4.35](#) shows another important issue: the difference between standard display of the fast Fourier transform and the *optical display*. The standard display groups high frequencies in the middle of the FFT image, whereas low frequencies are located in the four corners (left image in [Figure 4.35](#)). The optical display behaves conversely: Low frequencies are grouped in the middle and high frequencies are located in the corners.[\[5\]](#)

[5] Note a mistake in IMAQ Vision's help file: **IMAQ FFT** generates output in optical display, not in standard.

Figure 4.35. FFT Spectrum (Standard and Optical Display)



FFT Filtering: Truncate

If we have the entire frequency information of an image, we can easily remove or attenuate certain frequency ranges. The simplest way to do this is to use the filtering function **IMAQ ComplexTruncate**. You can modify [Exercise 4.8](#) to do this.

Exercise 4.9: Truncation Filtering.

Add the function **IMAQComplexTruncate** to [Exercise 4.8](#) and also add the necessary controls (see [Figures 4.36](#) and [4.37](#)). The control called Truncation Frequencies specifies the frequency percentage below which (high pass) or above which (low pass) the frequency amplitudes are set to 0.

In addition, use **IMAQInverseFFT** to recalculate the image, using only the remaining frequencies. [Figure 4.36](#) shows the result of a low-pass truncation filter; [Figure 4.38](#), the result of a high-pass truncation filter.

Figure 4.36. FFT Low-Pass Filter

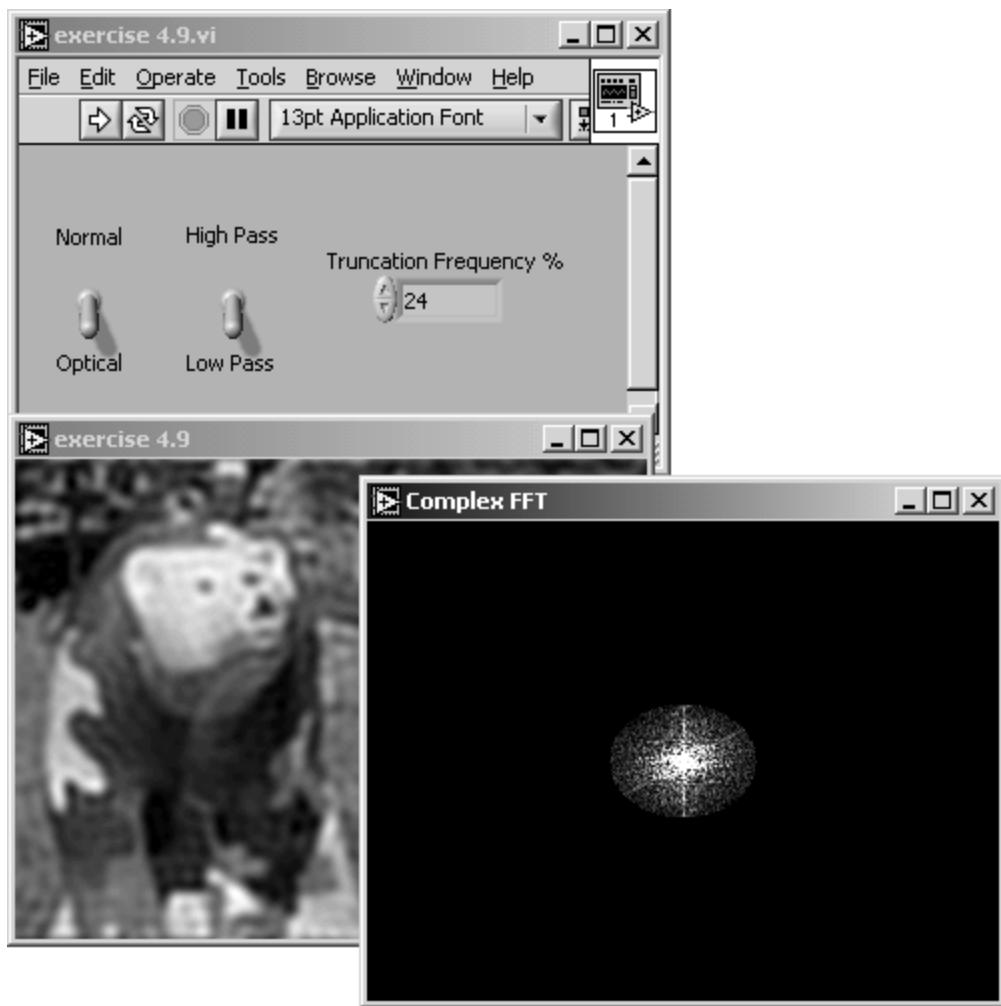


Figure 4.37. Diagram of [Exercise 4.9](#)

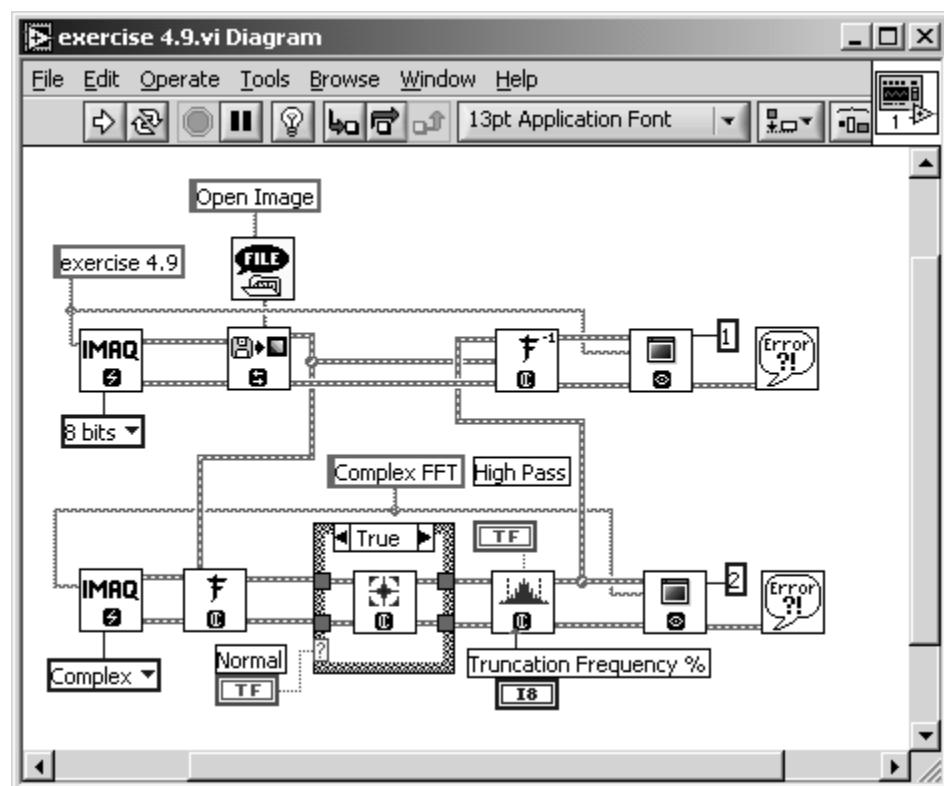
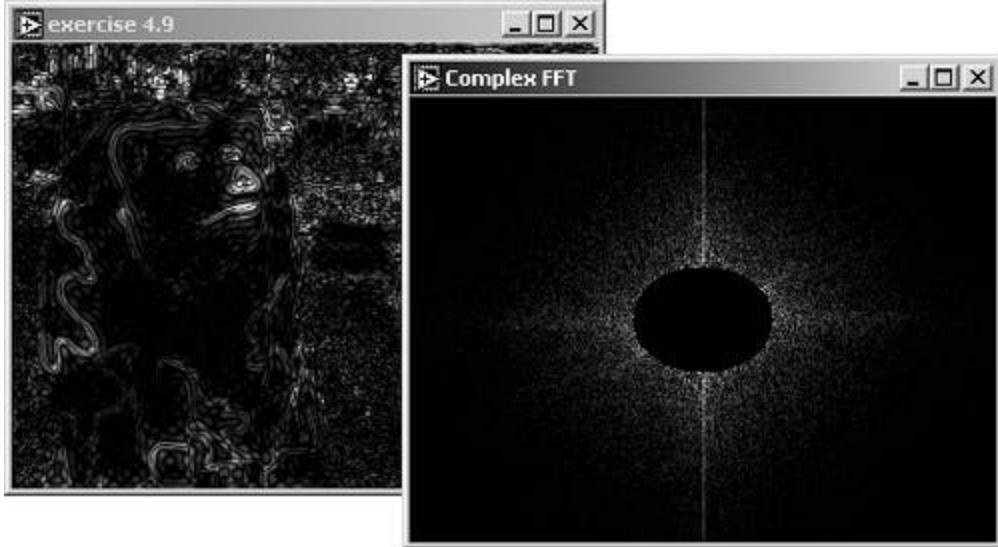


Figure 4.38. FFT High-Pass Filter



The inverse fast Fourier transform (FFT) used in [Exercise 4.9](#) and in the function `IMAQ_InverseFFT` is calculated as follows:

Equation 4.21

$$s(x, y) = \sum_{u=0}^{n-1} \sum_{v=0}^{m-1} S(u, v) e^{i 2\pi \left(\frac{ux}{n} + \frac{vy}{m} \right)} .$$

FFT Filtering: Attenuate

An *attenuation* filter applies a linear attenuation to the full frequency range. In a low-pass attenuation filter, each frequency f from f_0 to f_{\max} is multiplied by a coefficient C , which is a function of f according to

Equation 4.22

$$C(f) = \frac{f_{\max} - f}{f_{\max} - f_0} .$$

It is easy to modify [Exercise 4.9](#) to a VI that performs attenuation filtering.

Exercise 4.10: Attenuation Filtering.

Replace the IMAQ Vision function `IMAQComplex_Truncate` with `IMAQ_ComplexAttenuate` and watch the results (see [Figure 4.39](#)). Note that the high-pass setting obviously returns a black image; is that true?

The formula for the coefficient C in a high-pass attenuation filter is

Equation 4.23

$$C(f) = \frac{f - f_0}{f_{\max} - f_0} ,$$

which means that only high frequencies are multiplied by a significantly high coefficient $\alpha(\lambda)$. Therefore, the high-pass result of [Exercise 4.10](#) ([Figure 4.40](#)) contains a number of frequencies that are not yet visible. You may insert the LuT function **IMAQ Equalize** to make them visible (do not forget to cast the image to 8 bits).

Figure 4.39. FFT Low-Pass Attenuation Filter

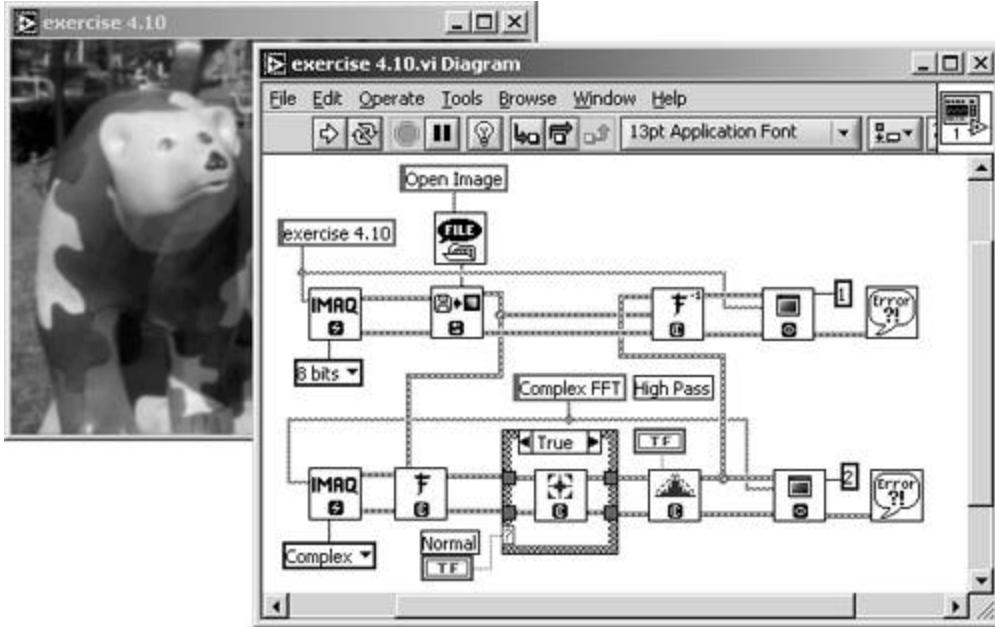
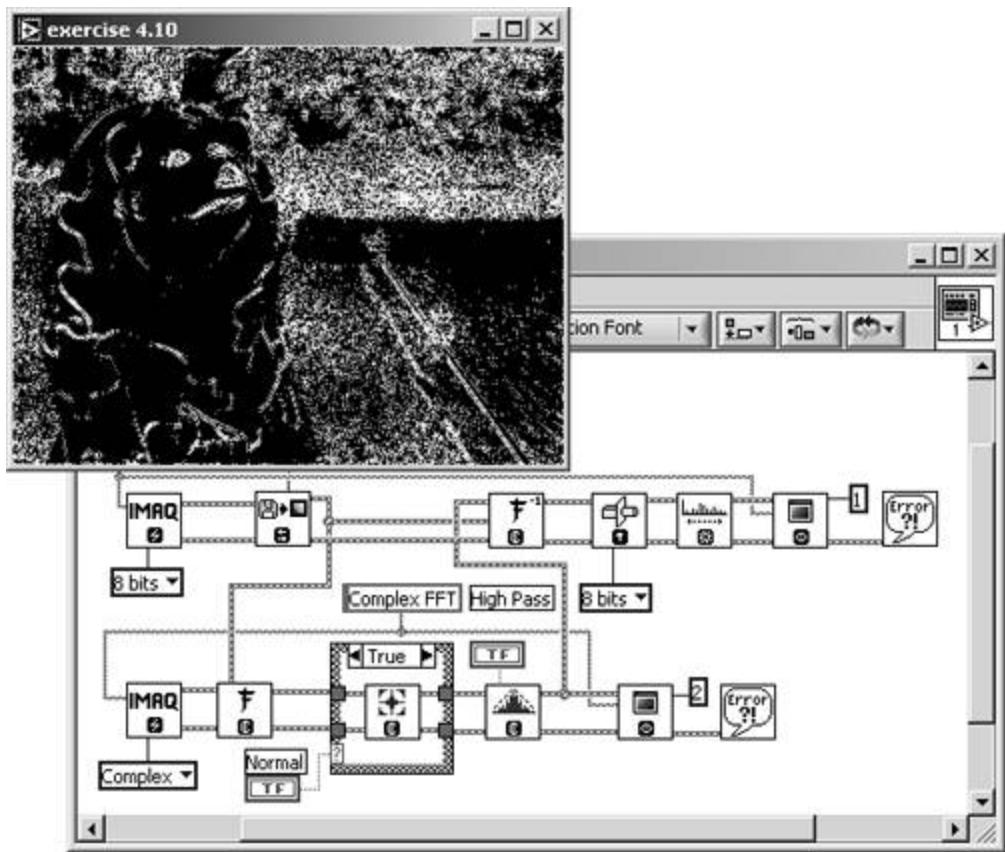


Figure 4.40. FFT High-Pass Attenuation Result



[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

Morphology Functions

If you search IMAQ Vision Builder's menus for morphology functions, you will find two different types ([Figure 4.41](#)):

- Basic and Advanced Morphology in the Binary menu;
- Gray Morphology in the Grayscale menu.

Figure 4.41. Morphology Functions in IMAQ Vision Builder



Basically, morphology operations change the structure of particles in an image. Therefore, we have to define the meaning of "particle": this is easy for a binary image: *Particles* are regions in which the pixel value is 1. The rest of the image (pixel value 0) is called *background*.[\[6\]](#)

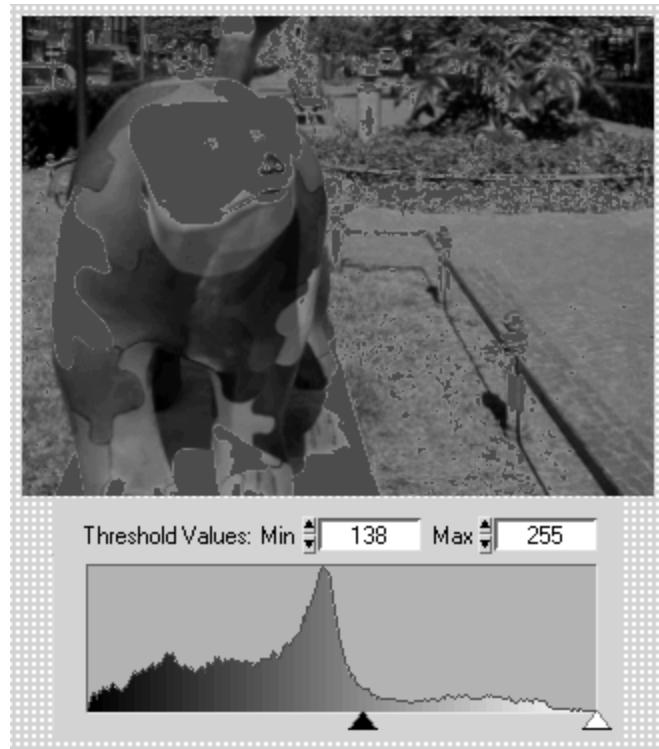
[6] Remember that a binary image consists only of two different pixel values: 0 and 1.

That is why we first learn about binary morphology functions; gray-level morphology is discussed later in this section.

Thresholding

First of all, we have to convert the gray-scaled images we used in previous exercises into binary images. The function that performs this operation is called *thresholding*. [Figure 4.42](#) shows this process, using IMAQ Vision Builder: two gray-level values, adjusted by the black and the white triangle, specify a region of values within which the pixel value is set to 1; all other levels are set to 0.

Figure 4.42. Thresholding with IMAQ Vision Builder



[Figure 4.43](#) shows the result. Here and in the Vision Builder, pixels with value 1 are marked red. In reality, the image is binary, that is, containing only pixel values 0 and 1.

Figure 4.43. Result of Thresholding Operation



Next, we try thresholding directly in LabVIEW and IMAQ Vision.

Exercise 4.11: Thresholded Image.

Use the function `IMAQ Threshold` to obtain a binary image. In this case, use a While loop so that the threshold level can be adjusted easily. Note that you have to specify a binary palette with the function `IMAQ GetPalette` for the thresholded image. Compare your results to [Figures 4.44](#) and [4.45](#).

Figure 4.44. Thresholding with IMAQ Vision

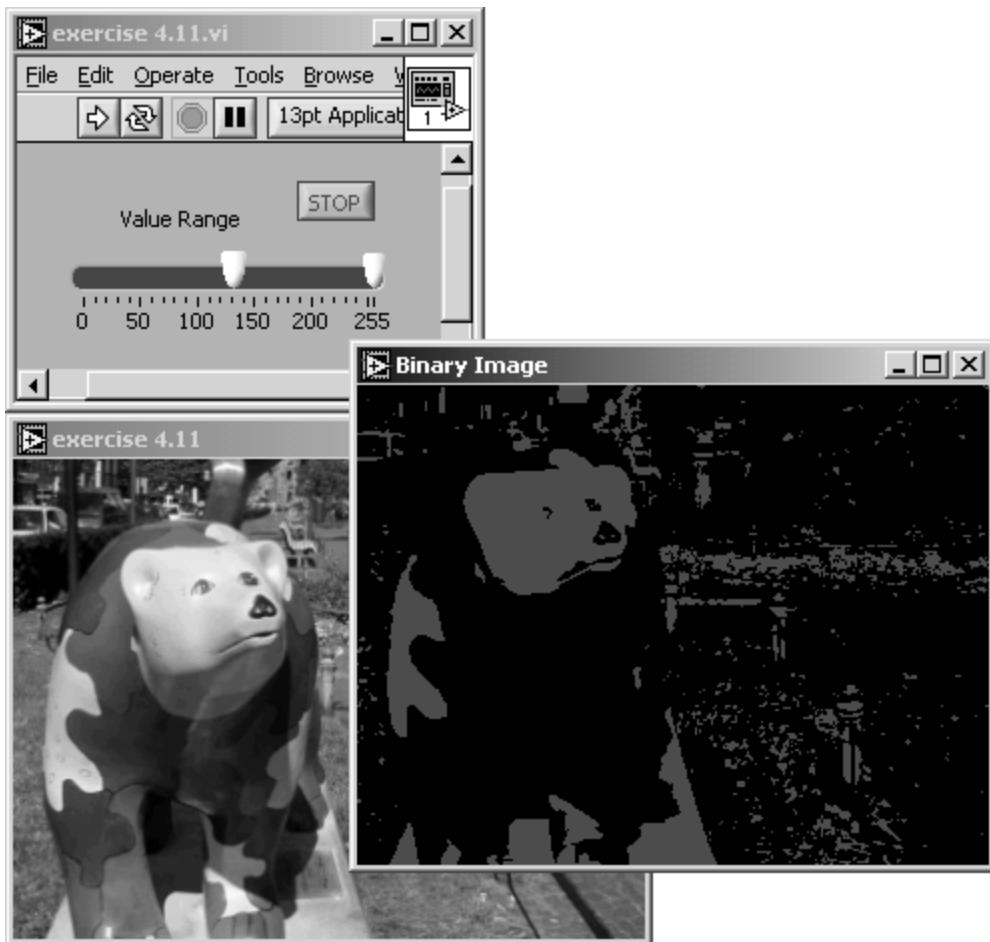
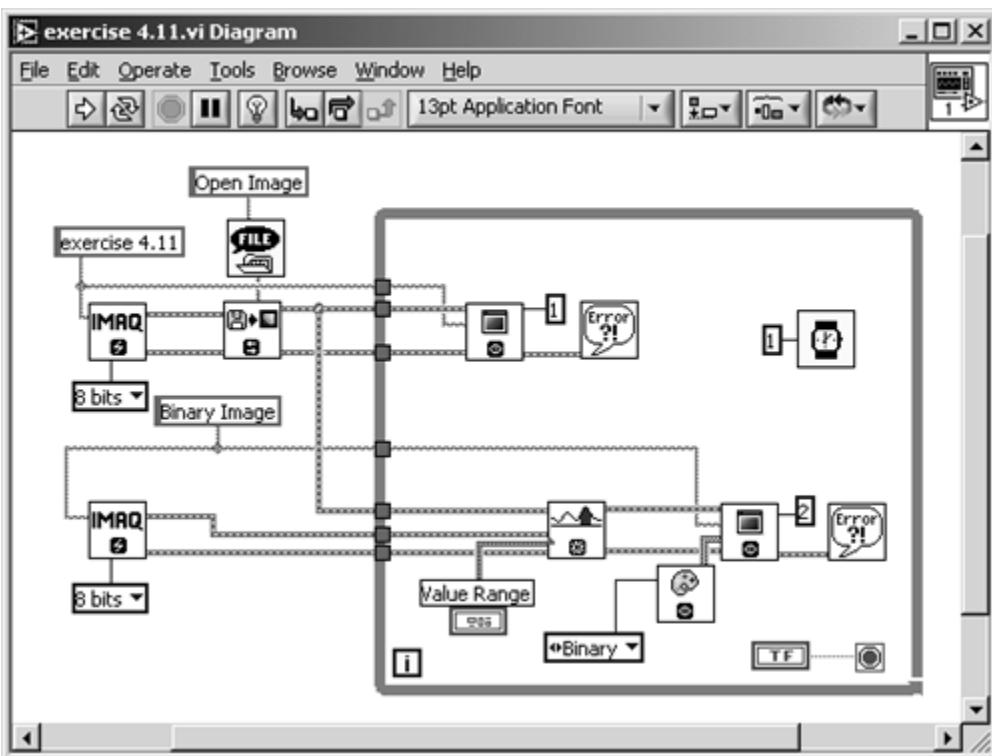


Figure 4.45. Diagram of [Exercise 4.11](#)



IMAQ Vision also contains some predefined regions for areas with pixel values equal to 1; they are classified with the following names:

- Clustering
- Entropy
- Metric
- Moments
- Interclass Variance

All of these functions are based on statistical methods. *Clustering*, for example, is the only method out of these five that allows the separation of the image into multiple regions of a certain gray level if the method is applied sequentially.^[7] All other methods are described in [4]; [Exercise 4.12](#) shows you how to use them and how to obtain the lower value of the threshold region (when these methods are used, the upper value is always set to 255):

^[7] Sequential clustering is also known as *multiclass thresholding* [4].

Exercise 4.12: Auto Thresholding.

Replace the control for adjusting the threshold level in [Exercise 4.11](#) by the function **IMAQ_AutoBThreshold**, which returns the respective threshold value, depending on the image, of course, and the selected statistical method. See [Figures 4.46](#) and [4.47](#).

Figure 4.46. Predefined Thresholding Functions

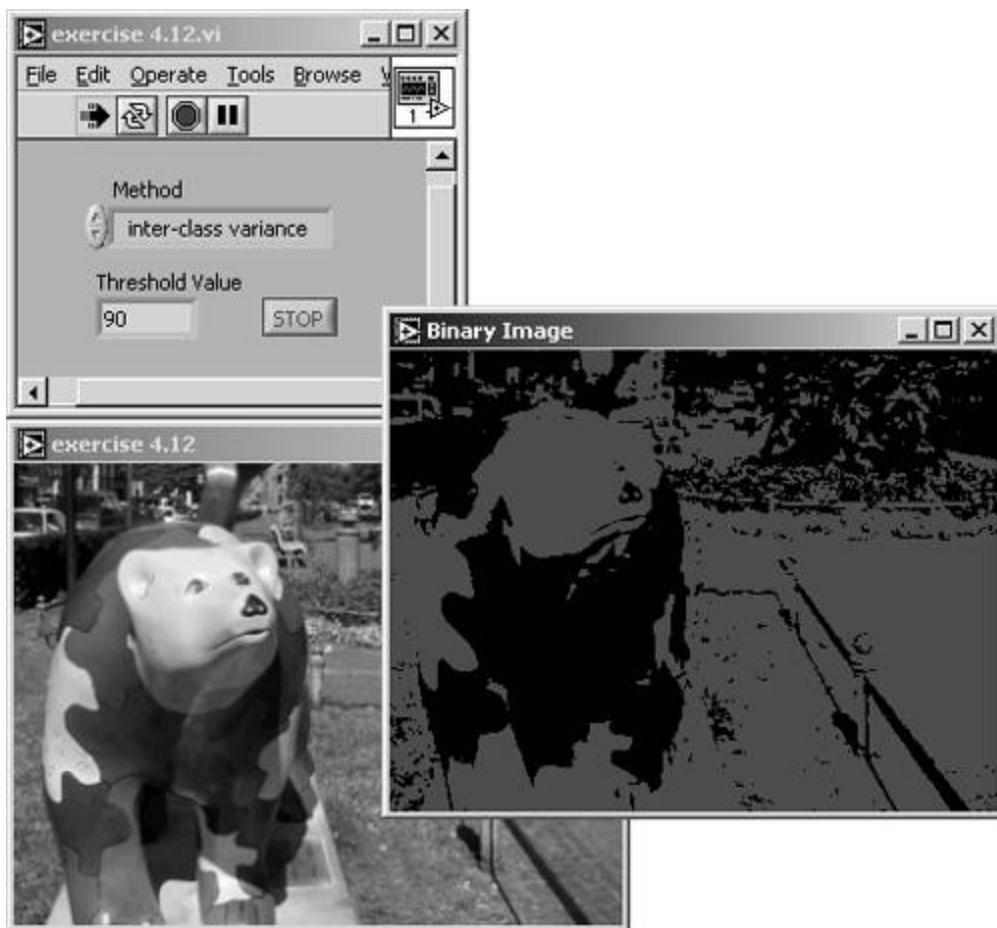
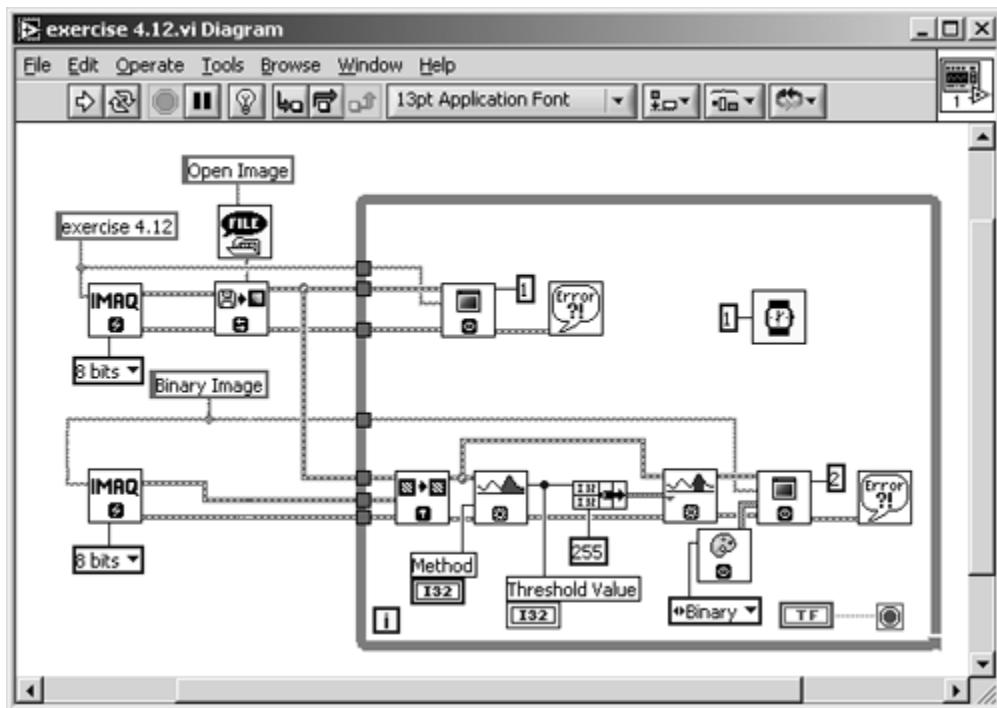


Figure 4.47. Diagram of [Exercise 4.12](#)



Reference [4] also gives the good advice that sometimes it makes sense to invert your image before applying one of those threshold methods, to get more useful results.

An interesting method for the separation of objects from the background is possible if you have a reference image of the background only. By calculating the *difference image* of these two and

then applying a threshold function, you will get results of a much higher quality. You can read more about difference images in the application section of [Chapter 5](#).

IMAQ Vision also contains a function—[IMAQ MultiThreshold](#)—that allows you to define multiple threshold regions to an image. By the way, you can also apply threshold regions to special planes of color images, for example, the hue plane. You can also find an interesting application that uses this method in [Chapter 5](#).

What about direct thresholding of color images? If you use the function [IMAQ ColorThreshold](#), you can directly apply threshold values to color images; either to the red, green, and blue plane (RGB color model) or to the hue, saturation, and luminance plane (HSL color model).

A final remark: Because we usually display our images on a computer screen, our *binary* images cannot have only pixel values of 0 and 1; in the 8-bit gray-scale set of our monitor, we would see no difference between them. Therefore, all image processing and analysis software, including IMAQ Vision, displays them with gray-level values of 0 and 255, respectively.

Binary Morphology

As explained above, morphology functions change the structure of objects (often called *particles*) of a (usually binary) image. Therefore, we have to define what the term *structure* (or *shape*) means.

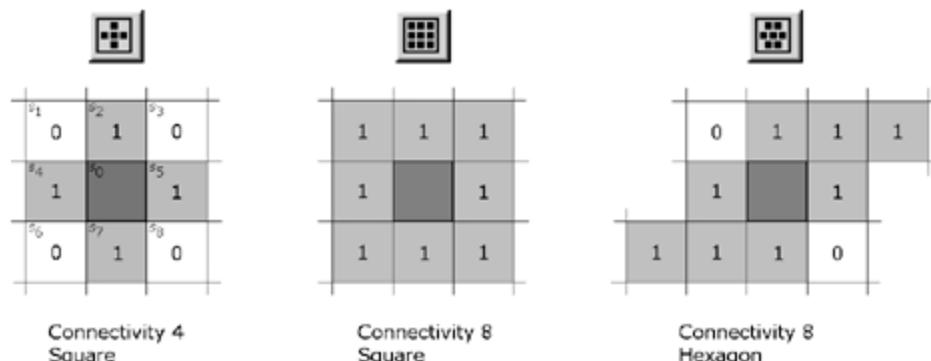
We always talk about images, which use *pixels* as their smallest element with certain properties. Objects or particles are coherent groups of pixels with the same properties (especially pixel value or pixel value range). The shape or structure of an object can be changed if pixels are added to or removed from the border (the area where pixel values change) of the object.

To calculate new pixel values, *structuring elements* are used: the group of pixels surrounding the pixel to be calculated. [Figure 4.48](#) shows some examples of the elements defined below:

- The *shape* of the structuring element is either rectangular (square; left and center examples) or hexagonal (example on the right).
- The *connectivity* defines whether four or all eight surrounding pixels are used to calculate the new center pixel value.^[8]

[8] In the case of a 3 x 3 structuring element.

Figure 4.48. Examples of Structuring Elements



The symbols at the top of [Figure 4.48](#) are used by IMAQ Vision Builder to indicate or to set the shape and the connectivity of the structuring element. A 1 in a pixel means that this pixel value is used for the calculation of the new center pixel value, according to

Equation 4.24

$$s'_0 = f(s_2, s_4, s_5, s_7)$$

for connectivity = 4 and

Equation 4.25

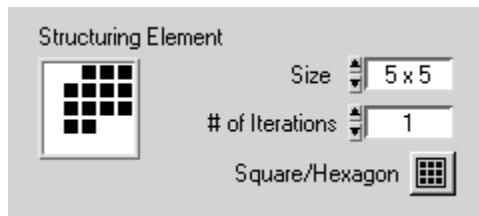
$$s'_0 = f(s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8)$$

for connectivity = 8.

It is also possible to define other (bigger) structuring elements than the 3×3 examples of [Figure 4.48](#). In IMAQ Vision Builder, 5×5 and 7×7 are also possible. Note that the procedure is quite similar to the filtering methods we discussed in a previous section.

In IMAQ Vision Builder, you can change the shape of the structuring element by simply clicking on the small black squares in the control (see [Figure 4.49](#)).

Figure 4.49. Configuring the Structuring Element in IMAQ Vision Builder



In our following examples and exercises we use a thresholded binary image, resulting from our gray-scaled bear image, by applying the *entropy* threshold function (or you can use manual threshold; set upper level to 255 and lower level to 136). The resulting image ([Figure 4.50](#)) contains a sufficient number of particles in the background, so the following functions have a clearly visible effect.

Erosion and Dilation

These two functions are fundamental for almost all morphology operations. *Erosion* is a function that basically removes (sets the value to 0) pixels from the border of particles or objects. If particles are very small, they may be removed totally.

The algorithm is quite easy if we consider the structuring element, centered on the pixel (value) s_0 : *If the value of at least one pixel of the structuring element is equal to 0* (which means that the structuring element is located at the border of an object), s_0 is set to 0, else s_0 is set to 1 (Remember that we only consider pixels masked with 1 in the structuring element) [4].

We can test all relevant morphology functions in [Exercise 4.13](#).

Exercise 4.13: Morphology Functions.

Create a VI that displays the results of the most common binary morphology functions, using the IMAQ Vision function [IMAQ Morphology](#). Make the structuring element adjustable with a 3×3 array control. Compare your results to [Figure 4.50](#), which also shows the result of an erosion, and [Figure 4.51](#).

Note that some small particles in the background have disappeared; also note that the big particles are smaller than in the original image. Try different structuring elements by changing some of the array elements and watch the results.

Figure 4.50. Morphology Functions: Erosion

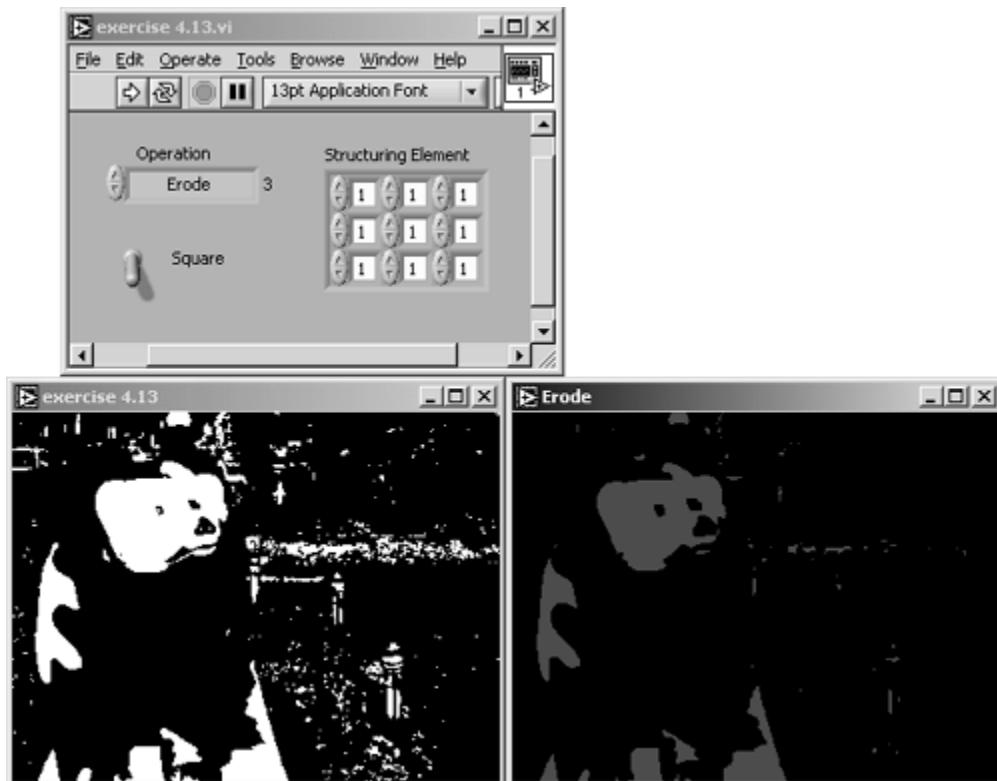
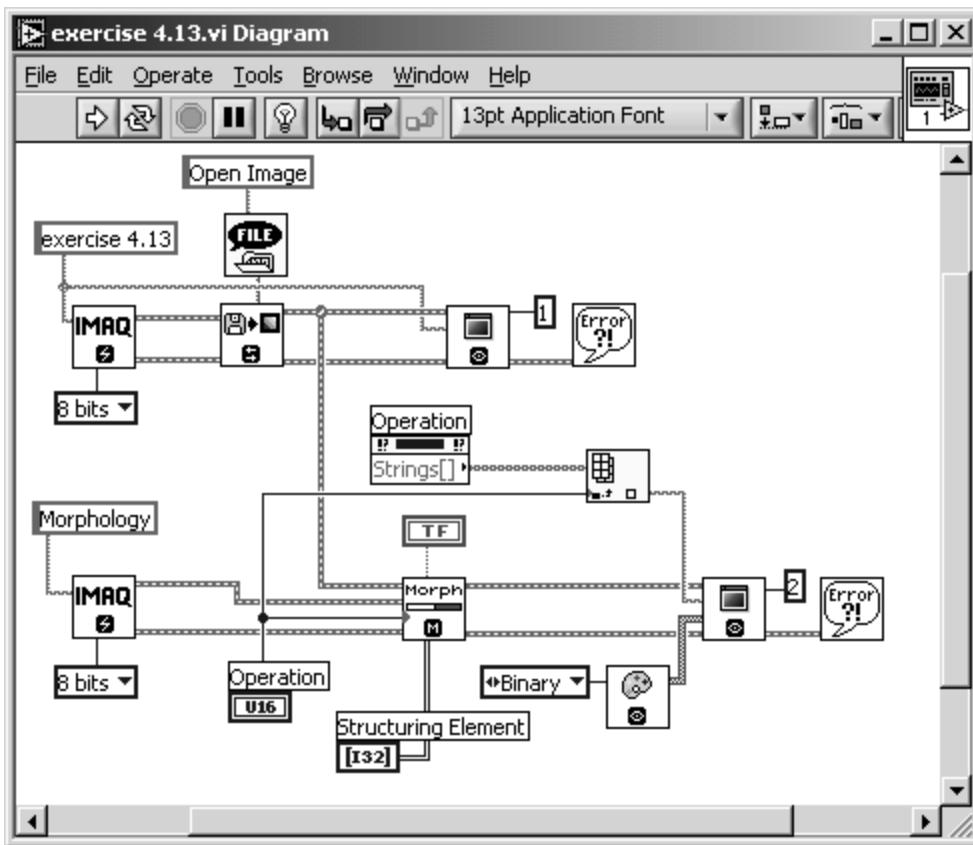


Figure 4.51. Diagram of [Exercise 4.13](#)



A special notation for morphology operations can be found in [13], which shows the structure of concatenated functions very clearly. If $/$ is an image and \mathcal{M} is the structuring element (mask), the erosion (operator \ominus) is defined as

Equation 4.26

$$\text{erosion}(\mathcal{I}) = \mathcal{I} \ominus \mathcal{M} = \cap_{a \in \mathcal{M}} \mathcal{I}_{-a} ,$$

where $/_a$ indicates a basic shift operation in direction of the element a of \mathcal{M} . $/_{-a}$ would indicate the reverse shift operation [13].

Dilation adds pixels (sets their values to 1) to the border of particles or objects. According to [13], the dilation (operator \oplus) is defined as

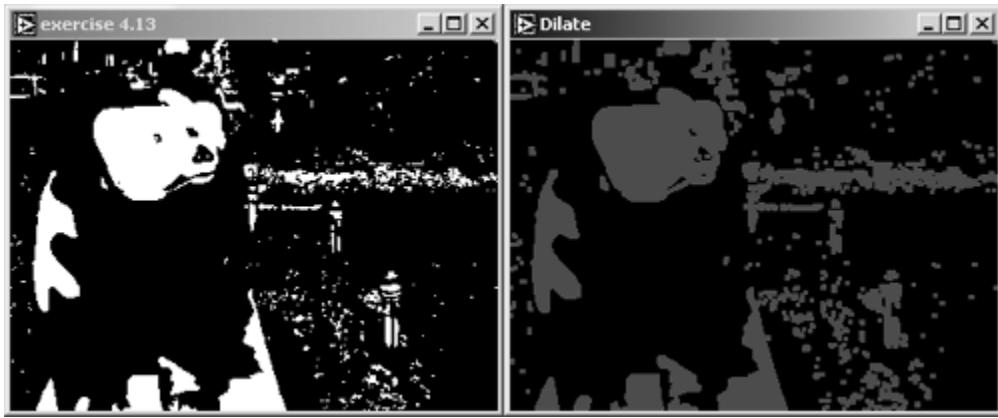
Equation 4.27

$$\text{dilation}(\mathcal{I}) = \mathcal{I} \oplus \mathcal{M} = \cup_{a \in \mathcal{M}} \mathcal{I}_a .$$

Again, the structuring element is centered on the pixel (value) s_0 : *If the value of at least one pixel of the structuring element is equal to 1, s_0 is set to 1, else s_0 is set to 0* [13].

[Figure 4.52](#) shows the result of a simple dilation. Note that some "holes" in objects (the bear's eyes for example) are closed or nearly closed; very small objects get bigger. You can find more information about the impact of different structuring elements in [4].

Figure 4.52. Morphology: Dilation Result



Opening and Closing

Opening and closing are functions that combine erosion and dilation functions. The *opening* function is defined as an erosion followed by a dilation using the same structuring element. Mathematically, the opening function can be described by

Equation 4.28

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

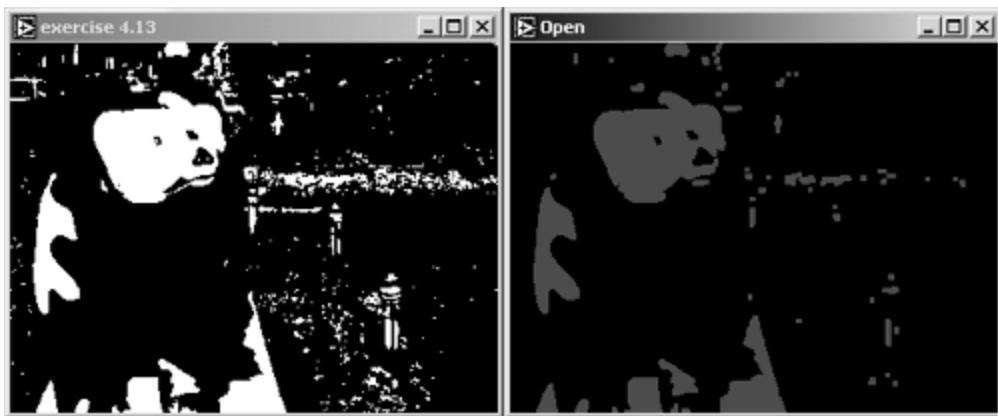
or, using the operator \circ ,

Equation 4.29

$$\mathcal{I} \circ \mathcal{M} = (\mathcal{I} \ominus \mathcal{M}) \oplus \mathcal{M}$$

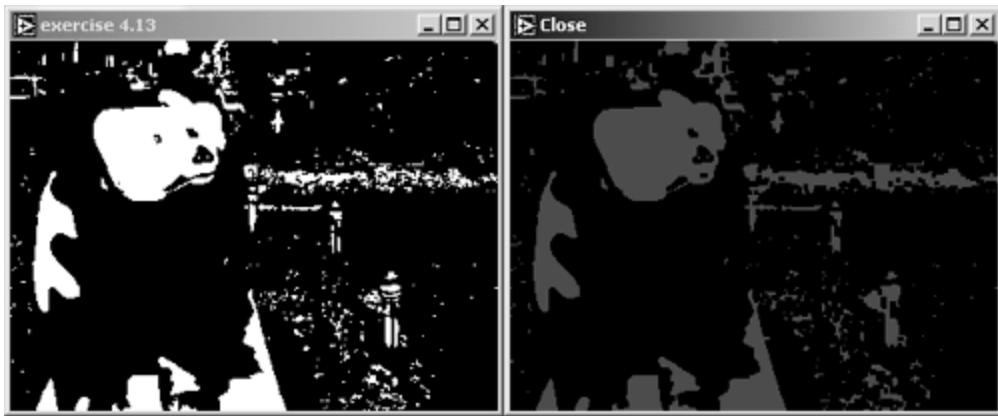
[Figure 4.53](#) shows the result of an opening function. It is easy to see that objects of the original image are not significantly changed by opening; however, small particles are removed. The reason for this behavior is that borders, which are removed by the erosion, are restored by the dilation function. Naturally, if a particle is so small that it is removed totally by the erosion, it will not be restored.

Figure 4.53. Morphology: Opening Result



The result of a *closing* function is shown in [Figure 4.54](#). The closing function is defined as a dilation followed by an erosion using the same structuring element:

Figure 4.54. Morphology: Closing Result



Equation 4.30

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

or, using the operator \bullet ,

Equation 4.31

$$\mathcal{I} \bullet \mathcal{M} = (\mathcal{I} \oplus \mathcal{M}) \ominus \mathcal{M}$$

Again, objects of the original image are not significantly changed. Complementarily to the opening function, small holes in particles can be closed.

Both opening and closing functions use the duality of erosion and dilation operations. The proof for this duality can be found in [13].

Proper Opening and Proper Closing

IMAQ Vision provides two additional functions called proper opening and proper closing. *Proper opening* is defined as a combination of two opening functions and one closing function combined with the original image:

Equation 4.32

$$\text{proper opening}(\mathcal{I}) = \mathcal{I} \text{ AND } \text{opening}(\text{closing}(\text{opening}(\mathcal{I})))$$

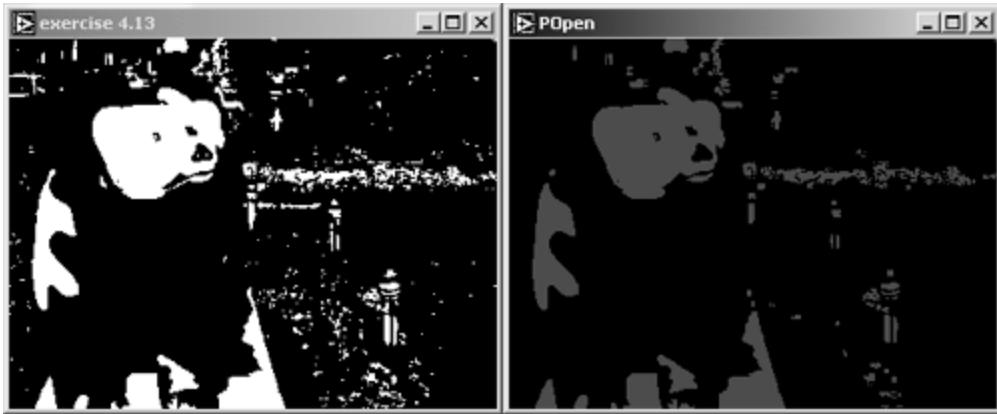
or

Equation 4.33

$$\mathcal{I} \circ_{\text{prop}} \mathcal{M} = \mathcal{I} \cap (((\mathcal{I} \circ \mathcal{M}) \bullet \mathcal{M}) \circ \mathcal{M})$$

As [Figure 4.55](#) shows, the proper opening function leads to a smoothing of the borders of particles. Similarly to the opening function, small particles can be removed.

Figure 4.55. Morphology: Proper Opening Result



Proper closing is the combination of two closing functions combined with one opening function and the original image:

Equation 4.34

$$\text{proper closing}(\mathcal{I}) = \mathcal{I} \text{ OR } \text{closing}(\text{opening}(\text{closing}(\mathcal{I})))$$

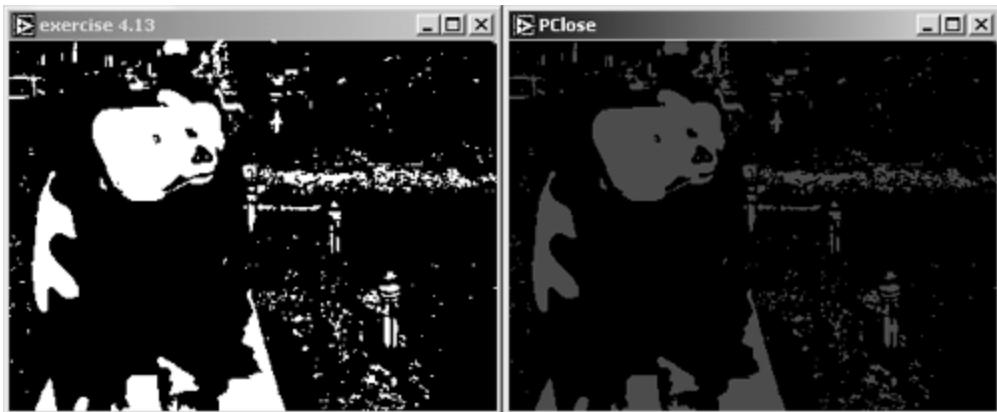
or

Equation 4.35

$$\mathcal{I} \bullet_{\text{prop}} \mathcal{M} = \mathcal{I} \cup (((\mathcal{I} \bullet \mathcal{M}) \circ \mathcal{M}) \bullet \mathcal{M}) \quad .$$

Proper closing smooths the contour of holes if they are big enough so that they will not be closed. [Figure 4.56](#) shows that there is only a little difference between the original image and the proper closing result.

Figure 4.56. Morphology: Proper Closing Result



Hit-Miss Function

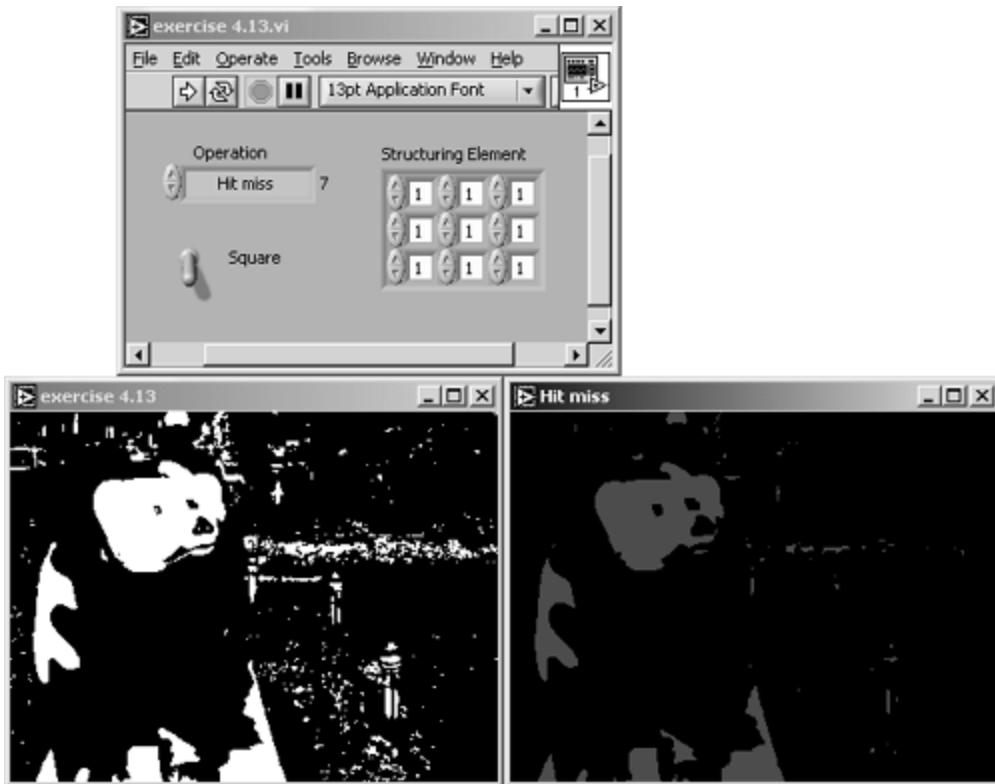
This function is the first simple approach to pattern matching techniques. In an image, each pixel that has exactly the neighborhood defined in the structuring element (mask) is kept; all others are removed. In other words: *If the value of each surrounding pixel is identical to the value of the structuring element, the center pixel's 0 is set to 1; else s 0 is set to 0.*

The hit-miss function typically produces totally different results, depending on the structuring element. For example, [Figure 4.57](#) shows the result of the hit-miss function with the

$$\mathcal{M} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}.$$

structuring element All pixels in the inner area of objects match \mathcal{M} , therefore, this function is quite similar to an erosion function.

Figure 4.57. Hit-Miss Result with Structuring Element That Is All 1s



$$\mathcal{M} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

[Figure 4.58](#) shows the result when the structuring element is used. This mask extracts pixels that have no other pixels in their 8-connectivity neighborhood.

Figure 4.58. Hit-Miss Result with Structuring Element That Is All 0s



It is easy to imagine that the hit-miss function can be used for a number of matching operations, depending on the shape of the structuring element. For example, a more complex

$$\mathcal{M} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix},$$

mask, like will return no pixels at all, because none match this mask.

More mathematic details can be found in [13]; we use the operator \odot for the hit-miss function: [9]

[9] Davies [13] uses a different one; I prefer \odot because it reminds me of the pixel separation in Figure 4.58.

Equation 4.36

$$hitmiss(\mathcal{I}) = \mathcal{I} \odot \mathcal{M} .$$

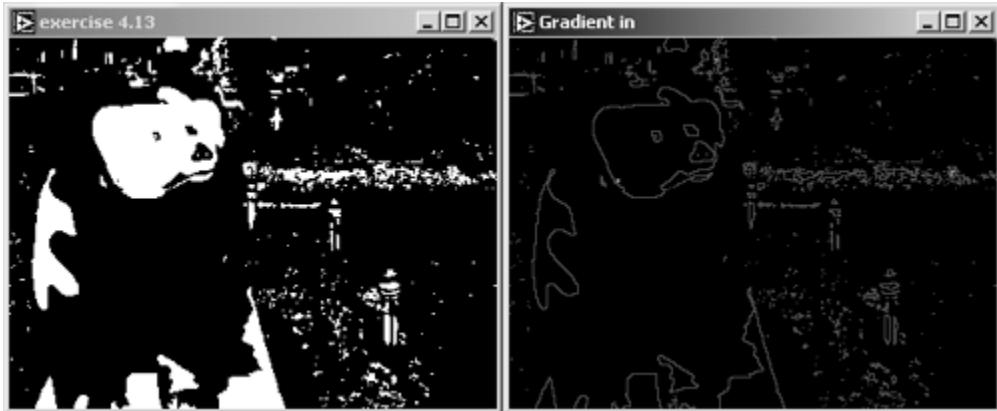
Gradient Functions

The gradient functions provide information about pixels eliminated by an erosion or added by a dilation. The *inner gradient* (or *internal edge*) function subtracts the result of an erosion from the original image, so the pixels that are removed by the erosion remain in the resulting image (see Figure 4.59 for the results):

Equation 4.37

$$inner\ gradient(\mathcal{I}) = \mathcal{I} - erosion(\mathcal{I}) = \mathcal{I} \text{ XOR } erosion(\mathcal{I}) .$$

Figure 4.59. Inner Gradient (Internal Edge) Result

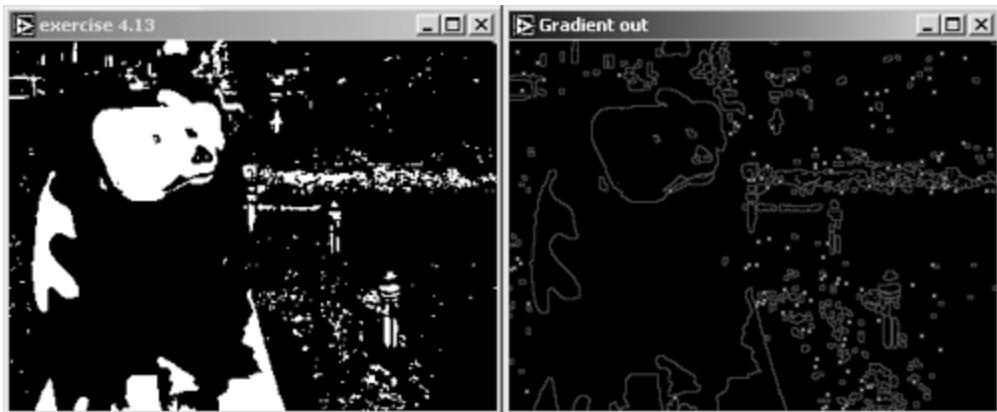


The *outer gradient* (or *external edge*) function subtracts the original image from a dilation result. [Figure 4.60](#) shows that only the pixels that are added by the dilation remain in the resulting image:

Equation 4.38

$$\text{outer gradient}(\mathcal{I}) = \text{dilation}(\mathcal{I}) - \mathcal{I} = \mathcal{I} \text{ XOR } \text{dilation}(\mathcal{I}) .$$

Figure 4.60. Outer Gradient (External Edge) Result

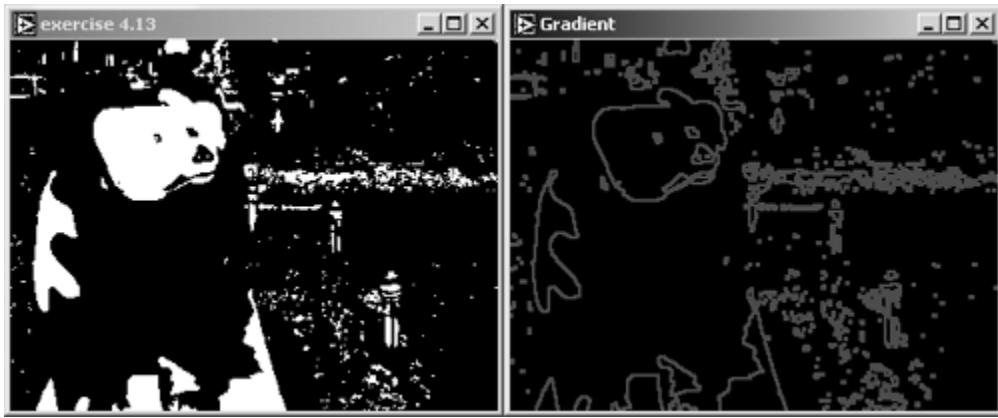


Finally, the *gradient* function adds the results of the inner and outer gradient functions (see [Figure 4.61](#) for the result):

Equation 4.39

$$\begin{aligned}\text{gradient}(\mathcal{I}) &= \text{inner gradient}(\mathcal{I}) + \text{outer gradient}(\mathcal{I}) \\ &= \text{inner gradient}(\mathcal{I}) \text{ OR } \text{outer gradient}(\mathcal{I}) .\end{aligned}$$

Figure 4.61. Morphology: Gradient Result



Thinning and Thickening

These two functions use the result of the hit-miss function. The *thinning* function subtracts the hit-miss result of an image from the image itself:

Equation 4.40

$$\text{thinning}(\mathcal{I}) = \mathcal{I} - \text{hitmiss}(\mathcal{I}) = \mathcal{I} \text{ XOR } (\mathcal{I} \odot \mathcal{M}) ,$$

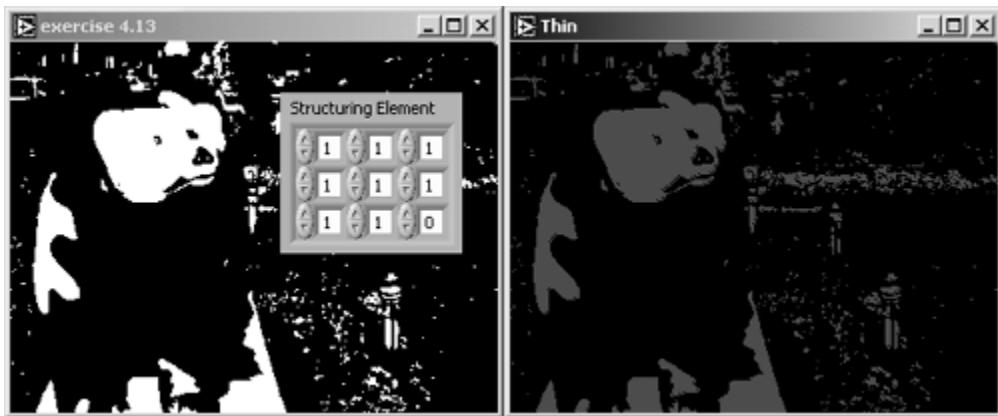
which means that certain pixels that match the mask \mathcal{M} are eliminated. Similarly to the hit-miss function itself, the result depends strongly on the structuring element.[\[10\]](#)[Figure 4.62](#) shows

$$\mathcal{M} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}.$$

the result for the structuring element

[\[10\]](#) According to [\[4\]](#), the thinning function does not provide appropriate results if $s_0 = 0$. On the other hand, the thickening function does not work if $s_0 = 1$.

Figure 4.62. Morphology: Thinning Result



The *thickening* function adds the hit-miss result of an image to the image itself (pixels matching the mask \mathcal{M} are added to the original image):

Equation 4.41

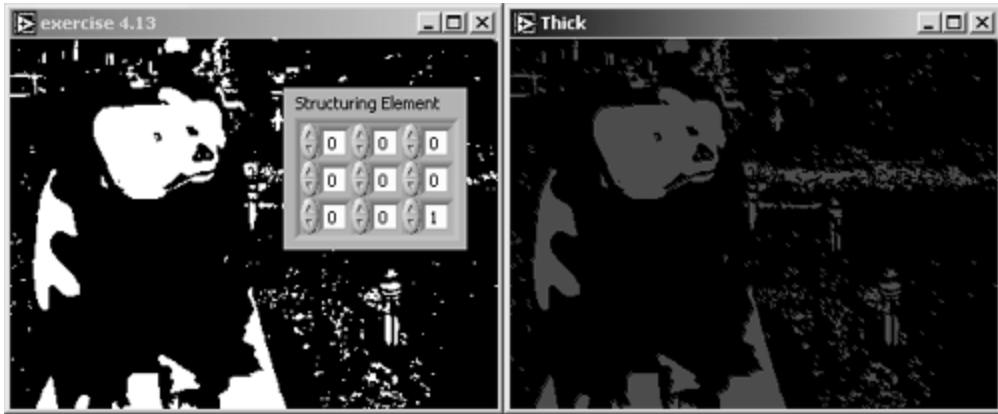
$$\text{thickening}(\mathcal{I}) = \mathcal{I} + \text{hitmiss}(\mathcal{I}) = \mathcal{I} \text{ OR } (\mathcal{I} \odot \mathcal{M}) .$$

[Figure 4.63](#) shows the result of a thickening operation with the structuring element

$$\mathcal{M} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Both functions can be used to smooth the border of objects and to eliminate single pixels (thinning) or small holes (thickening).

Figure 4.63. Morphology: Thickening Result



Auto-Median Function

Finally, the *auto-median* function generates simpler particles with fewer details by using a sequence of opening and closing functions, according to

Equation 4.42

$$\text{automedian}(\mathcal{I}) = \text{opening}(\text{closing}(\text{opening}(\mathcal{I}))) \\ \text{AND } \text{closing}(\text{opening}(\text{closing}(\mathcal{I})))$$

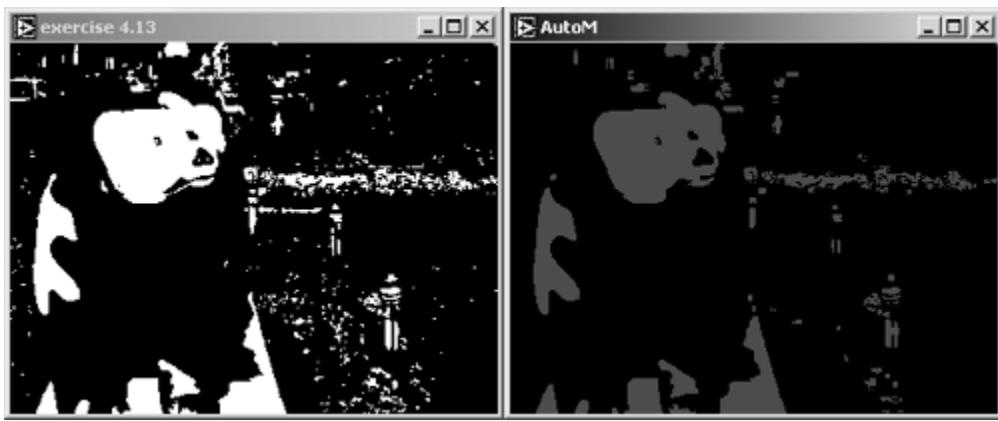
or, using the operator \otimes ,

Equation 4.43

$$\mathcal{I} \otimes \mathcal{M} = (((\mathcal{I} \circ \mathcal{M}) \bullet \mathcal{M}) \circ \mathcal{M}) \cap (((\mathcal{I} \bullet \mathcal{M}) \circ \mathcal{M}) \bullet \mathcal{M}) .$$

[Figure 4.64](#) shows the result of applying the auto-median function to our binary bear image.

Figure 4.64. Morphology: Auto-Median Result



Particle Filtering

For the next functions, which can be found under the Adv. Morphology menu in IMAQ Vision Builder, we have to slightly modify our [Exercise 4.13](#) because the `IMAQ Morphology` function does not provide them.

The `removeparticle` function detects all particles that are resistant to a certain number of erosions. These particles can be removed (similarly to the function of a *low-pass* filter), or these particles are kept and all others are removed (similarly to the function of a *high-pass* filter).

Exercise 4.14: Removing Particles.

Replace the function `IMAQ Morphology` in [Exercise 4.13](#) with `IMAQ RemoveParticle`. Provide controls for setting the shape of the structuring element (square or hexagonal), the connectivity (4 or 8), and the specification of low-pass or high-pass functionality. See [Figure 4.65](#), which also shows the result of a low-pass filtering, and [Figure 4.66](#).

Figure 4.65. Remove Particle: Low Pass

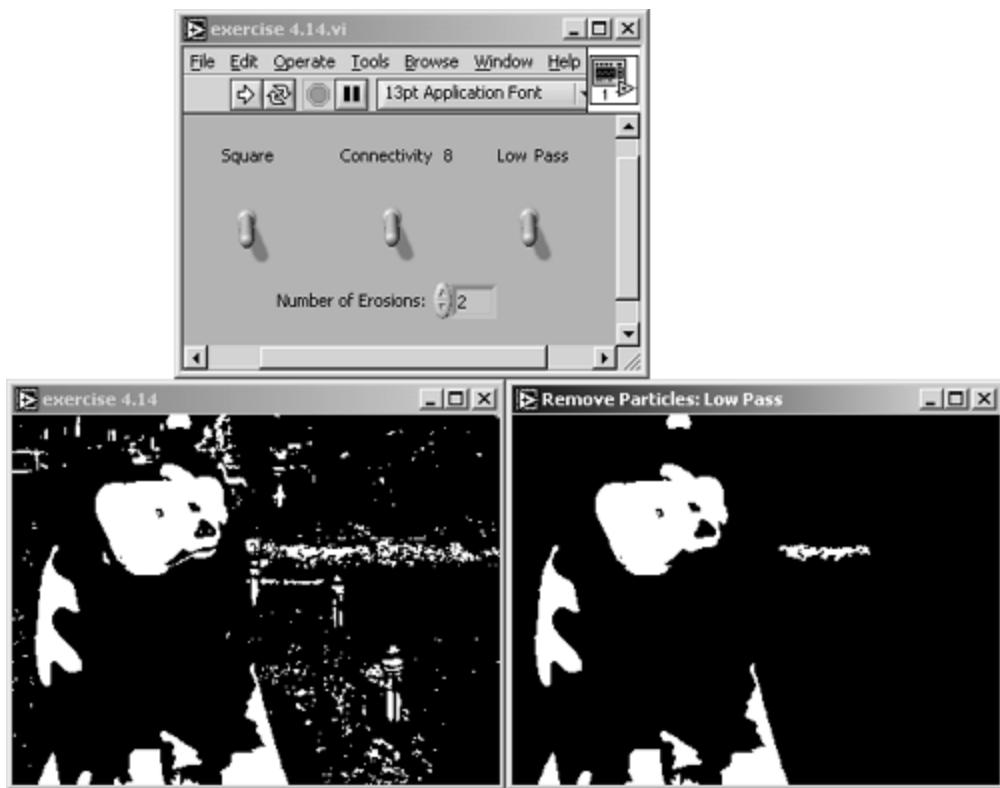
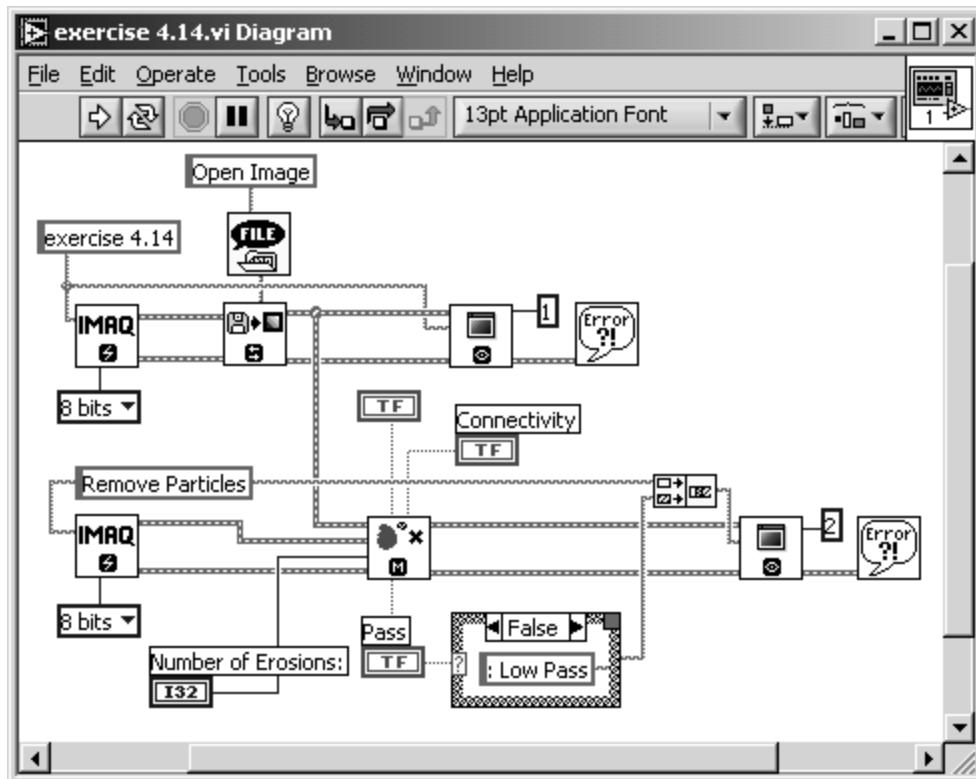


Figure 4.66. Diagram of [Exercise 4.14](#)



[Figure 4.67](#) shows the result of high-pass filtering with the same settings. It is evident that the sum of both result images would lead again to the original image.

Figure 4.67. Remove Particle: High Pass



The next function, `IMAQ RejectBorder`, is quite simple: The function removes particles touching the border of the image. The reason for this is simple as well: Particles that touch the border of an image may have been truncated by the choice of the image size. In case of further particle analysis, they would lead to unreliable results.

Exercise 4.15: Rejecting Border Particles.

Replace the function `IMAQ Morphology` in [Exercise 4.13](#) or `IMAQ RemoveParticle` in [Exercise 4.14](#) by `IMAQ RejectBorder`. The only necessary control is for connectivity (4 or 8). See [Figures 4.68](#) and [4.69](#).

Figure 4.68. Particles Touching the Border Are Removed

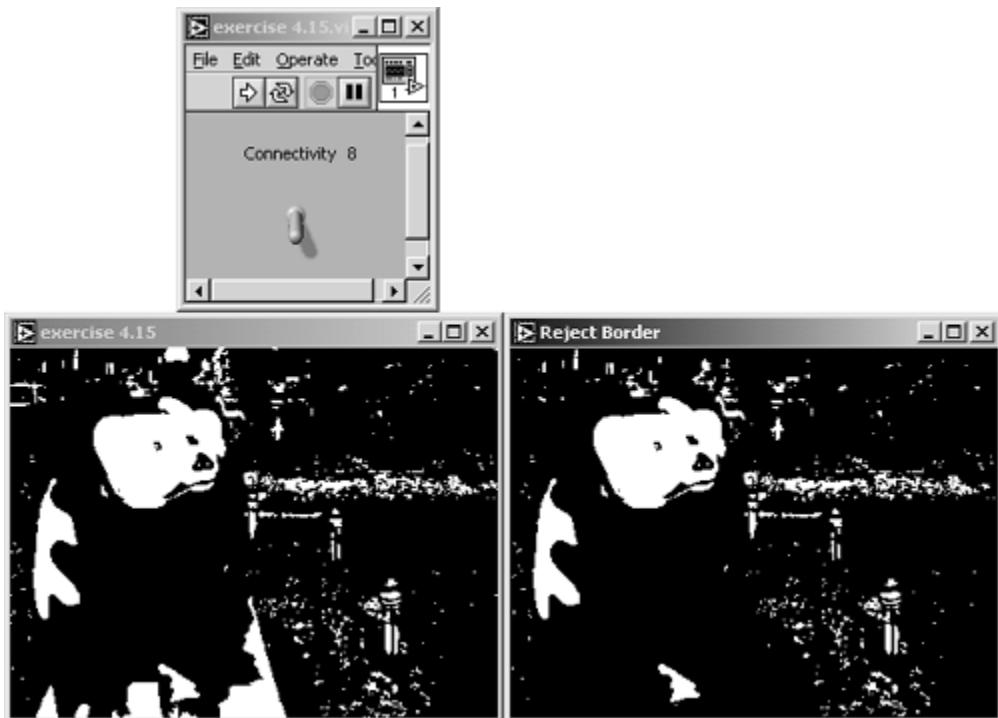
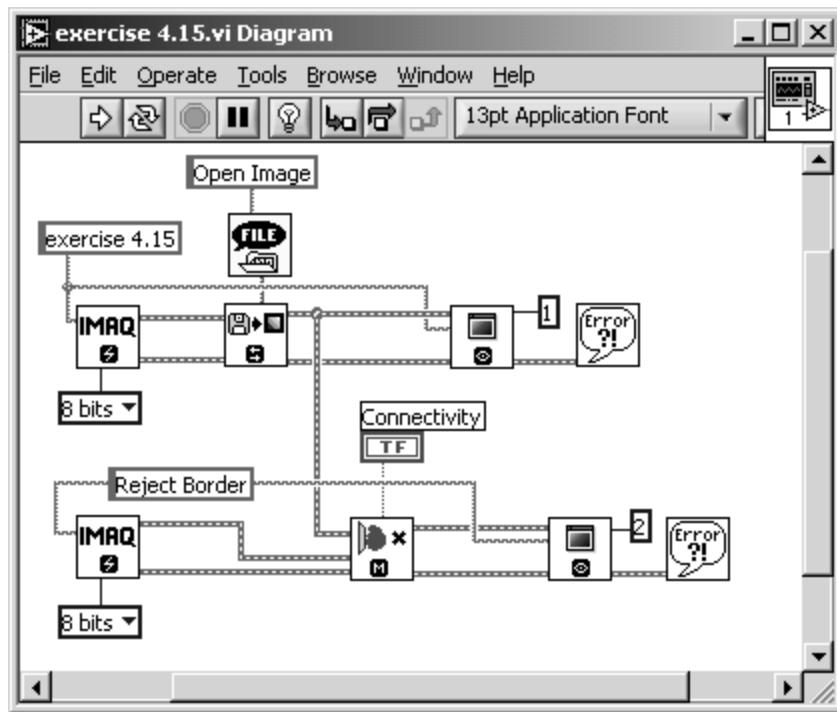


Figure 4.69. Diagram of [Exercise 4.15](#)



The next function—**IMAQ ParticleFilter**—is much more complex and much more powerful. Using this function, you can filter all particles of an image according to a large number of criteria. Similarly to the previous functions, the particles matching the criteria can be kept or removed, respectively.

Exercise 4.16: Particle Filtering.

Replace the function **IMAQ Morphology** in [Exercise 4.13](#) with **IMAQ ParticleFilter**. The control for the selection criteria is an array of clusters, containing the criteria names, lower and upper level of the criteria values, and the selection, if the specified interval is to be included or excluded.

Because of the array data type of this control, you can define multiple criteria or multiple regions for the particle filtering operation. [Figure 4.70](#) shows an example in which the filtering criterion is the starting x coordinate of the particles. All particles starting at an x coordinate greater than 150 are removed. See also [Figure 4.71](#).

[Figure 4.70. Particle Filtering by \$x\$ Coordinate](#)

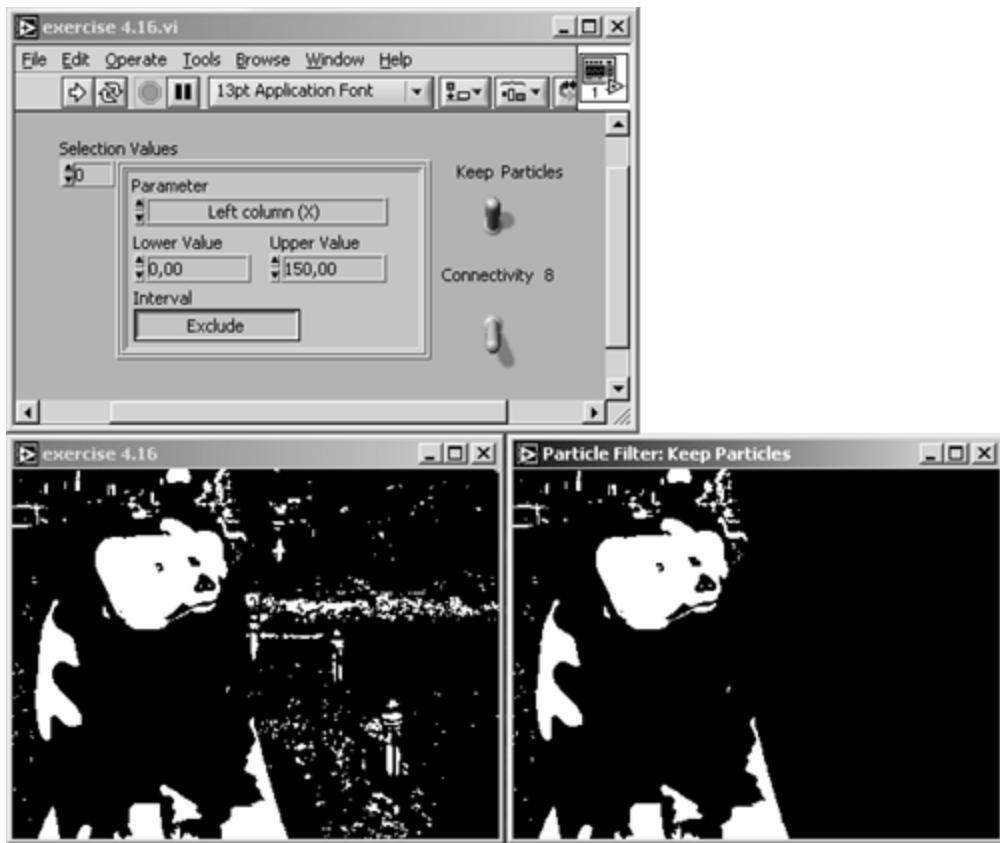
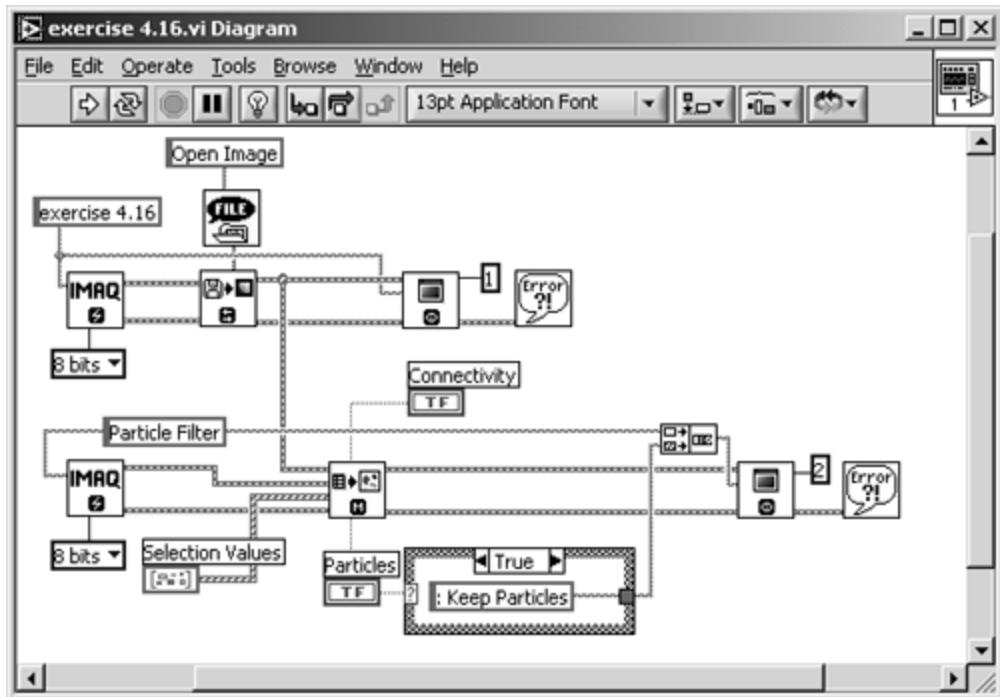


Figure 4.71. Diagram of [Exercise 4.16](#)



The following list is taken from [5] and briefly describes each criterion that can be selected by the corresponding control. The criteria are also used in functions like [IMAQComplexMeasure](#) in [Chapter 5](#).

- *Area (pixels)*: Surface area of particle in pixels.
- *Area (calibrated)*: Surface area of particle in user units.
-

- *Number of holes*
- *Holes area*: Surface area of holes in user units.
- *Total area*: Total surface area (holes and particles) in user units.
- *Scanned area*: Surface area of the entire image in user units.
- *Ratio area/scanned area %*: Percentage of the surface area of a particle in relation to the scanned area.
- *Ratio area/total area %*: Percentage of particle surface in relation to the total area.
- *Center of mass (X)*: x coordinate of the center of gravity.
- *Center of mass (Y)*: y coordinate of the center of gravity.
- *Left column (X)*: Left x coordinate of bounding rectangle.
- *Upper row (Y)*: Top y coordinate of bounding rectangle.
- *Right column (X)*: Right x coordinate of bounding rectangle.
- *Lower row (Y)*: Bottom y coordinate of bounding rectangle.
- *Width*: Width of bounding rectangle in user units.
- *Height*: Height of bounding rectangle in user units.
- *Longest segment length*: Length of longest horizontal line segment.
- *Longest segment left column (X)*: Leftmost x coordinate of longest horizontal line segment.
- *Longest segment row (Y)*: y coordinate of longest horizontal line segment.
- *Perimeter*: Length of outer contour of particle in user units.
- *Hole perimeter*: Perimeter of all holes in user units.
- *SumX*: Sum of the x axis for each pixel of the particle.
- *SumY*: Sum of the y axis for each pixel of the particle.
- *SumXX*: Sum of the x axis squared for each pixel of the particle.
- *SumYY*: Sum of the y axis squared for each pixel of the particle.
- *SumXY*: Sum of the x axis and y axis for each pixel of the particle.
- *Corrected projection x*: Projection corrected in x .
- *Corrected projection y*: Projection corrected in y .
- *Moment of inertia / xx*: Inertia matrix coefficient in xx .
- *Moment of inertia / yy*: Inertia matrix coefficient in yy .
- *Moment of inertia / xy*: Inertia matrix coefficient in xy .
- *Mean chord X*: Mean length of horizontal segments.
- *Mean chord Y*: Mean length of vertical segments.
-

- *Max intercept*: Length of longest segment.
- *Mean intercept perpendicular*: Mean length of the chords in an object perpendicular to its max intercept.
- *Particle orientation*: Direction of the longest segment.
- *Equivalent ellipse minor axis*: Total length of the axis of the ellipse having the same area as the particle and a major axis equal to half the max intercept.
- *Ellipse major axis*: Total length of major axis having the same area and perimeter as the particle in user units.
- *Ellipse minor axis*: Total length of minor axis having the same area and perimeter as the particle in user units.
- *Ratio of equivalent ellipse axis*: Fraction of major axis to minor axis.
- *Rectangle big side*: Length of the large side of a rectangle having the same area and perimeter as the particle in user units.
- *Rectangle small side*: Length of the small side of a rectangle having the same area and perimeter as the particle in user units.
- *Ratio of equivalent rectangle sides*: Ratio of rectangle big side to rectangle small side.
- *Elongation factor*: Max intercept/mean perpendicular intercept.
- *Compactness factor*: Particle area/(height x width).
- *Heywood circularity factor*: Particle perimeter/perimeter of circle having the same area as the particle.
- *Type factor*: A complex factor relating the surface area to the moment of inertia.
- *Hydraulic radius*: Particle area/particle perimeter.
- *Waddel disk diameter*: Diameter of the disk having the same area as the particle in user units.
- *Diagonal*: Diagonal of an equivalent rectangle in user units.

Fill Holes and Convex

These two functions can be used for correcting the shape of objects, which should be circular but which show some enclosures or holes in the original (binary) image. These holes may result from a threshold operation with a critical setting of the threshold value, for example.

Exercise 4.17: Filling Holes.

Modify [Exercise 4.16](#) by adding `IMAQFillHole` after `IMAQParticleFilter`. We use particle filtering here, so only one particle with holes remains. If you use the criterion Area (pixels), set the lower value to 0 and the upper value to 1500, and exclude this interval, then only the bear's head with the eyes as holes remains.

You can use a case structure to decide whether the holes should be filled or not.

[Figure 4.72](#) shows all three steps together: the original image, the filtered image with only one particle remaining, and the result image with all holes filled. See [Figure 4.73](#)

for the diagram.

Figure 4.72. Filling Holes in Particles

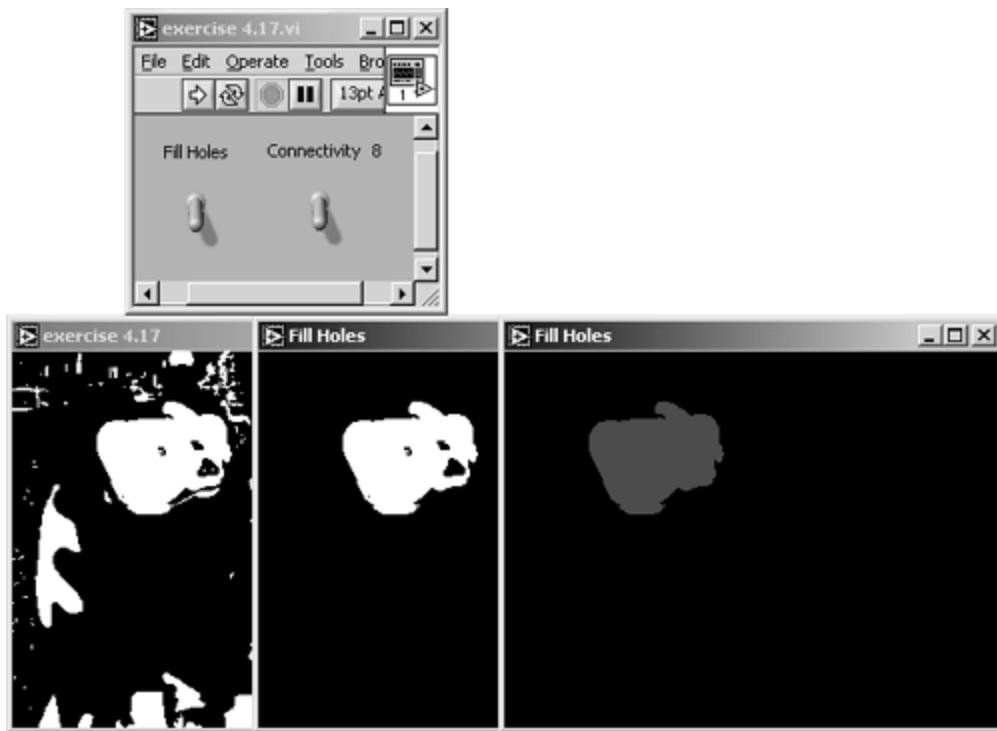
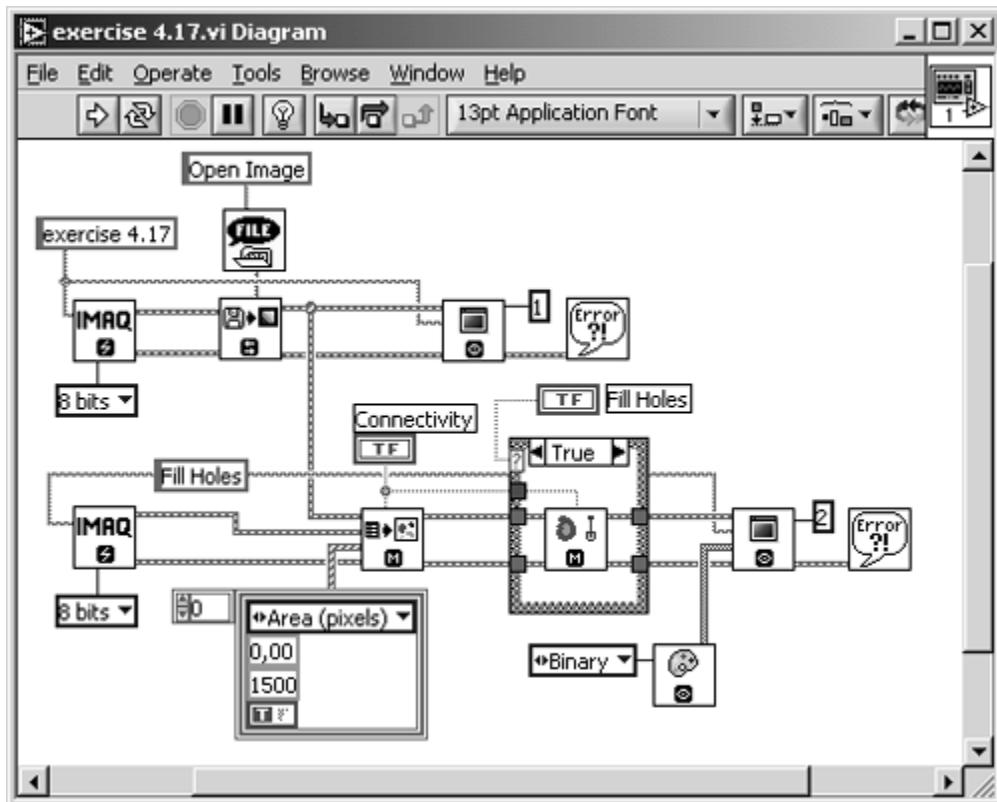


Figure 4.73. Diagram of [Exercise 4.17](#)



Exercise 4.18: Creating Convex Particles.

Modify [Exercise 4.17](#) by replacing `IMAQ FillHole` with `IMAQ Convex`. Set the lower value of the filtering parameters to 1300 and the upper value to 1400 and include this interval. This leaves only a single particle near the left image border, showing a curved outline.

`IMAQConvex` calculates a convex envelope around the particle and fills all pixels within this outline with 1. Again, [Figure 4.74](#) shows all three steps together: the original image, the filtered image with only one particle remaining, and the resulting image with a convex outline. See [Figure 4.75](#) for the diagram.

Figure 4.74. IMAQ Convex Function

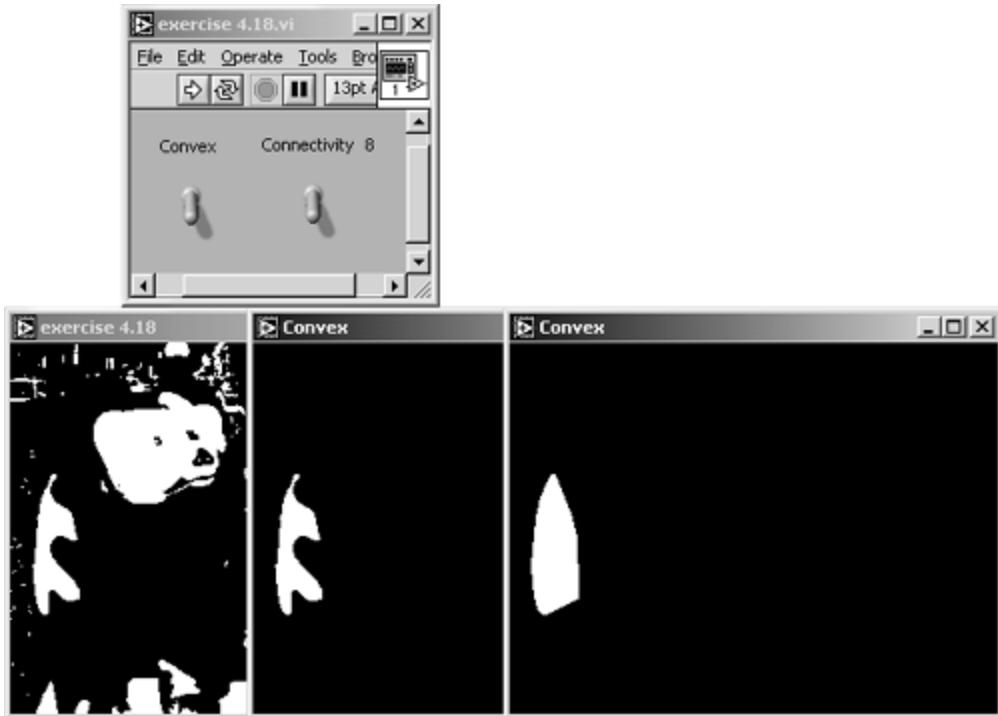
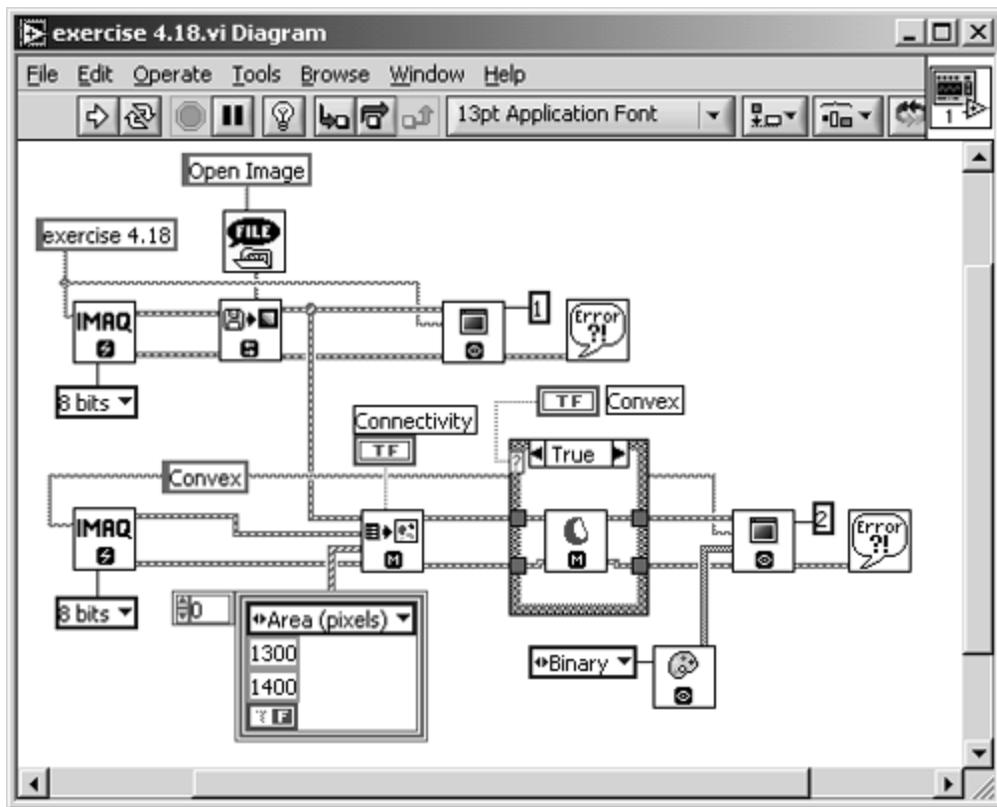


Figure 4.75. Diagram of [Exercise 4.18](#)



If you change the parameters of the selection criteria in [Exercise 4.18](#) so that more than one particle remains, you will find that our exercise does not work: It connects all existing particles. The reason for this behavior is that usually the particles in a binary image have to be labeled so that the subsequent functions can distinguish between them. We discuss labeling of particles in a future exercise.

Separation and Skeleton Functions

A simple function for the separation of particles is [IMAQSeparation](#). It performs a predefined number of erosions and separates objects that would be separated by these erosions. Afterwards, the original size of the particles is restored.

Exercise 4.19: Separating Particles.

Take one of the previous simple exercises, for example, [Exercise 4.14](#), and replace its IMAQ Vision function with [IMAQ Separation](#). You will need controls for the structuring element, the shape of the structuring element, and the number of erosions.

The separation result is not easily visible in our example image. Have a look at the particle with the curved outline we used in the "convex" exercise. In [Figure 4.76](#), this particle shows a small separation line in the resulting image. This is the location where the erosions would have split the object. See [Figure 4.77](#) for the diagram.

Figure 4.76. Separation of Particles

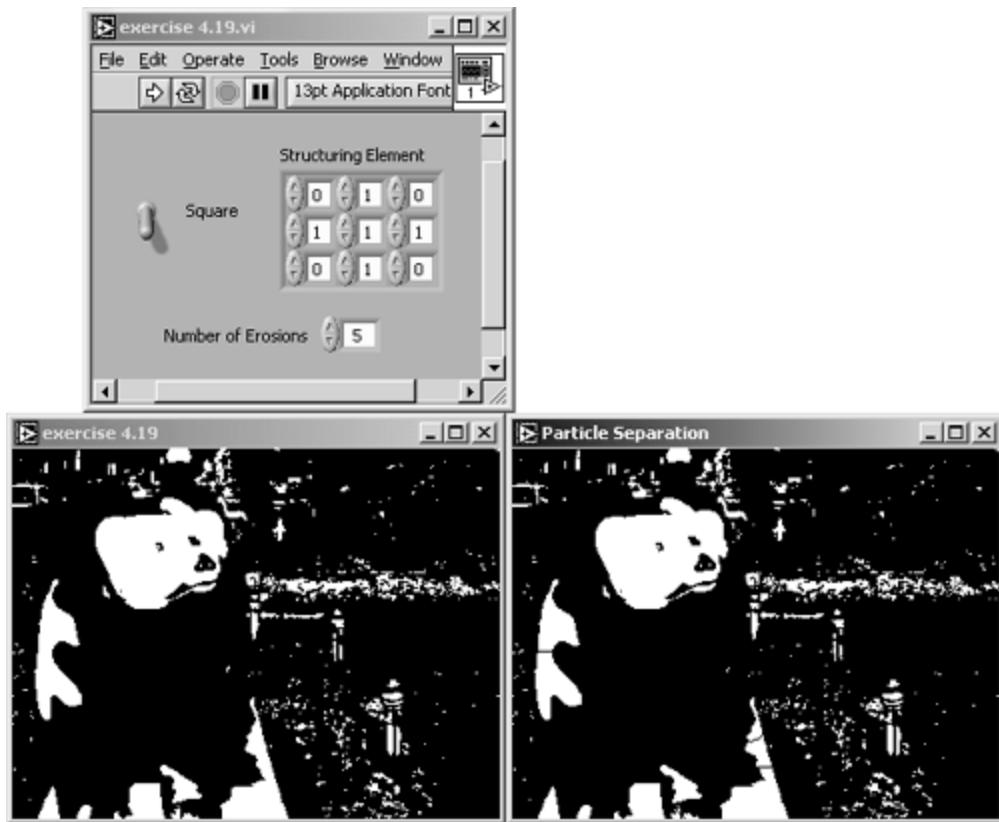
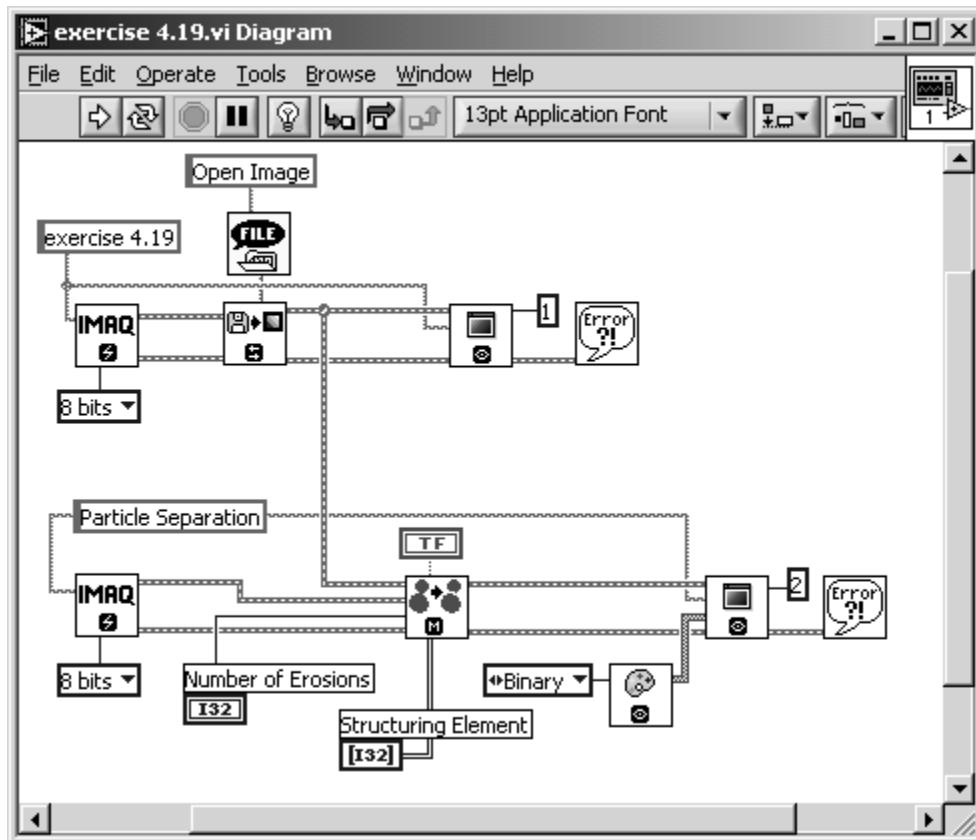


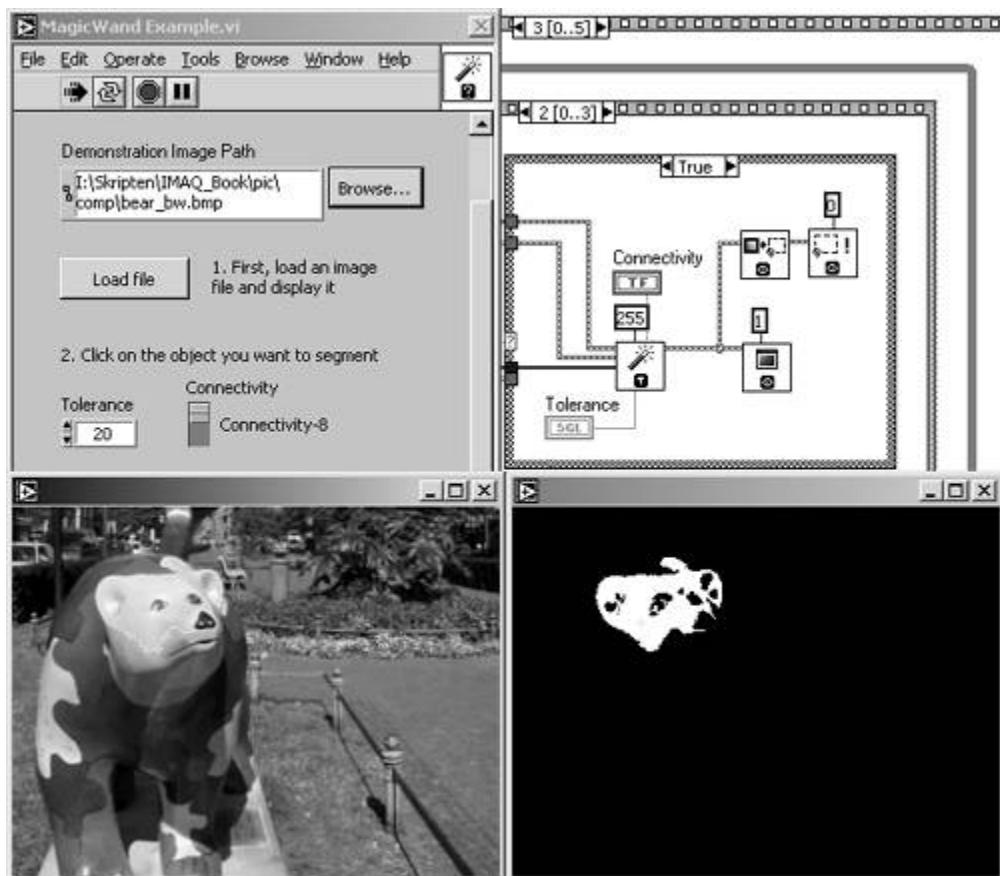
Figure 4.77. Diagram of [Exercise 4.19](#)



The next exercise is taken directly from IMAQ's own examples: [IMAQ MagicWand](#). Just open the help window with the VI [IMAQ MagicWand](#) selected (you can find it in the Processing submenu) and click on MagicWand example; the example shown in [Figure 4.78](#) is displayed.

Here, you can click on a region of a *gray-scaled* image, which most likely would be a particle after a thresholding operation. The result is displayed in a binary image and shows the selected object. Additionally, the object is highlighted in the original (gray-level) image by a yellow border. [Figure 4.78](#) also shows the use of the **IMAQ MagicWand** function in the respective VI diagram.

Figure 4.78. IMAQ MagicWand: Separating Objects from the Background (National Instruments Example)



The last three functions in this section are similar to the analysis VIs we discuss in [Chapter 5](#); they already supply some information about the particle structure of the image. Prepare the following exercise first.

Exercise 4.20: Skeleton Images.

Again, take one of the previous simple exercises, for example, [Exercise 4.14](#), and replace its IMAQ Vision function by **IMAQ Skeleton**. You will need only a single control for the skeleton mode; with that control you select the Skeleton L, Skeleton M, or Skiz function. See [Figure 4.79](#), showing a Skeleton L function, and [Figure 4.80](#) for the results.

Figure 4.79. L-Skeleton Function

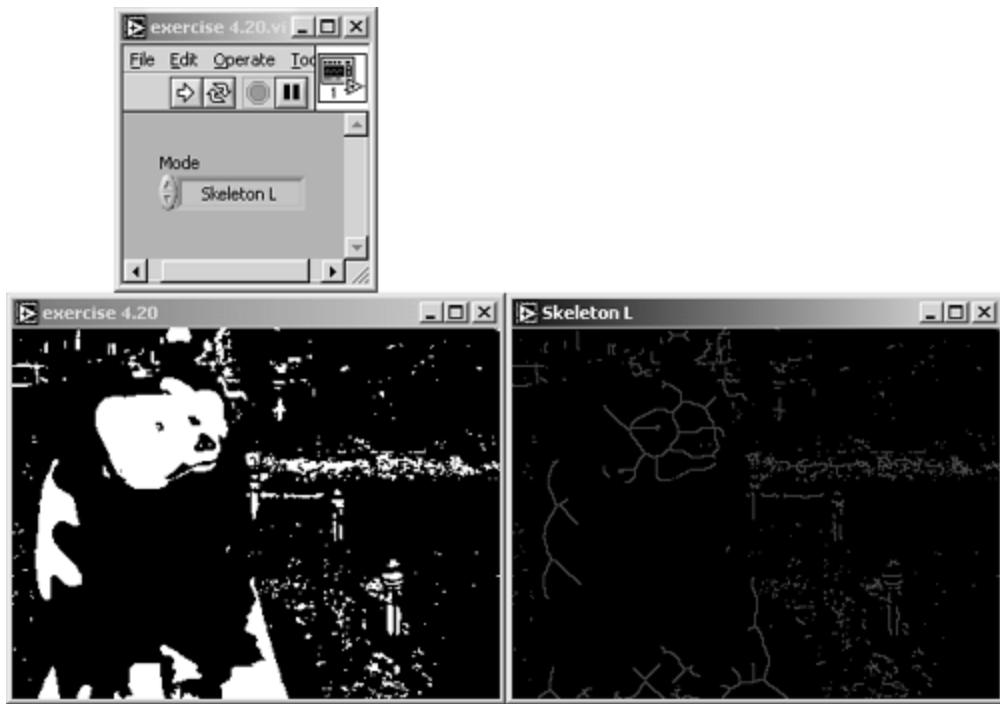
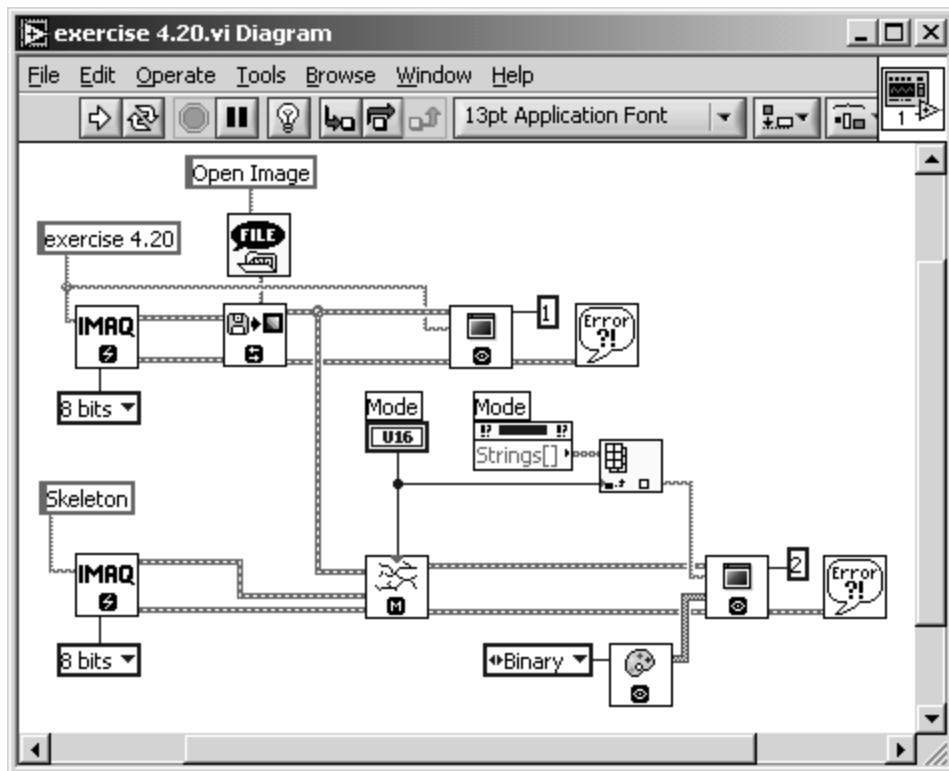


Figure 4.80. Diagram of [Exercise 4.20](#)



All *skeleton* functions work according to the same principle: they apply combinations of morphology operations (most of them are thinning functions) until the width of each particle is equal to 1 (called the *skeleton line*). Other declarations for a skeleton function are these:

- The skeleton line shall be in (approximately) the center area of the original particle.
- The skeleton line shall represent the structure and the shape of the original particle. This means that the skeleton function of a coherent particle can only lead to a coherent skeleton line.

$$\mathcal{M} = \begin{pmatrix} 0 & d & 1 \\ 0 & 1 & 1 \\ 0 & d & 1 \end{pmatrix}$$

The *L-skeleton* function is based on the structuring element (see also [5]); the *d*' in this mask means "don't care"; the element can be either 0 or 1 and produces the result shown in [Figure 4.79](#).

[Figure 4.81](#) shows the result of the *M-skeleton* function. Typically, the M-skeleton leads to skeleton lines with more branches, thus filling the area of the original particles with a higher

$$\mathcal{M} = \begin{pmatrix} d & d & 1 \\ 0 & 1 & 1 \\ d & d & 1 \end{pmatrix}$$

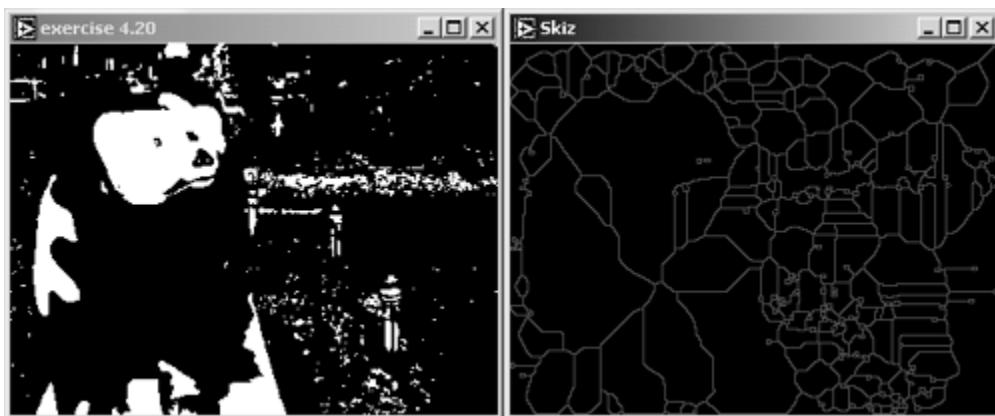
amount. The M-skeleton function uses the structuring element [5].

Figure 4.81. M-Skeleton Function



The *skiz* function ([Figure 4.82](#)) is slightly different. Imagine an L-skeleton function applied not to the objects but to the background. You can try this with our [Exercise 4.20](#). Modify it by adding an inverting function and compare the result of the Skiz function to the result of the L-skeleton function of the original image.

Figure 4.82. Skiz Function



Gray-Level Morphology

We return to the processing of gray-scaled images in the following exercises. Some morphology functions work not only with binary images, but also with images scaled according to the 8-bit gray-level set.

Exercise 4.21: Gray Morphology Functions.

Modify [Exercise 4.13](#) by replacing the function `IMAQMorphology` with `IMAQGrayMorphology`. Add a digital control for Number of iterations and prepare your VI for the use of larger structuring elements, for example, 5×5 and 7×7 .

You will need to specify the number of iterations for the erosion and dilation examples; larger structuring elements can be used in opening, closing, proper opening, and proper closing examples. See [Figures 4.83](#) and [4.84](#) for results.

Gray-Level Erosion and Dilation

[Figure 4.83](#) shows a gray-level erosion result after eight iterations (we need this large number of iterations here because significant results are not so easily visible in gray-level morphology). As an alternative, enlarge the size of your structuring element.

Figure 4.83. Gray Morphology: Erosion

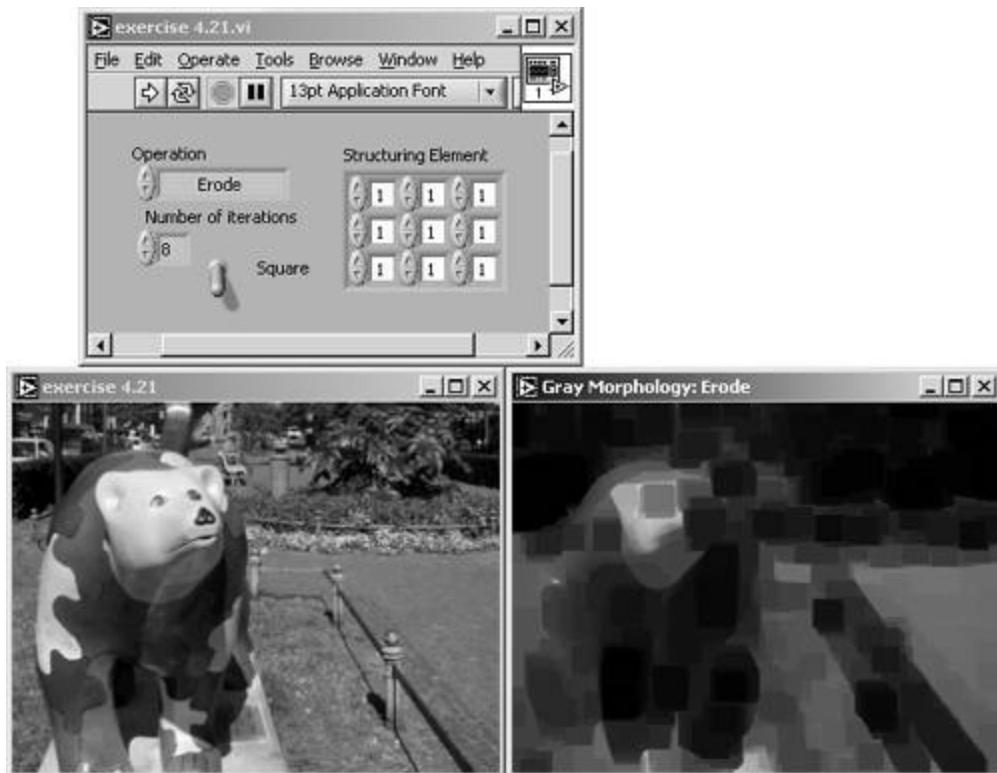
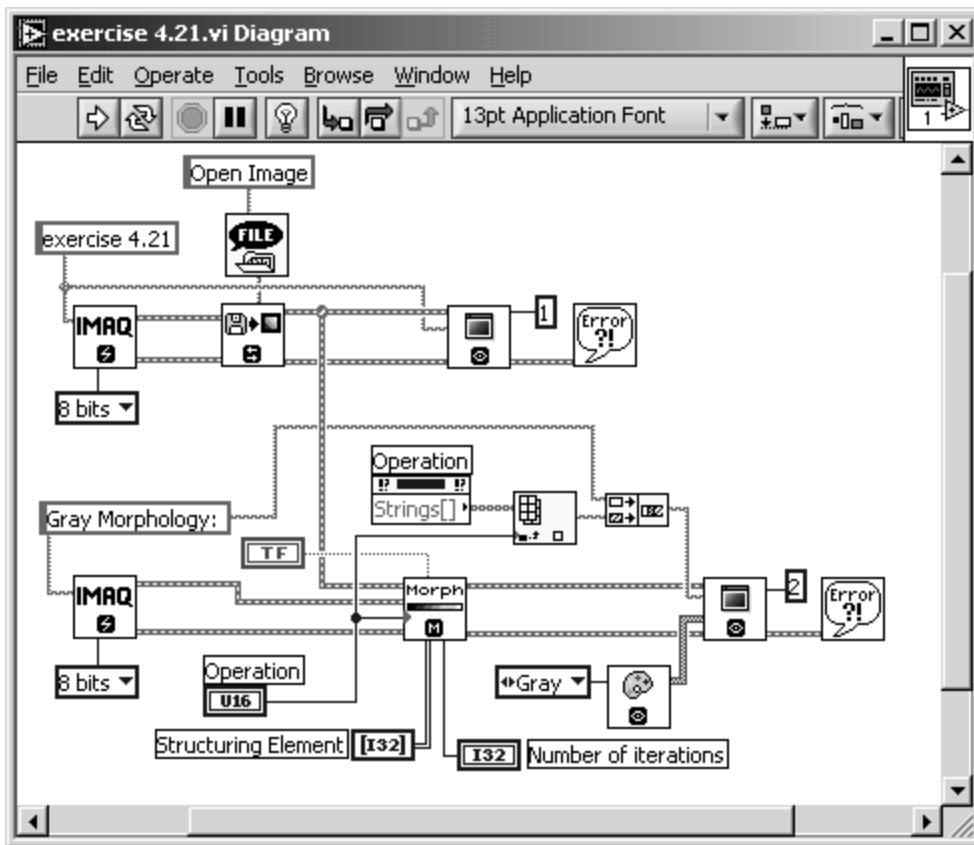


Figure 4.84. Diagram of [Exercise 4.21](#)



Of course, we have to modify the algorithm of the erosion function. In binary morphology, we set the center pixel value s_0 to 0 if at least one of the other elements (s_i) is equal to 0. Here we have gray-level values; therefore, we set s_0 to the *minimum* value of all other coefficients s_i :

Equation 4.44

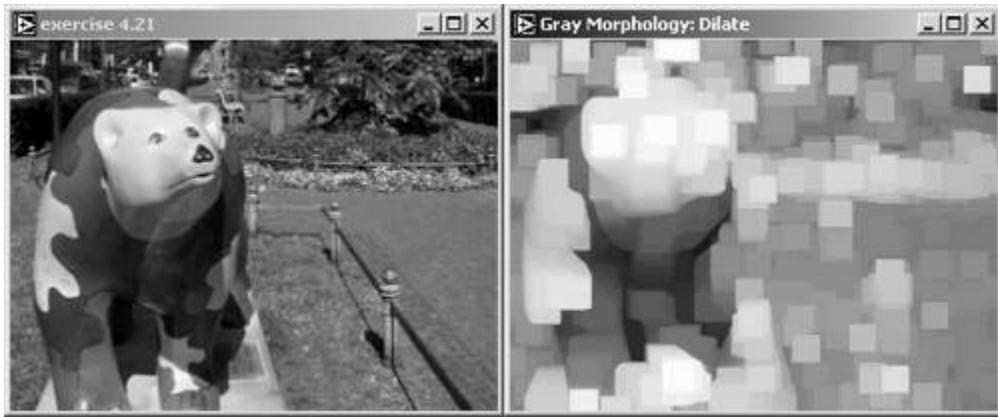
$$s_0 = \min(s_i) .$$

The gray-level dilation is shown in [Figure 4.85](#) and is now defined easily: s_0 is set to the *maximum* value of all other coefficients s_i :

Equation 4.45

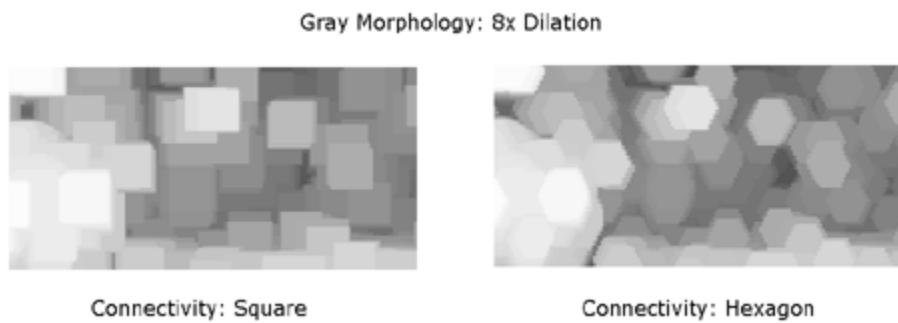
$$s_0 = \max(s_i) .$$

Figure 4.85. Gray Morphology: Dilation



This exercise is a good opportunity to visualize the impact of different connectivity structures: square and hexagon. [Figure 4.86](#) shows the same dilation result with square connectivity (left) and hexagon connectivity (right). The shape of the "objects" (it is not easy to talk of objects here because we do not have a binary image) after eight iterations is clearly determined by the connectivity structure.

Figure 4.86. Comparison of Square and Hexagon Connectivity



Gray-Level Opening and Closing

Also in the world of gray-level images, the opening function is defined as a (gray-level) erosion followed by a (gray-level) dilation using the same structuring element:

Equation 4.46

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

[Figure 4.87](#) shows the result of a gray-level opening function.

Figure 4.87. Gray Morphology: Opening



The result of a gray-level closing function can be seen in [Figure 4.88](#). The gray-level closing function is defined as a gray-level dilation followed by a gray-level erosion using the same structuring element:

Figure 4.88. Gray Morphology: Closing



Equation 4.47

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

Gray-Level Proper Opening and Proper Closing

The two gray-level functions *proper opening* and *proper closing* are a little bit different from their binary versions. According to [4], gray-level proper opening is used to remove bright pixels in a dark surrounding and to smooth surfaces; the function is described by

Equation 4.48

$$\text{proper opening}(\mathcal{I}) = \min(\mathcal{I}, \text{opening}(\text{closing}(\text{opening}(\mathcal{I})))) \quad ;$$

note that the AND function of the binary version is replaced by a minimum function. A result is shown in [Figure 4.89](#).

Figure 4.89. Gray Morphology: Proper Opening



The gray-level proper closing function is defined as

Equation 4.49

$$\text{proper closing}(\mathcal{I}) = \max(\mathcal{I}, \text{closing}(\text{opening}(\text{closing}(\mathcal{I})))) ;$$

a result is shown in [Figure 4.90](#).

Figure 4.90. Gray Morphology: Proper Closing



Auto-Median Function

Compared to the binary version, the gray-level *auto-median* function also produces images with fewer details. The definition is different; the gray-level auto-median calculates the minimum value of two temporary images, both resulting from combinations of gray-level opening and closing functions:

Equation 4.50

$$\begin{aligned} \text{automedian}(\mathcal{I}) = & \min(\text{opening}(\text{closing}(\text{opening}(\mathcal{I}))), \\ & \text{closing}(\text{opening}(\text{closing}(\mathcal{I})))) . \end{aligned}$$

See [Figure 4.91](#) for a result of the gray-level auto-median function.

Figure 4.91. Gray Morphology: Auto-Median Function



[\[Team LiB \]](#)

[PREVIOUS](#) [NEXT](#)

Chapter 5. Image Analysis

This chapter is divided into two parts. In the first part, we discuss the most relevant image analysis and machine vision techniques, starting with the analysis of *pixel values* and *particle measurements*. Another main topic is an introduction to *pattern matching*.

The exercises in this chapter, like the ones before, mainly describe how to use specific functions directly in LabVIEW and IMAQ Vision. Although it would be possible to use the IMAQ Vision Builder (I do use it in some cases), I focus on IMAQ Vision programs because of their greater flexibility.

The second part of this chapter deals with some application examples, which use some of the described techniques.

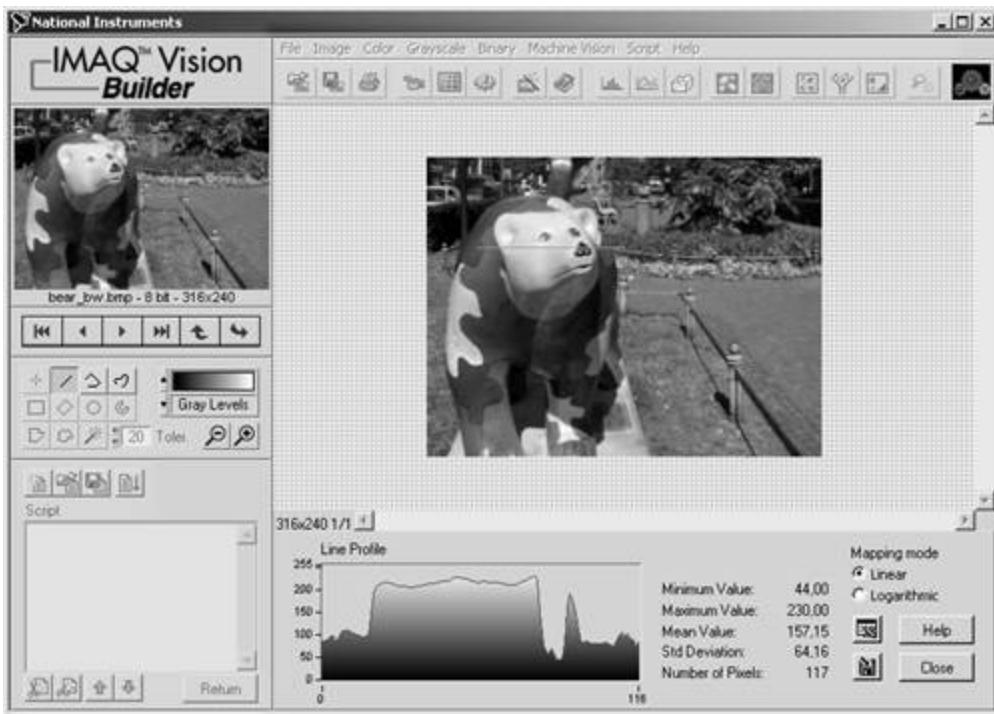
Pixel Value Analysis

Although this chapter would be the correct place for the histogram function, we already discussed it in [Chapter 4](#) on page 151, so go there for detailed information. Another function, which gives a pixel value distribution, is discussed here: the *line profile*.

Line Profile

The line profile function simply visualizes the pixel values (gray-level values for gray-scaled images) along a line, which must be specified by its coordinates or results from a manually drawn line. [Figure 5.1](#) shows this function in the IMAQ Vision Builder.

Figure 5.1. Line Profile Function in IMAQ Vision Builder



Directly in LabVIEW and IMAQ Vision, you must specify the line coordinates by using an array control or an array constant. [Exercise 5.1](#) shows how.

Exercise 5.1: Line Profile in Images.

Create a LabVIEW VI that draws the pixel value profile along a previously defined line in an image, using the function [IMAQ Line Profile](#) (found in Motion and Vision / Image Processing / Analysis). [Figures 5.2](#) and [5.3](#) show a possible solution.

Actually, it is more convenient to draw the line directly in the image, using a function like [IMAQ SelectLine](#), much like the behavior of IMAQ Vision Builder. I leave the extension of [Exercise 5.1](#) up to you.

[Figures 5.2](#) and [5.3](#) also show another interesting group of VIs: the *overlay* functions. You can find them in Motion and Vision / Vision Utilities / Overlay (see [Figure 5.4](#)); they are useful if you want to provide additional information in the IMAQ image window itself.

Figure 5.2. Line Profile of an Image



Figure 5.3. Diagram of [Exercise 5.1](#)

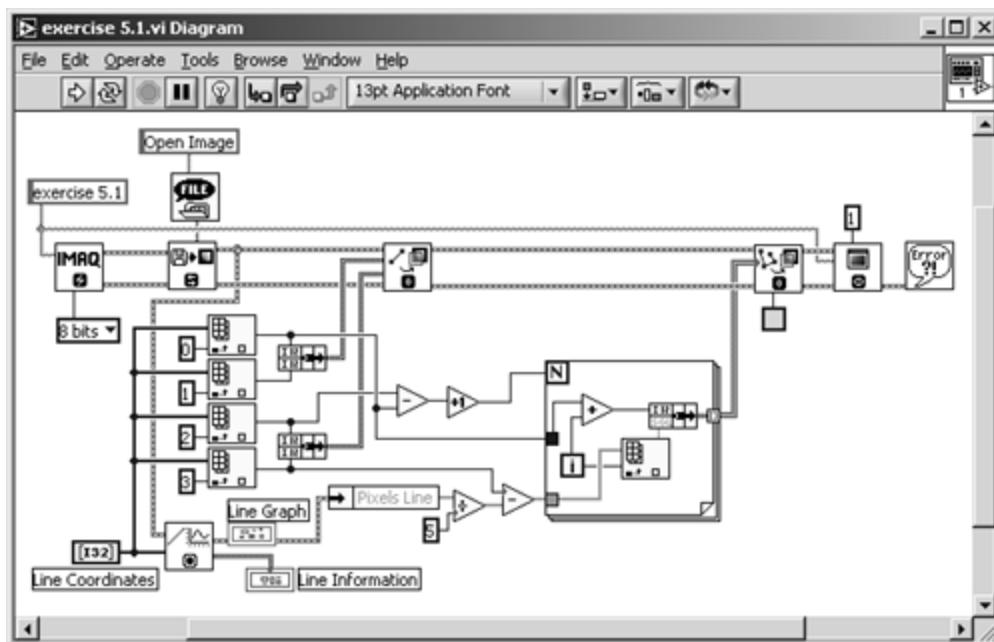
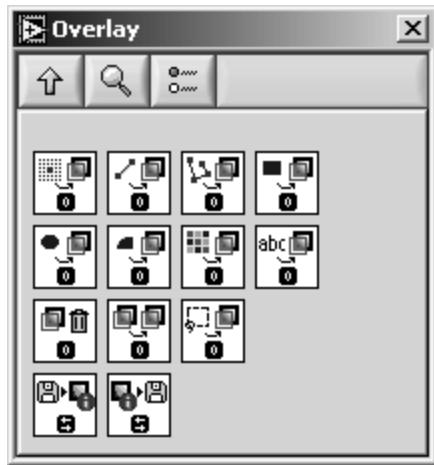


Figure 5.4. Menu Palette Containing Overlay Functions



Almost all overlay functions have the following inputs:

- the original image;
- the item type, which should be placed over the original image, for example:
 - single points,
 - lines (single or multiple),
 - rectangles,
 - circles and ovals,
 - text,
 - bitmaps;
- at least one coordinate pair, specifying where to place the overlay;
- the color of the overlay (not with bitmaps).

The only output is the resulting image with the overlay information, so the overlay is typically placed between processing or analysis functions and the `IMAQ WindDraw` function.

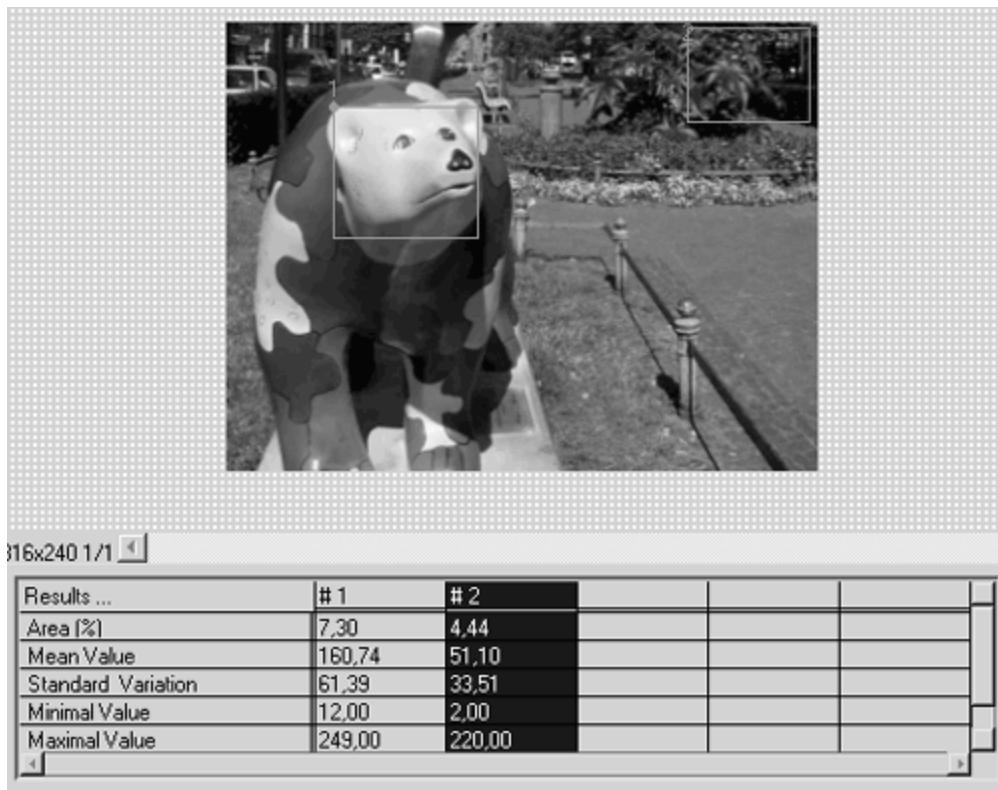
I used two different overlay functions in [Exercise 5.1](#). As you can see in [Figure 5.3](#), the first one simply draws the line you can specify in the Line Coordinates control. Note that the line coordinates are stored in a 4×1 array; but the data required by `IMAQ OverlayLine` is two clusters of the coordinates of the start and the end point, respectively.

The second function is more complex; the exercise uses the gray-scale values of the line profile, recalculates them so that they can be displayed in the IMAQ window, and displays them on top of the red line. Red is the default color of the overlay structure; the profile color can be changed by use of a color constant, for example, to yellow.

Quantify Areas

The `quantify` function can be used to obtain simple statistical data about the gray-level values of pixels in a specified region. We try this exercise in IMAQ Vision Builder, when you select the menu item Quantify in the Grayscale menu. After selecting a region of interest, you can view the results ([Figure 5.5](#)).

Figure 5.5. Quantifying Image Areas with IMAQ Vision Builder



The values provided by IMAQ Vision Builder using the quantify function are:

- mean value;
- standard variation;
- minimum value, and
- maximum value.

You will get an interesting result if you let IMAQ Vision Builder construct a LabVIEW VI ([Figure 5.6](#)). Obviously, the Vision Builder does not use the function **IMAQ Quantify**, as we expected; it uses a function **IVB Quantify** instead. IMAQ Vision Builder uses the IVB functions for the automated construction of VIs because they provide a more standardized system for input and output variables.

Figure 5.6. LabVIEW Quantify VI Generated with IMAQ Vision Builder

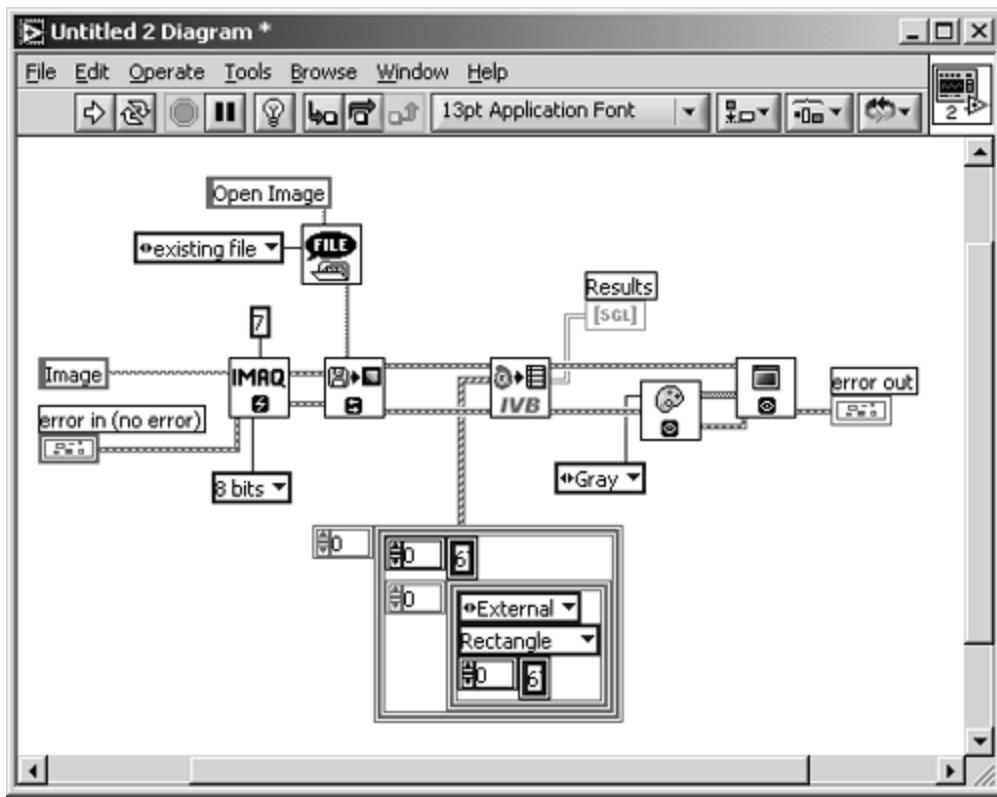
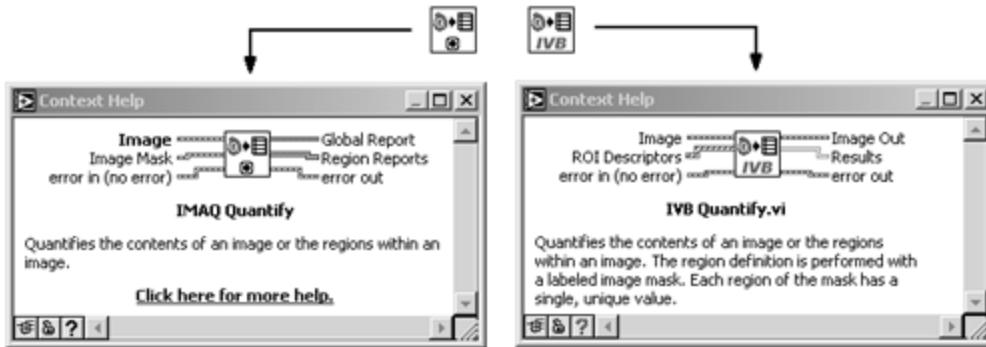


Figure 5.7. IVB (IMAQ Vision Builder) Functions

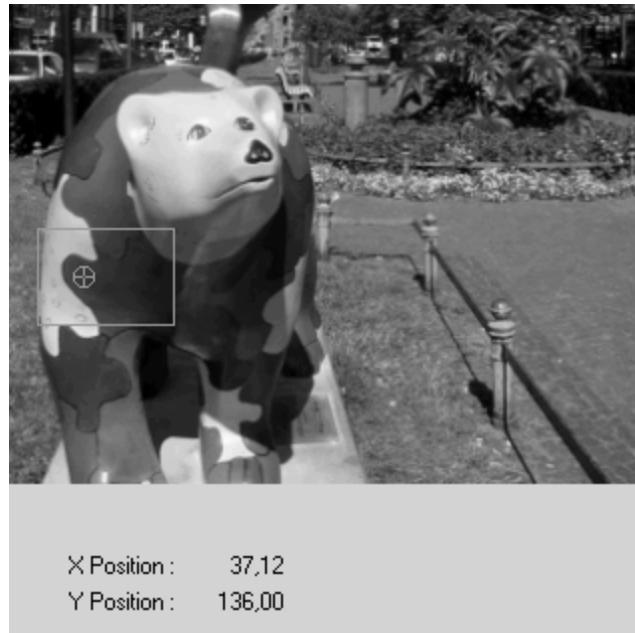


[Figure 5.7](#) shows the difference between the two functions used in this exercise. You can find more IVB functions following the directory path (depending on your system) ...[National Instruments\IMAQ Vision Builder 6\program\ext\addons](#).

Centroid Function

The function [IMAQ Centroid](#) calculates the Center of Energy of either a specified region or the entire image. You can watch the results by selecting the Centroid function in IMAQ Vision Builder and by specifying various areas with different brightness distributions ([Figure 5.8](#)).

Figure 5.8. Centroid Function (Center of Energy)



Linear Averages

We use overlay functions in the next exercise, with the IMAQ Vision function [IMAQ LinearAverage](#). The function itself is simple; it calculates the mean line profiles in x and in y direction, respectively.

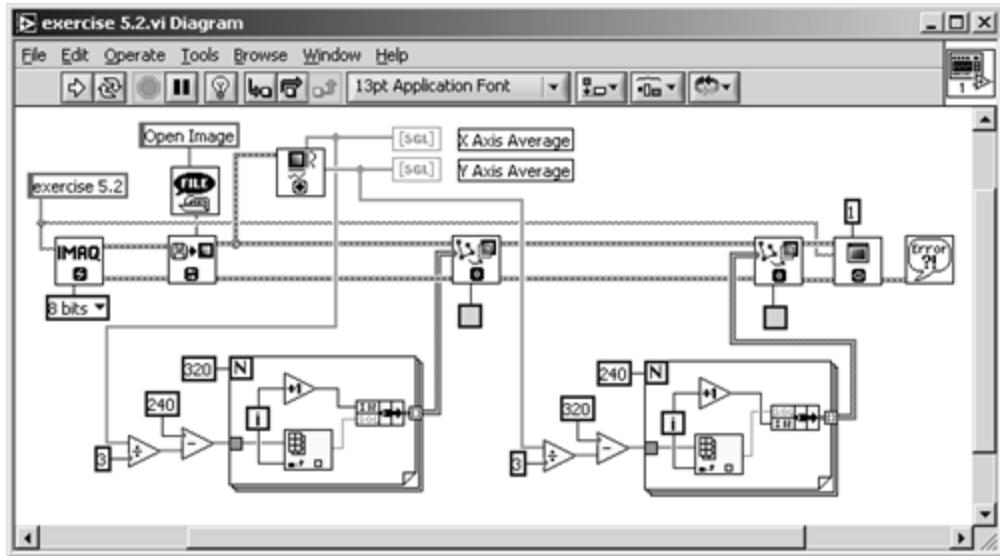
Exercie 5.2: Linear Averages.

Calculate the mean linear averages in x and in y direction, using the function [IMAQ LinearAverage](#). Display the line profiles in both directions directly in the image, similarly to the solution in [Exercise 5.1](#). Compare your solution to [Figures 5.9](#) and [5.10](#).

Figure 5.9. Linear Average of Pixel Values in x and y Direction



Figure 5.10. Diagram of [Exercise 5.2](#)



The function `IMAQ_LinearAverage` also enables you to specify only a region of the image, in which you want the average line profiles to be calculated. To do this, you add an array constant or an array control and enter the coordinates of an Optional Rectangle.

Edge Detection

In *Edge Detection and Enhancement* in [Chapter 4](#), we discussed only edge enhancing, because we did not get any information (in numbers^[1]) about the location of the edge. We can get numerical information in the following exercises.

[1] Remember the definitions of image processing and image analysis and the resulting difference between them.

Simple Edge Detector

We can use the information returned by the line profile function to determine edges along the line by simply checking whether the gray-level value of the pixels exceeds a previously given threshold value, as in [Exercise 5.3](#).

Exercise 5.3: Simple Edge Detector.

Create a LabVIEW VI that uses the line profile information for the realization of a simple edge detector, using the functions `IMAQ Line Profile` and `IMAQ SimpleEdge` (can be found in Motion and Vision / Machine Vision / Caliper). See a possible solution in [Figures 5.11](#) and [5.12](#).

Figure 5.11. Simple Edge Detector

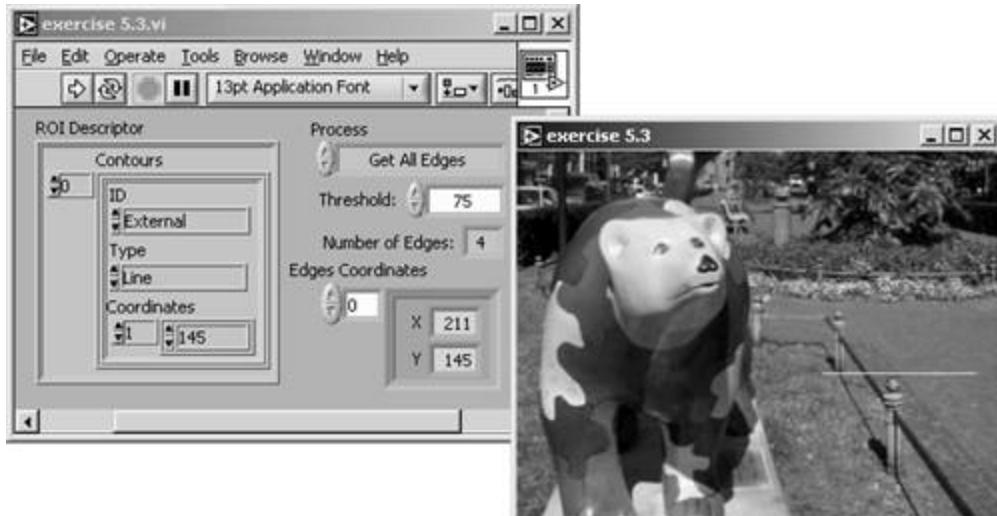
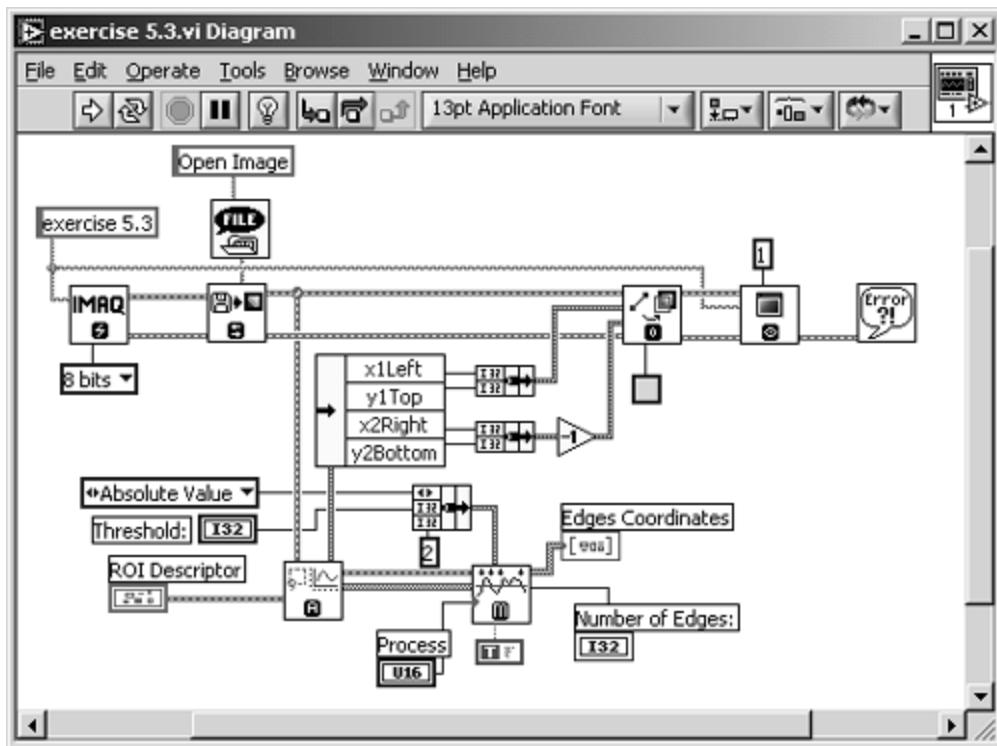


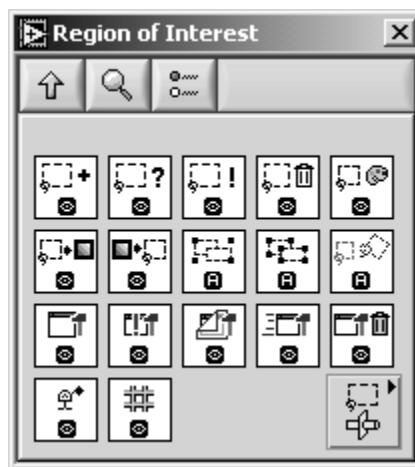
Figure 5.12. Diagram of [Exercise 5.3](#)



Again, we use an overlay function to visualize the line on which the edges should be detected. You can extend [Exercise 5.3](#) by indicating the edges, for example, by drawing small circles at the edge location, using [IMAQ OverlayOval](#). The application example *Feedback Form Reader* on page 304 offers a possible solution.

[Exercise 5.3](#) also uses the function [IMAQ ROIProfile](#), which converts a region of interest (ROI) to pixel coordinates. An ROI limits the desired operation to a smaller area of the image. IMAQ Vision provides a number of ROI functions in Motion and Vision / Vision Utilities / Region of Interest (see [Figure 5.13](#)).

Figure 5.13. Menu Palette Containing ROI Functions



An ROI is described by the ROI descriptor, which is a cluster containing the following:

- a *bounding rectangle*, and
- a *regions list*, an array of clusters containing
- a *contour identifier* (0 for exterior and 1 for interior contour),
-

- the *contour type*, for example, rectangle or oval,
- a *list of points* describing the contour.

You can read more about ROIs in [5].

IMAQ Edge Detection Tool

If the function `IMAQ_SimpleEdge` does not meet your needs, you can use the function `IMAQ_EdgeTool` (to be found in Motion and Vision / Machine Vision / Caliper). Here, you can specify additional parameters like the slope steepness or subpixel accuracy.

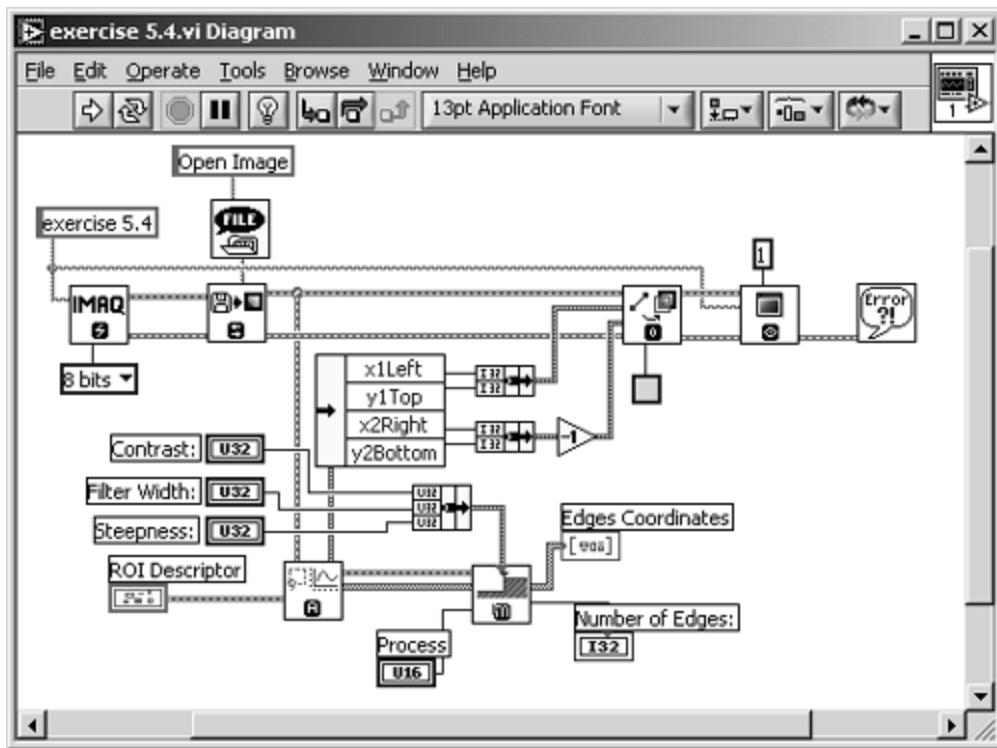
Exercise 5.4: Complex Edge Tool.

It is easy to change [Exercise 5.3](#) by replacing `IMAQ_SimpleEdge` with `IMAQ_EdgeTool` and to modify the necessary controls. You will need numeric controls for Contrast, Filter Width, and slope Steepness. I did not wire the input for subpixel information, which may enable higher resolution than the pixel size and uses special interpolation functions. Try it yourself, starting with [Figures 5.14](#) and [5.15](#).

Figure 5.14. Edge Detection Tool



Figure 5.15. Diagram of [Exercise 5.4](#)



Peak-Valley Detection

The next function is not really an edge detector; it detects peaks or valleys of a line profile. In addition, it returns information about the peaks or valleys, including the second derivatives.

Exercise 5.5: Peak-Valley Detector.

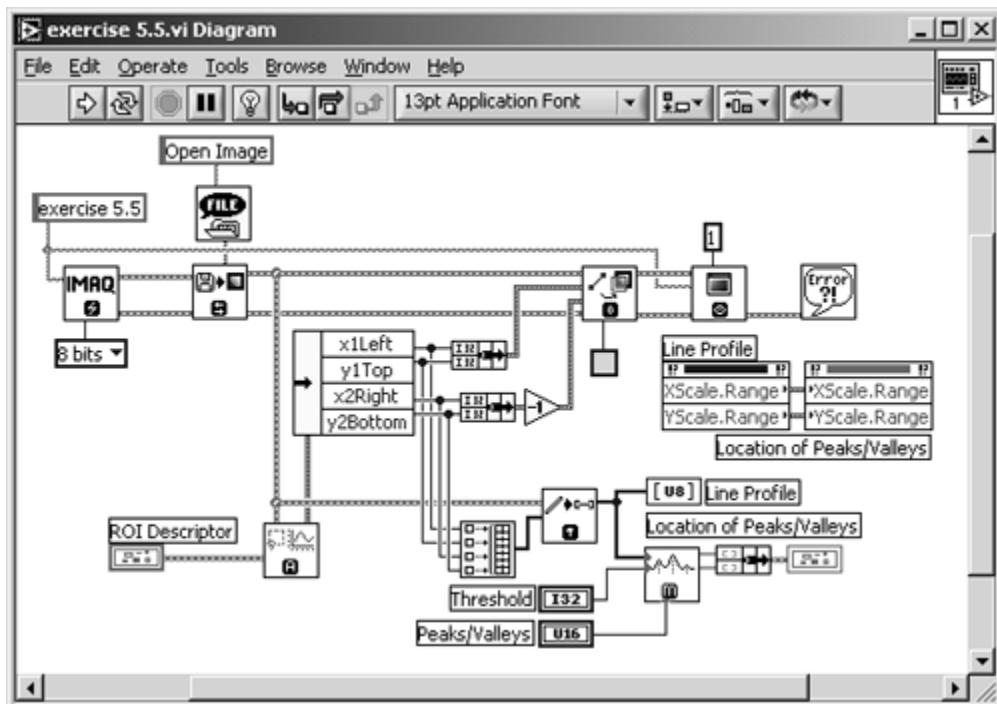
In this exercise, use the IMAQ Vision function [IMAQ Peak-Valley Detector](#). Display the line in the original image as in the exercises before and the gray-level profile in a waveform graph. Overlay this graph with the detected peaks or valleys.

I used a trick for the overlay information: the XY graph in the diagram ([Figure 5.17](#)) is a transparent one and placed over the waveform graph. Because the waveform graph has autoscaled x and y axes, the ranges of both axes are transferred to the XY graph by Property Nodes. You have to turn the autoscale mode off in the XY graph. See also [Figure 5.16](#).

Figure 5.16. Detecting Peaks and Valleys of a Line Profile



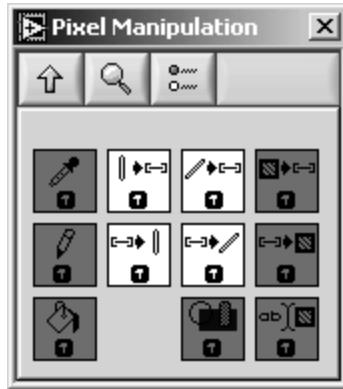
Figure 5.17. Diagram of [Exercise 5.5](#)



Another group of VIs, which can be found in Motion and Vision / Vision Utilities / Pixel Manipulation, is used in [Exercise 5.5](#). We have to use the function **IMAQ_GetPixelLine** here because the peak-valley detector needs an array containing the line profile information as input. The other functions, shown in [Figure 5.18](#), can be used to either set or get the pixel

values of image rows, columns, or lines, specified by their coordinates.

Figure 5.18. Menu Palette Containing Pixel Manipulation Functions



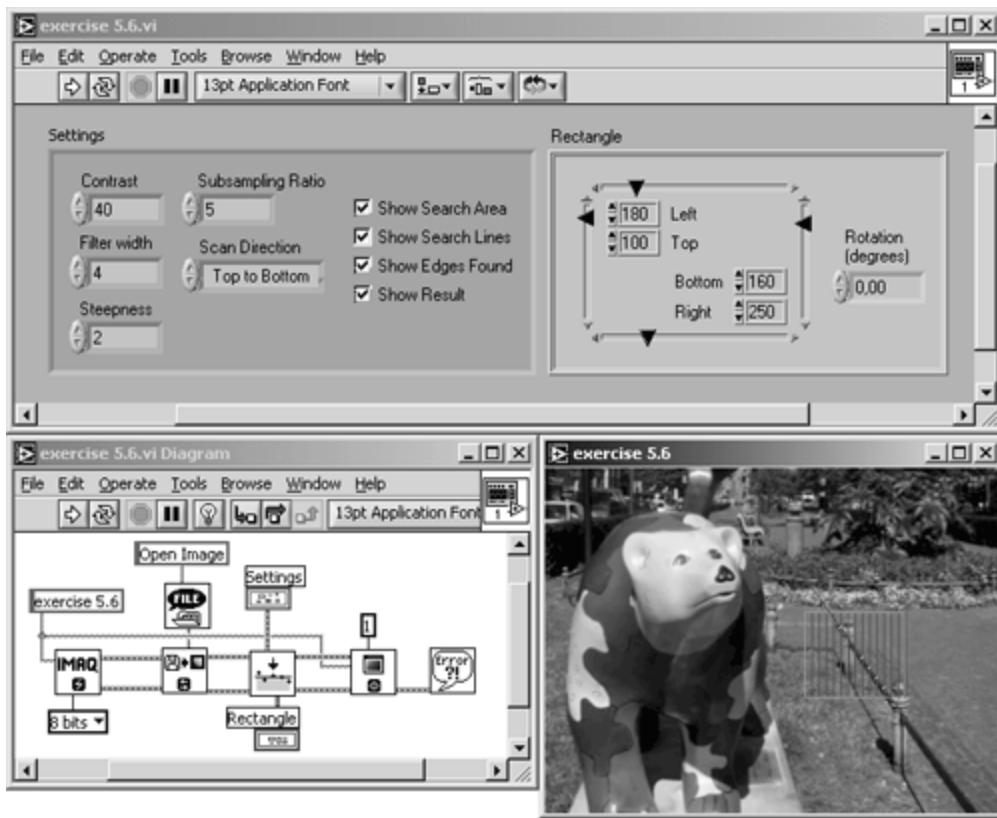
Locating Edges

In our next exercise we try to build an almost complete machine vision application, using the function `IMAQ Find Horizontal Edge` (found in Motion and Vision / Machine Vision / Locate Edges). The VI should locate horizontal edges of objects in a specified ROI of the image.

Exercise 5.6: Finding Horizontal Edges.

You can see that the diagram of this exercise ([Figure 5.19](#)) is small and simple, although the application itself is very powerful. The reason is that the function `IMAQ Find Horizontal Edge` already contains a full machine vision application. All you need to do is wire a cluster for the settings, including filter settings (similar to [Exercise 5.4](#)), the scan direction (left to right, right to left, top to bottom, or bottom to top), and check boxes for the overlay of information to the original image. Another cluster control specifies the search region within the image.

Figure 5.19. Locating Edges in Images



This exercise VI also specifies the edge coordinates inside the ROI. Because we are looking for straight edges here, the coordinates are used to calculate a line that fits the edge of the object. In [Figure 5.19](#) I used our bear image, but it is obvious that this image is not appropriate for this function.

[Figure 5.20](#) shows the application of [Exercise 5.6](#) to an image showing the top view of a small motor, which is common in household appliances and toys. The image is prepared for image vision by maximizing the contrast ratio; it is almost binary. If you want to use your own images, you can modify contrast, brightness, and saturation with a program like Corel Photo Paint; of course, you can also use IMAQ Vision to do this.

Figure 5.20. Locating Horizontal Edges



Let's now extend [Exercise 5.6](#) so that circular edges can be detected.

Exercise 5.7: Finding Circular Edges.

Modify [Exercise 5.6](#) by adding the function `IMAQ Find Circular Edge`. Use both edge location functions to detect horizontal and circular edges in the motor image by adjusting the parameters of the Rectangle and the Circle or Annulus cluster control. The Settings controls are used for both functions; only the Scan Direction control is different. See [Figure 5.21](#) for the front panel, [Figure 5.22](#) for the diagram, and [Figure 5.23](#) for a result.

Figure 5.21. Locating Horizontal and Circular Edges

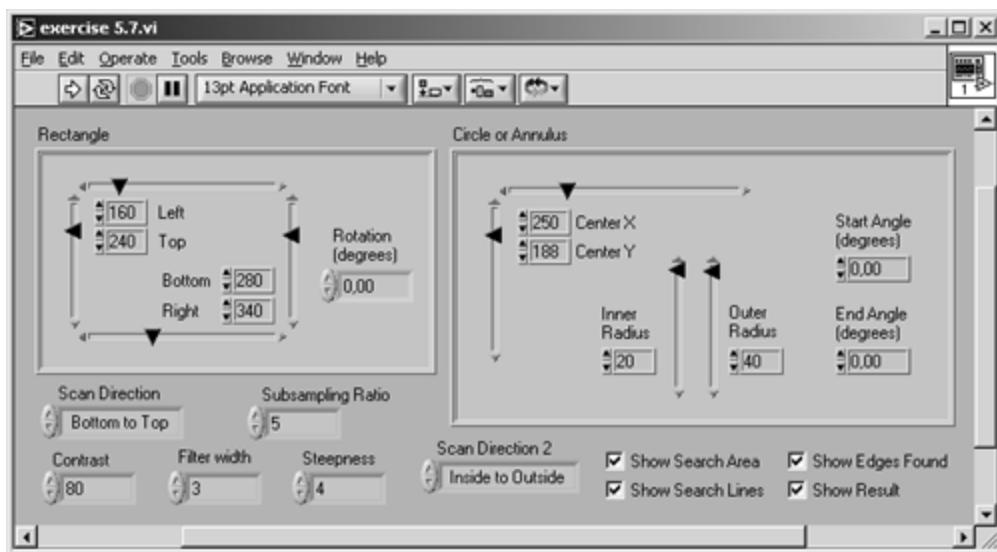


Figure 5.22. Diagram of [Exercise 5.7](#)

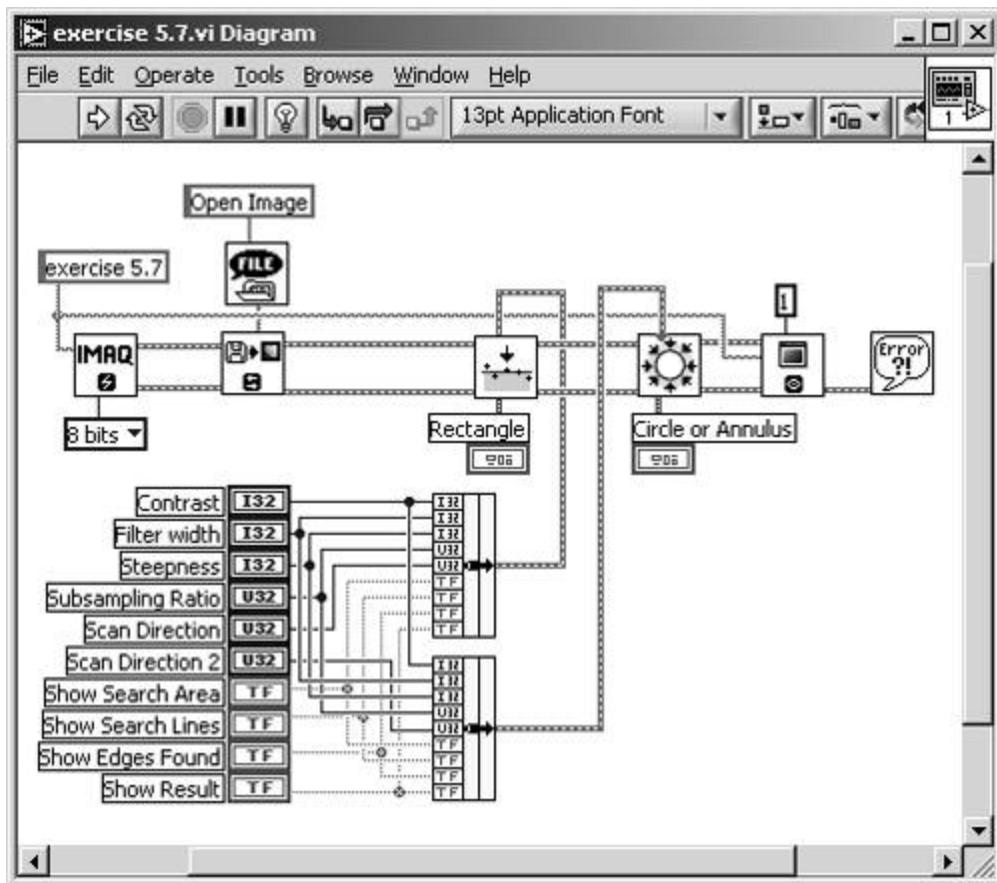
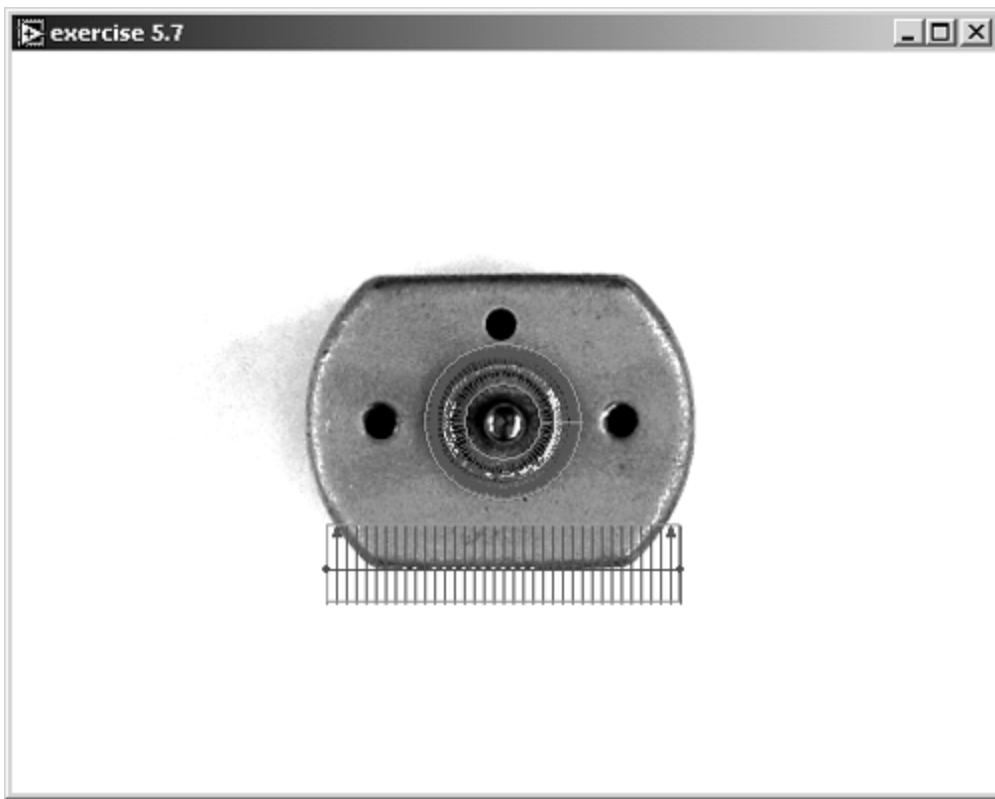


Figure 5.23. Edge-Locating Result (Motor)



Other functions in the submenu Motion and Vision / Machine Vision / Locate Edges provide the location of vertical edges ([IMAQ Find Vertical Edge](#)) and circular edges ([IMAQ Find Concentric Edge](#)). You should have no problem now in building your own edge-locating applications.

Morphology Analysis

We now return to the binary morphology VIs of [Chapter 4](#). Some of them, like skeleton or skiz functions, were already close to analysis functions according to our definition.

Distance and Danielsson

The [IMAQ Distance](#) and the [IMAQ Danielsson](#) functions, for example, provide information about the distance of an object (white areas) from the background border (black areas) and display this information by using colors of the binary palette.

Exercise 5.8: Distance and Danielsson.

Take an exercise from [Chapter 4](#), for example, [Exercise 4.20](#), and insert the function [IMAQ Distance](#). You may provide the function [IMAQ Danielsson](#) as well and put it in a Case Structure, which can be used to select the function type. See [Figure 5.24](#) for the front panel and [Figure 5.25](#) for the diagram.

Figure 5.24. Distance Indication in Binary Images

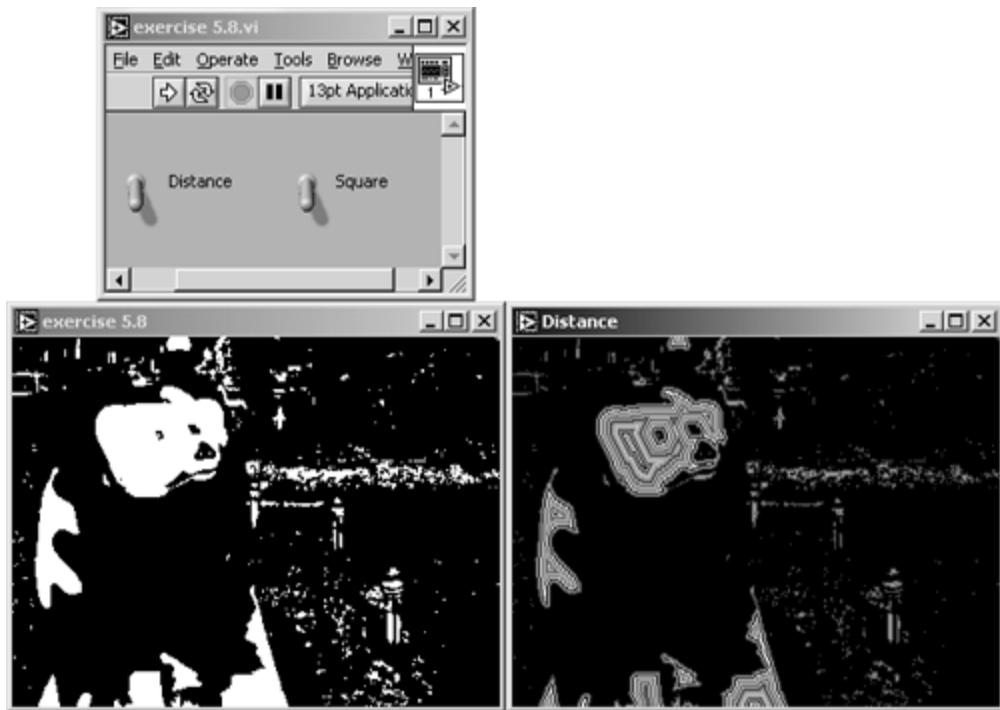
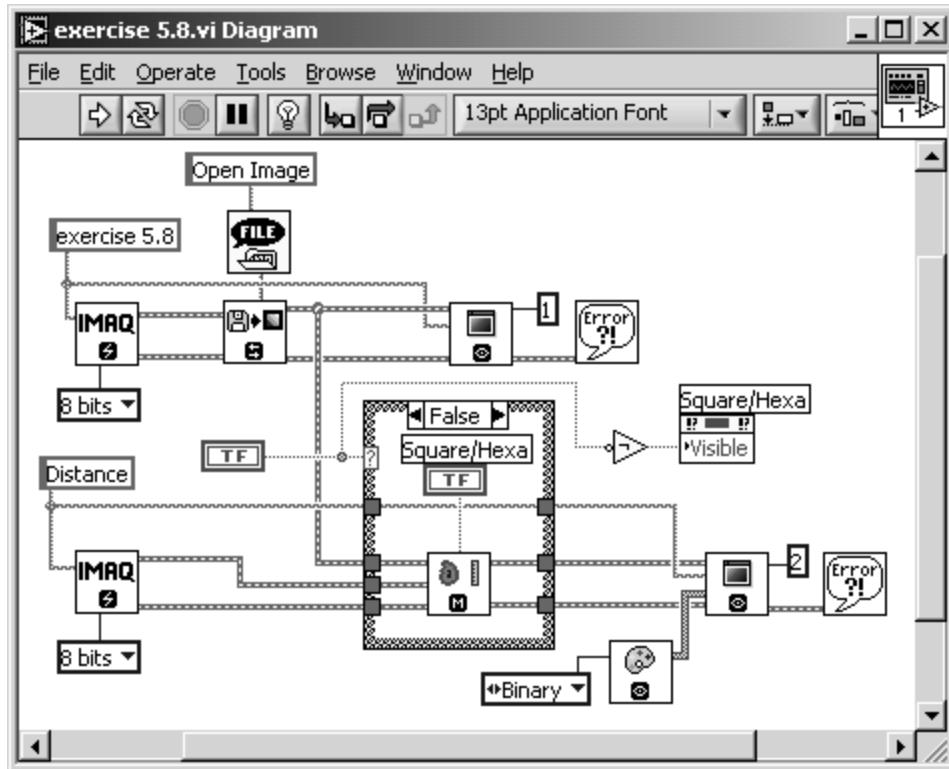


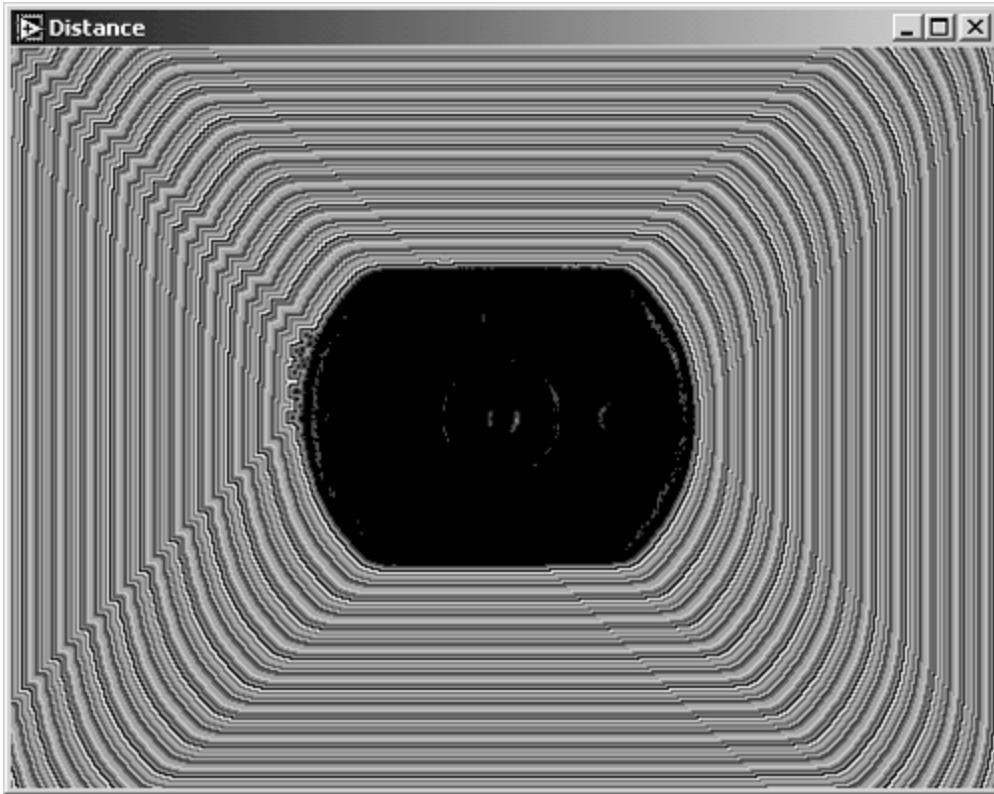
Figure 5.25. Diagram of [Exercise 5.8](#)



[Figure 5.24](#) shows the result of the distance function in the binary bear image. Again, it is clear that this image is not appropriate for this function, although the principle is clear: Each object shows a number of color lines, which indicate the distance to the background.

[Figure 5.26](#) shows the same function applied to the motor image of the last exercise. The effect of the distance function gets even clearer, but look at the areas near the straight edges of the image. Inside these four regions, which look like a big "V," are pixels with the same distance mapping. But it can be clearly seen that their distance to the next black pixel is *different*; this algorithm is obviously not very accurate.

Figure 5.26. Distance Function Applied to a Binary Motor Image

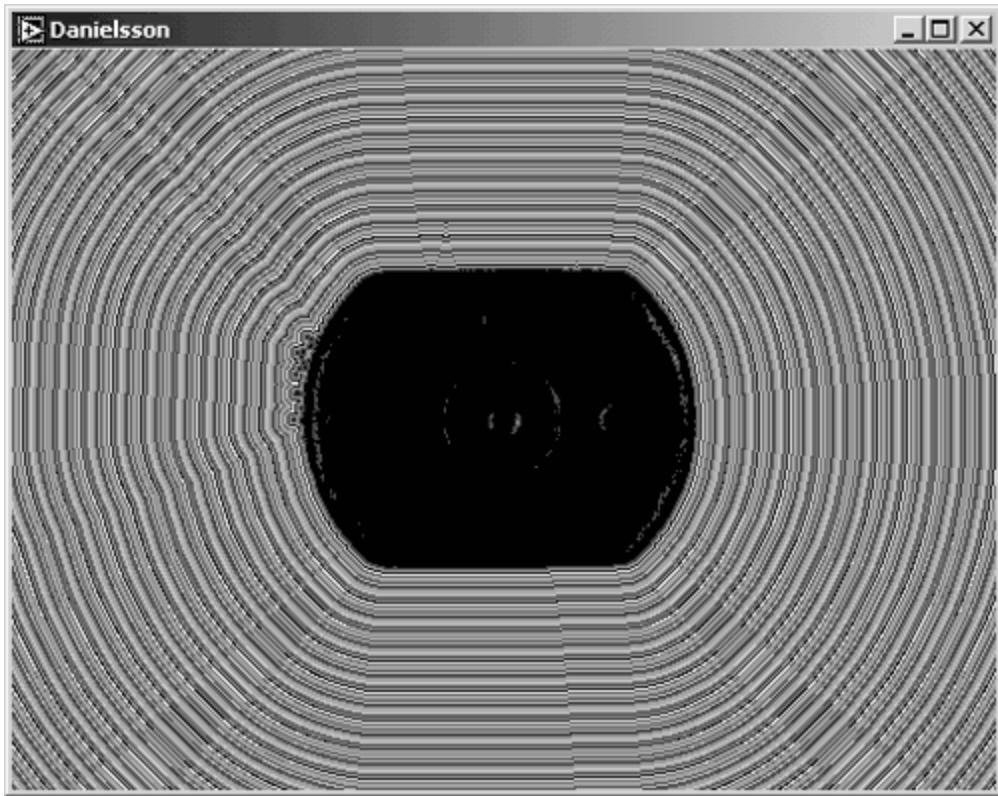


The *Danielsson* function is based on an algorithm called *Euclidean Distance Mapping* and was introduced by *P. E. Danielsson* in 1980.^[2] As [Figure 5.27](#) shows, the algorithm shows more reliable results.

^[2] P. E. Danielsson: "Euclidean Distance Mapping." *Computer Graphics and Image Processing* , Volume 14, 1980, pp. 227-248.

Danielsson himself calls the algorithms he developed "Four-point Sequential Euclidean Distance Mapping" (4SED) and "Eight-point Sequential Euclidean Distance Mapping" (8SED), respectively, which reminds us of [Figure 4.48](#), showing the definition of connectivity =4 and connectivity =8.

Figure 5.27. Danielsson Function Applied to a Binary Motor Image



Labeling and Segmentation

We also discussed functions like [IMAQ Convex](#), which require "labeling" of objects in a binary image, which means that these objects are classified and numbered. The next exercise creates a labeled binary image.

Exercise 5.9: Labeling Particles.

Modify [Exercise 5.8](#) by inserting the function [IMAQ Label](#) (to be found in Motion and Vision / Image Processing / Processing). [Figure 5.28](#) shows that the labeling is visualized by assignment of a color of the binary palette (visible by different gray levels in [Figure 5.28](#)). Use a numeric indicator to display the number of objects. See [Figure 5.29](#) for the diagram.

Figure 5.28. Labeling of Binary Images

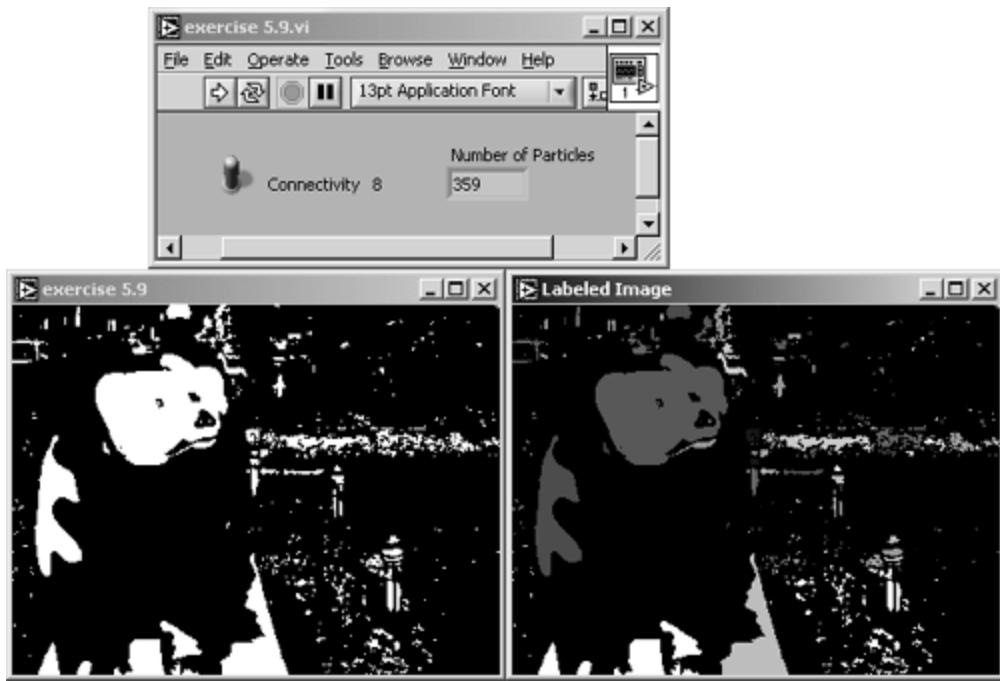
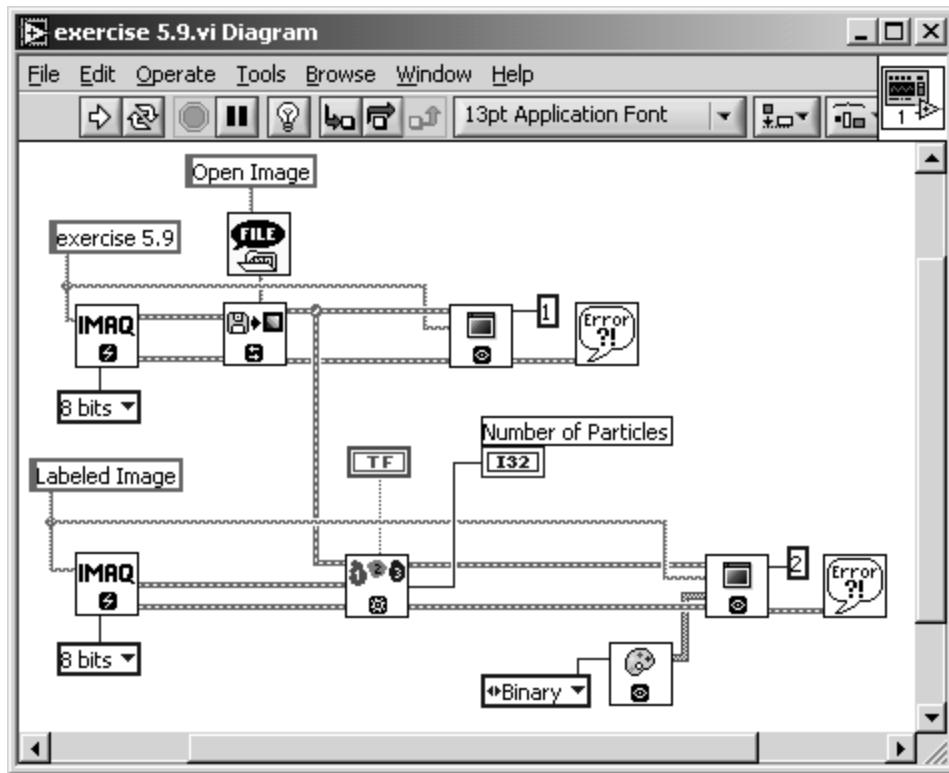


Figure 5.29. Diagram of [Exercise 5.9](#)



The function **IMAQ Segmentation** requires a labeled binary image and expands the objects until each object reaches its neighbors. The expanding is done by multiple dilation functions until the image contains no more background areas.

Exercise 5.10: Segmentation of Images.

Insert the function **IMAQ Segmentation** in [Exercise 5.9](#), so that it follows **IMAQ Label**. It is useful to display the labeled image as well as the segmented image, so

you have to do some changes in the wiring. See [Figure 5.30](#) for the panel and the results and [Figure 5.31](#) for the diagram.

Figure 5.30. Segmentation of Labeled Binary Images

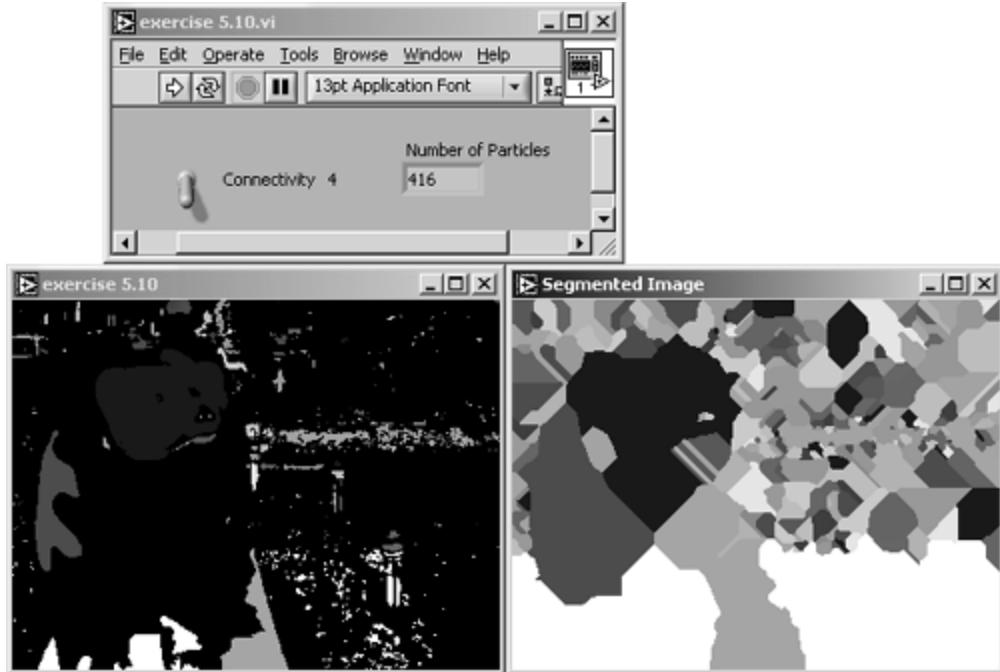
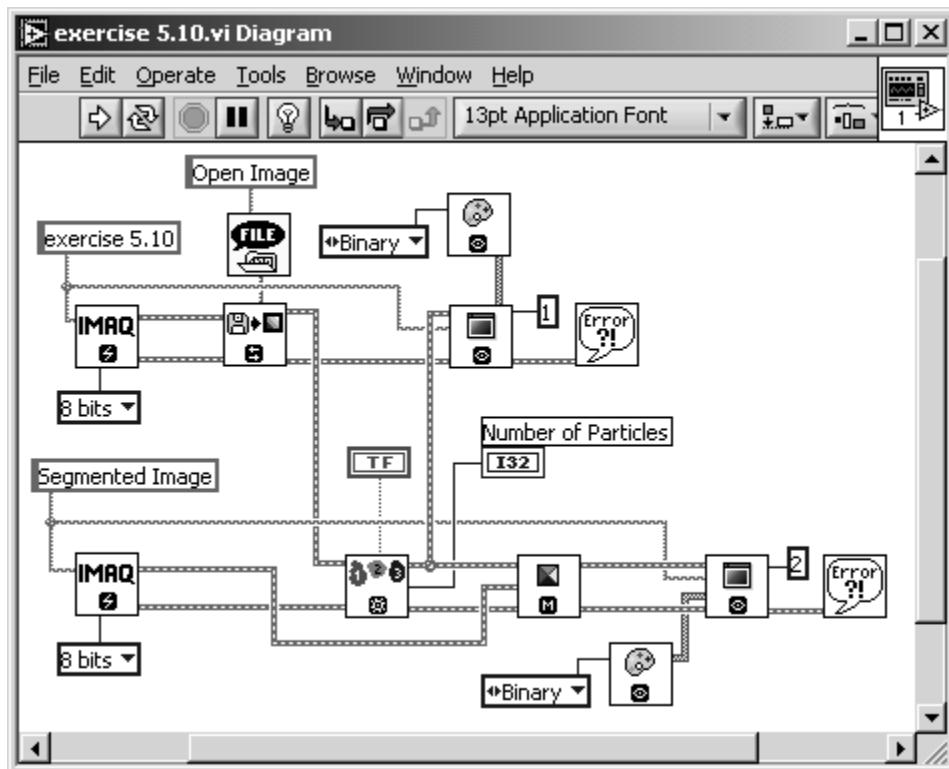
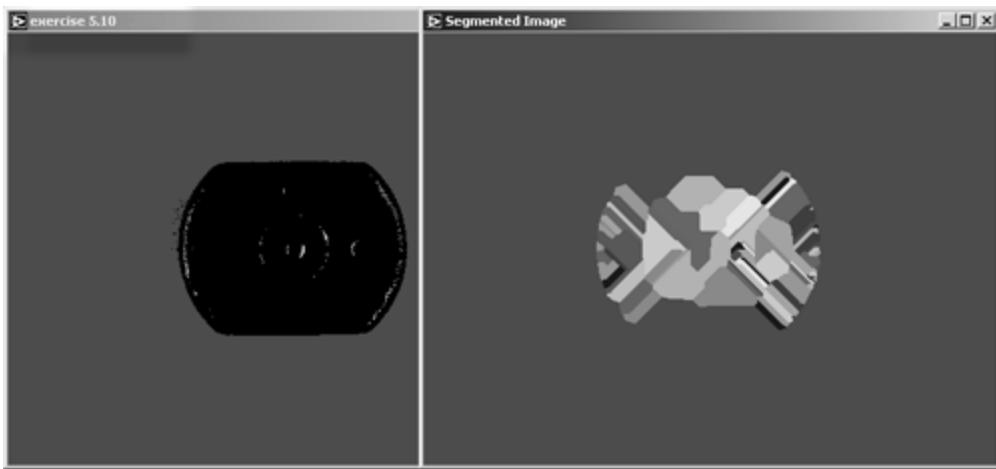


Figure 5.31. Diagram of [Exercise 5.10](#)



The segmentation of a binary image can be interpreted as the calculation of an influence region of each labeled object. [Figure 5.32](#) shows the labeling and segmentation result for the motor image.

Figure 5.32. Segmentation Result of the Motor Image



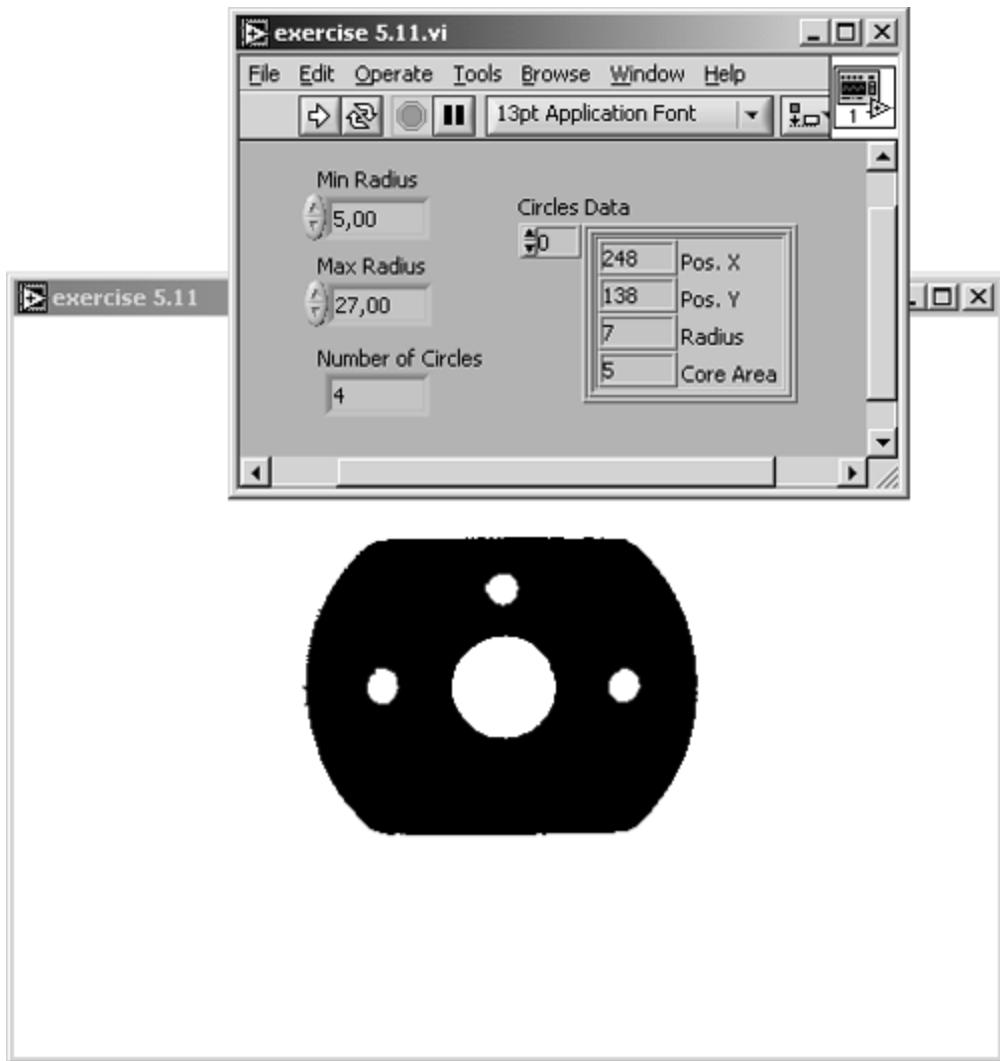
Circle Detection

The next function, `IMAQ Find Circles`, detects circles in binary images and is therefore very similar to the function `IMAQ Find Circular Edge`. The only difference is that `IMAQ Find Circles` searches not only for edges, but for the entire circle area. We test this function in the next exercise.

Exercise 5.11: Finding Circles.

Modify [Exercise 5.9](#) (*not* [5.10](#)) by replacing `IMAQ Label` with `IMAQ Find Circles`. Provide numeric controls for the specification of the minimum and the maximum circle radius value, and indicators for the number of detected circles and the circles data. See [Figure 5.33](#) for the panel, [Figure 5.34](#) for results, and [Figure 5.35](#) for the diagram.

Figure 5.33. Circle Detection Exercise



Circles Data is an array of clusters, one cluster for each detected circle, consisting of the following elements:

- the x coordinate,
- the y coordinate,
- the radius of the detected circle, and
- the entire circle area.

We need a new binary image for the testing of this function; you can either use `dc_motor_bin3.png` from the attached CD or modify `dc_motor.png`, for example, by using IMAQ Vision Builder, by performing the following steps:

1. Load `dc_motor.png` in the IMAQ Vision Builder.
2. Apply a threshold operation with a minimum value of 30 and a maximum of 190.
3. Invert the image.
4. Remove small objects from the Adv. Morphology menu (two iterations).
5. Invert the image.
- 6.

6. Remove small objects (seven iterations, hexagonal connectivity).
7. Invert the image.

[Figure 5.34](#) shows the result if the generated binary motor image is processed; four circles should be detected. If you need more information about the generation of the image we use, here is the entire IMAQ Vision Builder script.

Figure 5.34. Circle Detection Result of the Modified Motor Image

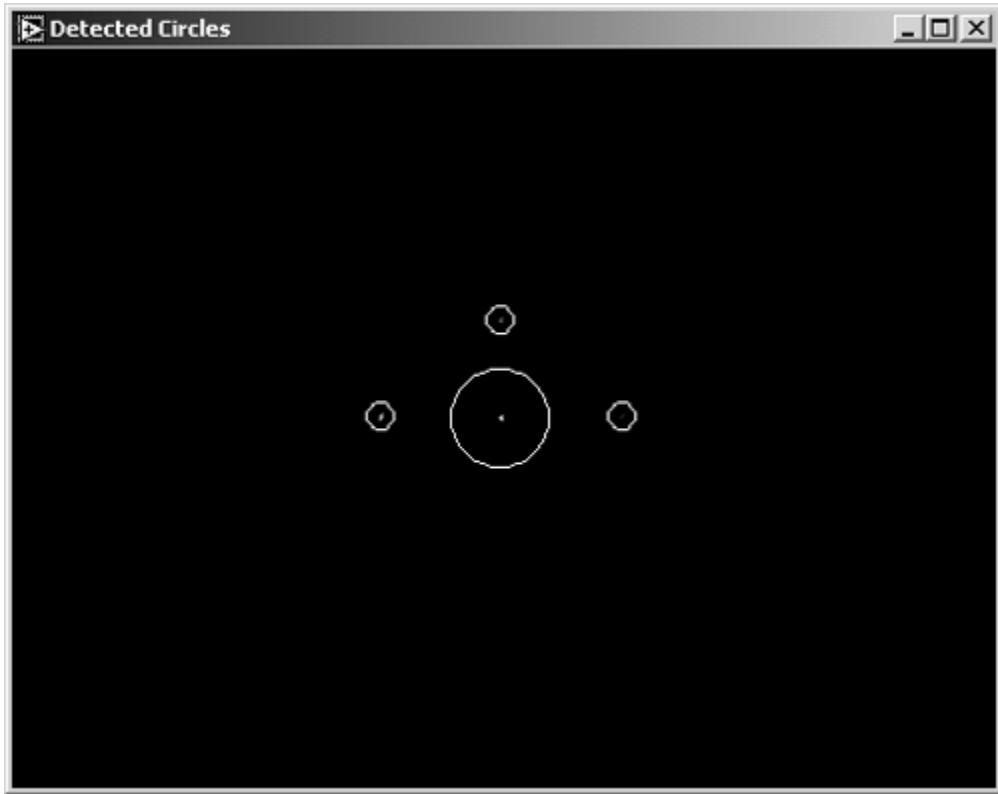
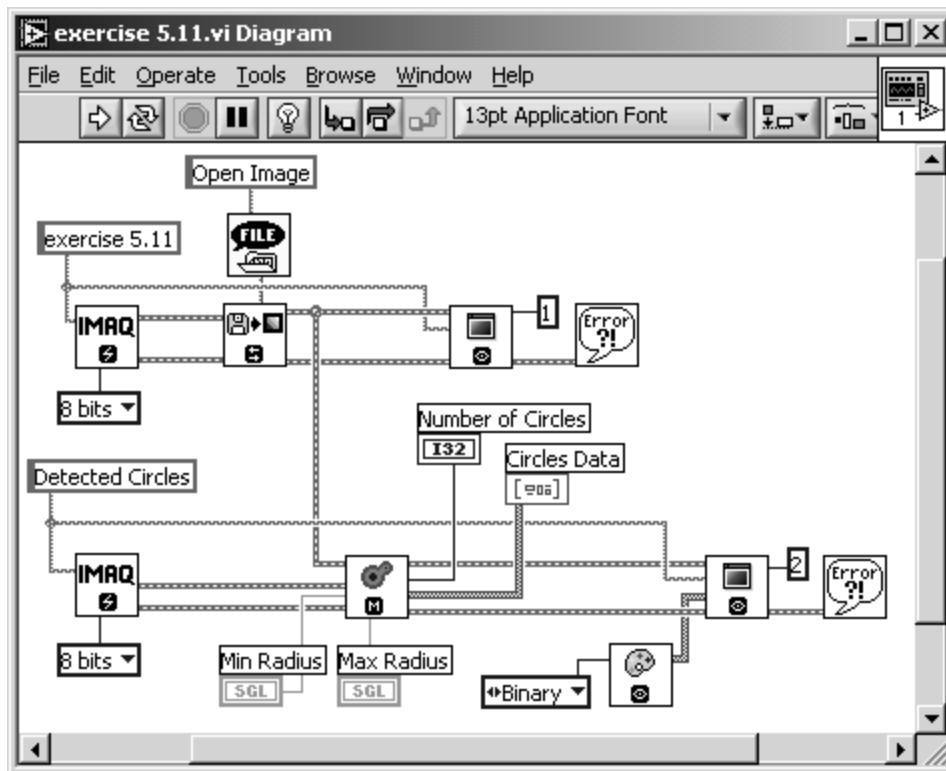


Figure 5.35. Diagram of [Exercise 5.11](#)



IMAQ Vision Builder 6.0
List of functions generated :
Sunday, 08. September 2002 18:54

STEP #1 Threshold : Manual Threshold
IMAQ Vision VI IMAQ Threshold
C Function imaqThreshold
Visual Basic Methods CWIMAQVision.Threshold
Parameters:
Range.Lower value Float (SGL) 30,118111
Range.Upper value Float (SGL) 189,744095

IMAQ Vision VI IMAQ Cast Image
C Function imaqCast
Visual Basic Methods CWIMAQVision.Cast
Parameters:
Image Type Long (I32) 0

Connections:
Connect output "Image Dst Out" of "IMAQ Threshold"
to input "Image Src" of "IMAQ Cast Image".
Connect output "error out" of "IMAQ Threshold" to
input "error in (no error)" of "IMAQ Cast Image".

STEP #2 Invert Binary Image
IMAQ Vision VI IVB Binary Inverse.vi
C Function imaqLookup
Visual Basic Methods CWIMAQVision.UserLookup
Parameters:

STEP #3 Adv. Morphology : Remove small objects
IMAQ Vision VI IMAQ RemoveParticle
C Function imaqSizeFilter
Visual Basic Methods CWIMAQVision.RemoveParticle
Parameters:

Connectivity 4/8 Boolean FALSE
Square / Hexa Boolean FALSE
Nb of Iteration Long (I32) 2
Low Pass / High Pass Boolean FALSE

STEP #4 Invert Binary Image
IMAQ Vision VI IVB Binary Inverse.vi
C Function imaqLookup
Visual Basic Methods CWIMAQVision.UserLookup
Parameters:

STEP #5 Adv. Morphology : Remove small objects
IMAQ Vision VI IMAQ RemoveParticle
C Function imaqSizeFilter
Visual Basic Methods CWIMAQVision.RemoveParticle
Parameters:
Connectivity 4/8 Boolean FALSE
Square / Hexa Boolean TRUE
Nb of Iteration Long (I32) 7
Low Pass / High Pass Boolean FALSE

STEP #6 Invert Binary Image
IMAQ Vision VI IVB Binary Inverse.vi
C Function imaqLookup
Visual Basic Methods CWIMAQVision.UserLookup
Parameters:

Comment: Display your image with a Binary palette.

[\[Team LiB \]](#)

[!\[\]\(47506e333d3b2d31ccde01308137cbf1_img.jpg\) PREVIOUS](#) [!\[\]\(30fd313e0e0ffb4639b6973c4dcb9150_img.jpg\) NEXT !\[\]\(3109d6278ca618f6f2aec7eb7bbe1874_img.jpg\)](#)

Quantitative Analysis

This section describes methods for the quantitative analysis of objects in binary images, including counting, measuring distances, and analyzing other object properties.

Particle Measurements

The following functions deal with various kinds of particle measurements, such as counting, clamping (measuring distances), and making other more complex measurements.

Counting Objects

The function `IMAQ Count Objects` is again a complete image analysis application, similar to the function `IMAQ Find Horizontal Edge` we used in [Exercise 5.6](#). It is possible to adjust various parameters, such as the search rectangle, the type of objects (bright or dark), whether holes in objects should be filled, or whether objects touching the border of the image are to be ignored. Because the function requires a gray-level image as input, the threshold value also has to be specified.

Exercise 5.12: Counting Objects.

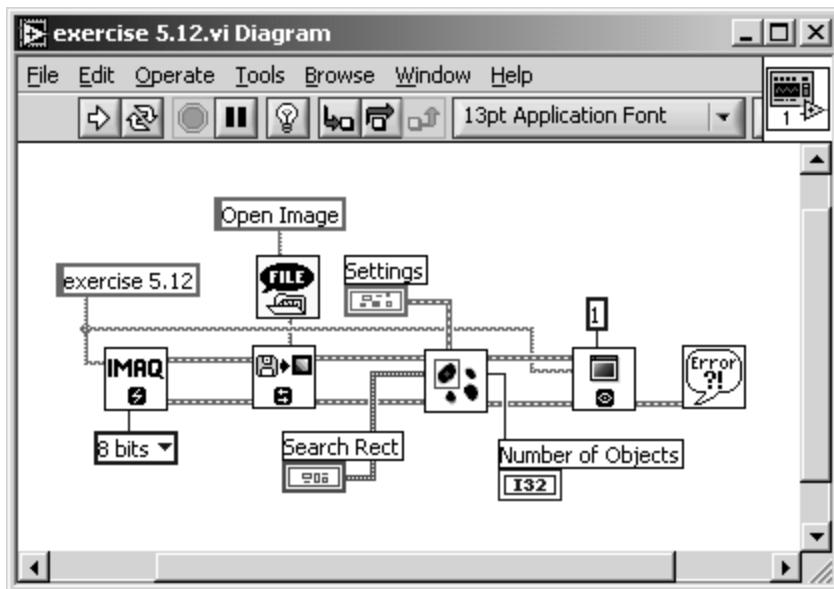
From [Exercise 5.6](#) (one of the "locating edges" exercises), replace the IMAQ function by `IMAQ Count Objects` (you will find the VI in Motion and Vision / Machine Vision / Count and Measure Objects). Add controls for the search rectangle and a cluster control for the settings, similar to the previous exercises. See [Figure 5.36](#) for the panel and [Figure 5.37](#) for the diagram.

The bottom-left corner of [Figure 5.36](#) shows the objects to count in IMAQ Vision Builder. The function `IMAQ Count Objects` shows the bounding box for each object, the center of each object, and the search area specified by the Search Rectangle cluster control.

Figure 5.36. Counting Objects in Gray-Scaled Images



Figure 5.37. Diagram of [Exercise 5.12](#)



Note that not only the big objects, such as the bear's face, are found, but also very small objects, which are hardly visible in the left part of [Figure 5.36](#). The range within objects that should be counted is adjusted by the controls Minimum Object Size and Maximum Object Size.

Measuring Distances (Clamping)

For our next exercise we examine a group of functions located in Motion and Vision / Machine Vision / Measure Distances. They are [IMAQ Clamp Horizontal Max](#), [IMAQ Clamp Horizontal Min](#), [IMAQ Clamp Vertical Max](#), and [IMAQ Clamp Vertical Min](#).

Again, these functions are complete vision applications and provide a number of outputs and

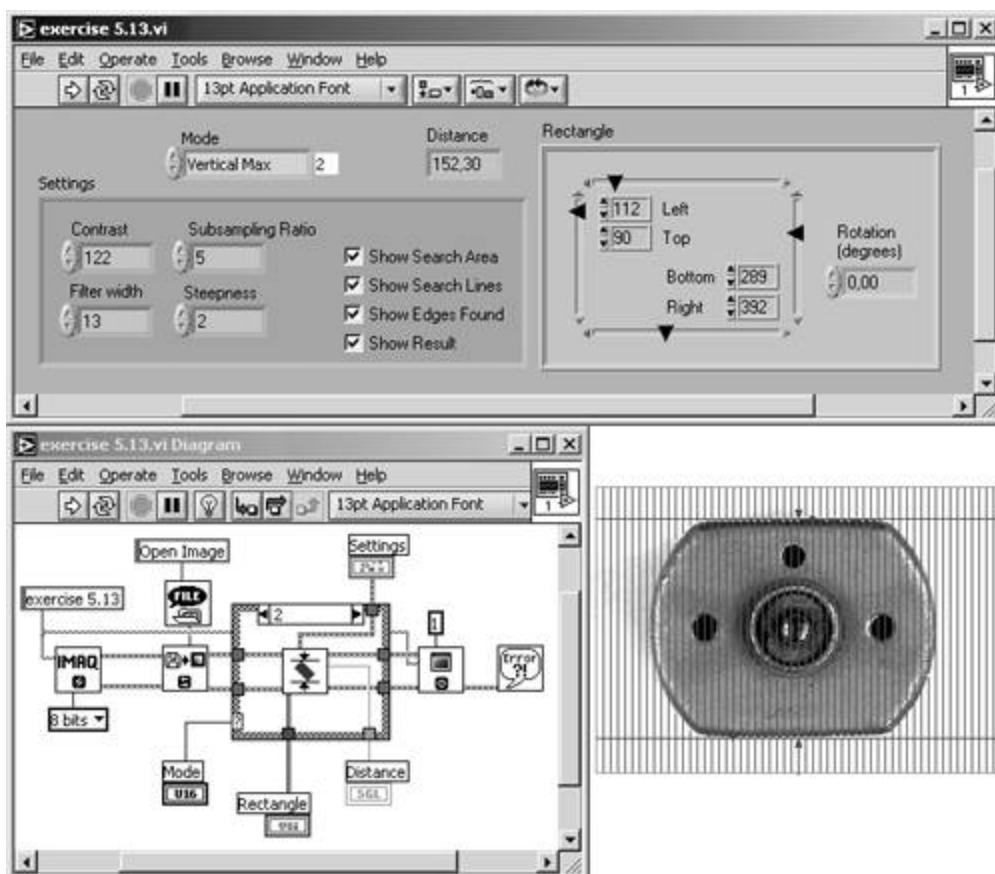
image overlays, similar to the "locating edge" examples. Using these VIs, we can measure the extensions of objects in gray-scaled images, either in a horizontal or vertical direction.

Exercise 5.13: Clamping Distances.

Modify [Exercise 5.6](#) (one of the "locating edges" exercises) by replacing the IMAQ function by **IMAQ Clamp Vertical Max**. Add controls for the search rectangle and a cluster control for the settings, as in the previous exercises.

In a next step, add a case structure to the VI, so that it is possible to select the desired function. Add a Text Ring control so that you can specify the function through the front panel. You can also add a numeric indicator for displaying the measured distance. See [Figure 5.38](#) for panel, diagram, and a first result showing the measuring of vertical maximum distances.

Figure 5.38. Measuring Vertical Maximum Distances



Naturally, the term "objects in gray-scaled images" I used a few sentences before is not correct, and you already know why: it is necessary to specify a threshold level to convert the gray-scaled image to a binary one. After that, you will find areas with pixel value 1 and other areas with pixel value 0; it depends on you, which of them you call "objects" and which of them "background."

The clamp functions use parameters called Contrast, Filter Width, Subsampling Ratio, and Steepness to define the contours of the objects that must be measured. See [Figures 5.39](#), [5.40](#), and [5.41](#) for measuring vertical minimum distances and horizontal maximum and minimum distances, respectively.

Figure 5.39. Measuring Vertical Minimum Distances

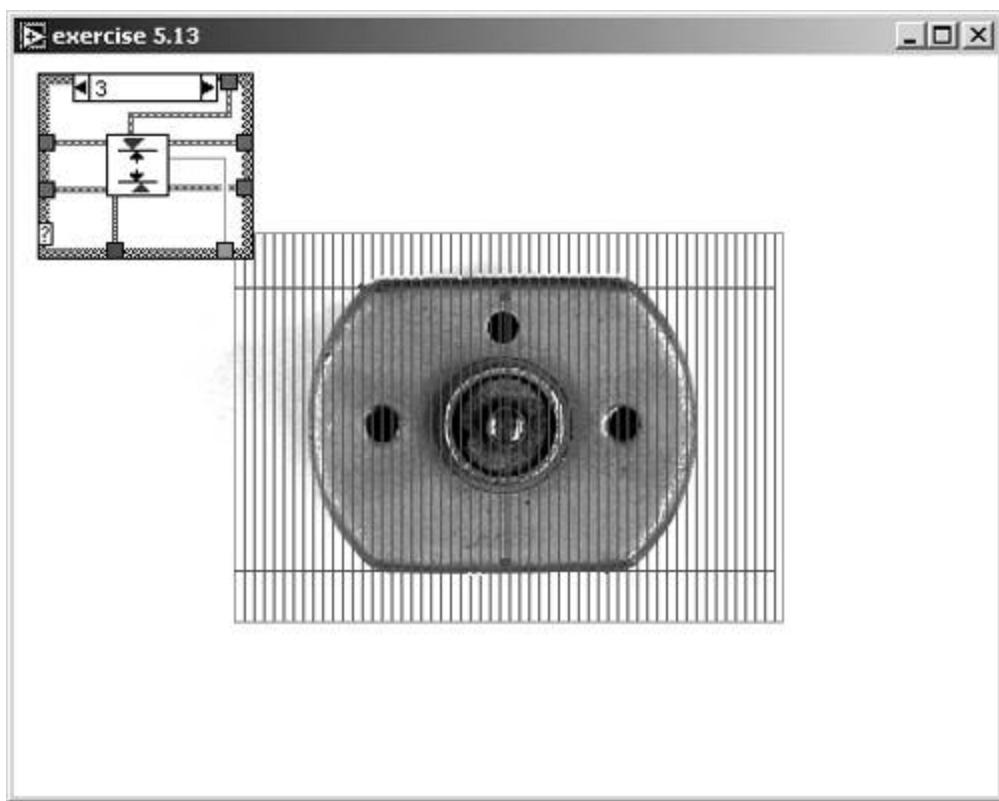


Figure 5.40. Measuring Horizontal Maximum Distances

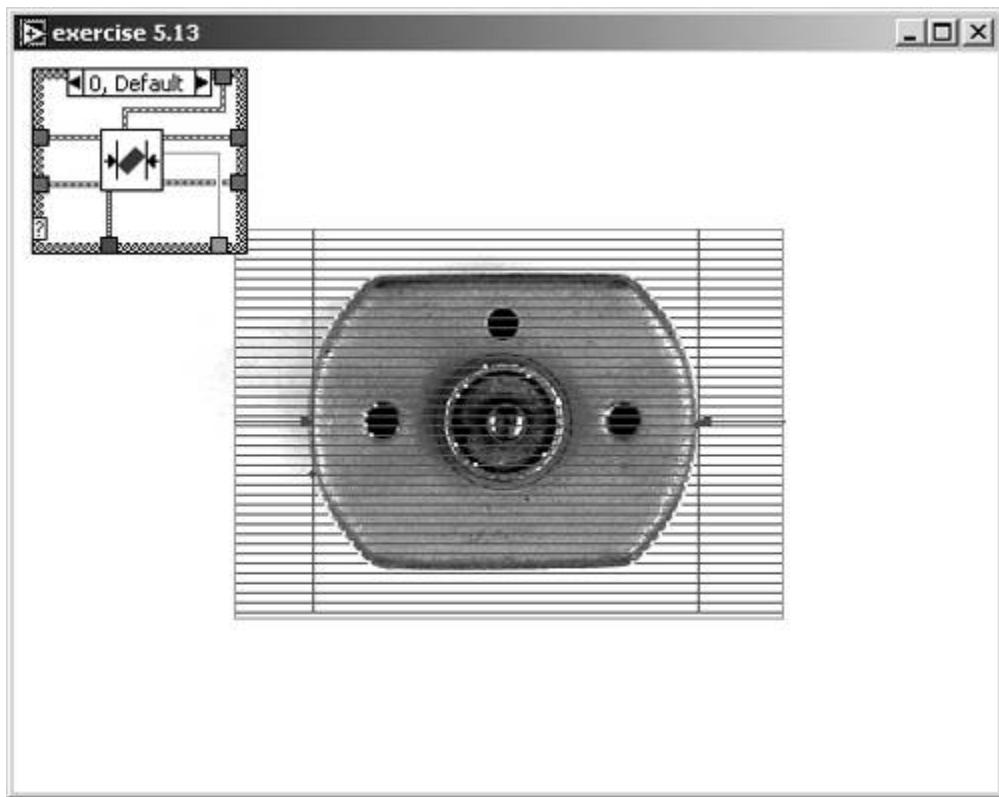
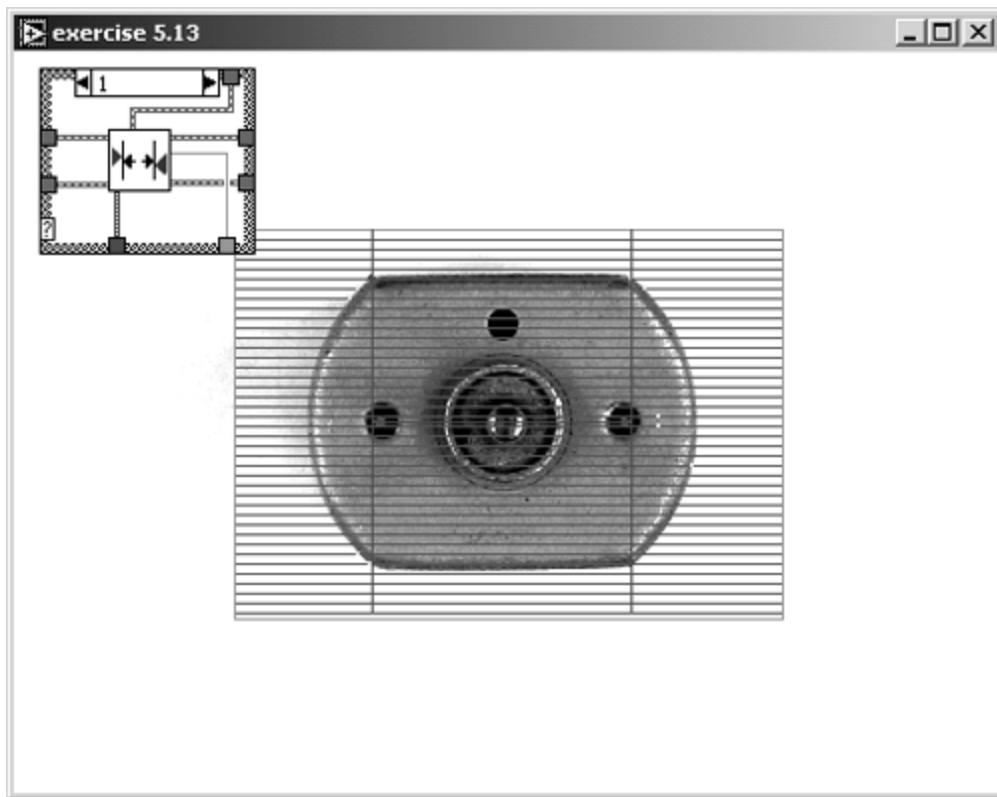


Figure 5.41. Measuring Horizontal Minimum Distances



Complex Particle Measurements

The next two functions we discuss in this section, `IMAQ_BasicParticle` and `IMAQ_ComplexParticle`, can be used for a detailed analysis of particles (objects) in binary images. The `basic particle` function provides only the following information:

- *Area (pixels)*: Surface area of particle in pixels;
- *Area (calibrated)*: Surface area of particle in user units;
- *Global Rectangle*, defined by the limits
 - left value,
 - top value,
 - right value, and
 - bottom value.

Exercise 5.14: Basic Particle Analysis.

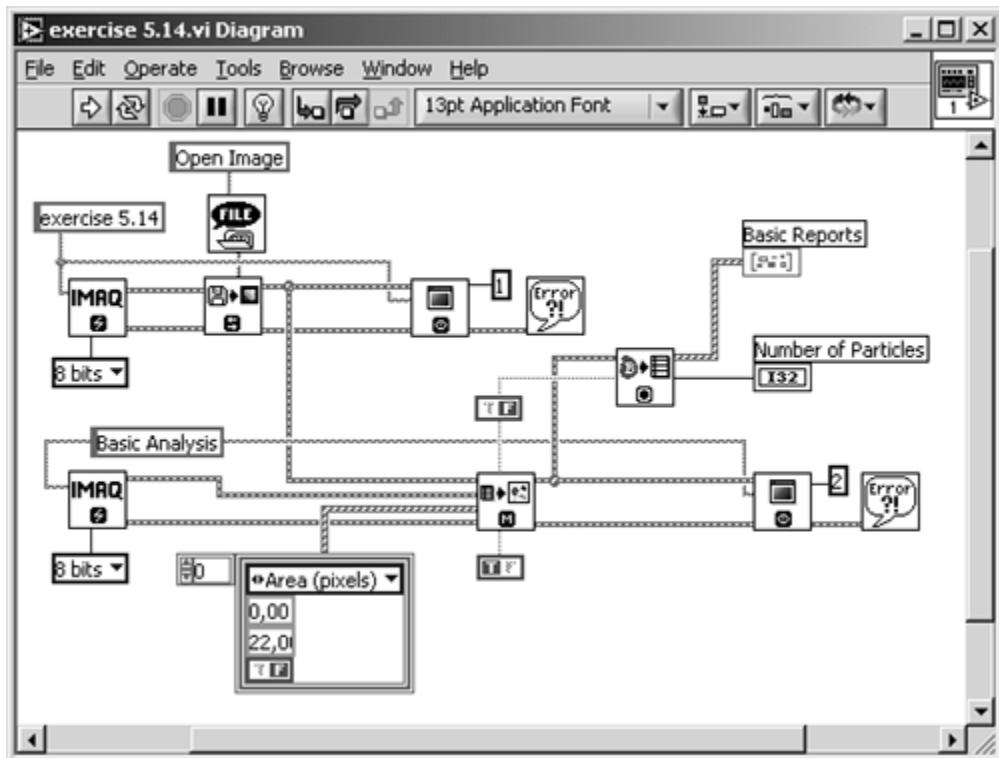
Start with [Exercise 4.16](#) and add the function `IMAQ_BasicParticle`. Change the Selection Values control to a constant, and filter the image so that not too many particles remain in your image. Add an indicator to display Basic Particle Reports.

[Figure 5.42](#) shows the panel of the exercise, the filtered image, and the labeled particles. I used IMAQ Vision Builder for the labeling, but you could include the labeling in this exercise. See also [Figure 5.43](#) for the diagram window.

Figure 5.42. Basic Particle Analysis of a Filtered Image



Figure 5.43. Diagram of [Exercise 5.14](#)



For our next exercise, which uses the function `IMAQ_ComplexParticle`, we simply replace the function `IMAQ_BasicParticle` in [Exercise 5.14](#). The Complex Particle function provides the following parameters:

- *Area (pixels)*;
- *Area (calibrated)*;
- *Perimeter*: Length of outer contour of particle in user units;
- *Number of holes*;
- *Holes area*: Surface area of holes in user units;
- *Hole perimeter*: Perimeter of all holes in user units;
- *Global rectangle*;
- *SumX*: Sum of the x axis for each pixel of the particle;
- *SumY*: Sum of the y axis for each pixel of the particle;
- *SumXX*: Sum of the x axis squared for each pixel of the particle;
- *SumYY*: Sum of the y axis squared for each pixel of the particle;
- *SumXY*: Sum of the x axis and y axis for each pixel of the particle;
- *Longest segment coordinates* with
 - Longest segment (X) and
 - Longest segment (Y);
- *Projection X*;
- *Projection Y*.

Exercise 5.15: Complex Particle Analysis.

Replace the IMAQ Vision function `IMAQ_BasicParticle` in [Exercise 5.14](#) with `IMAQ_ComplexParticle` and modify the indicator, here called Complex Reports, to display the results. See [Figure 5.44](#) and [Figure 5.45](#).

[Figure 5.44](#). Complex Particle Analysis of a Filtered Image

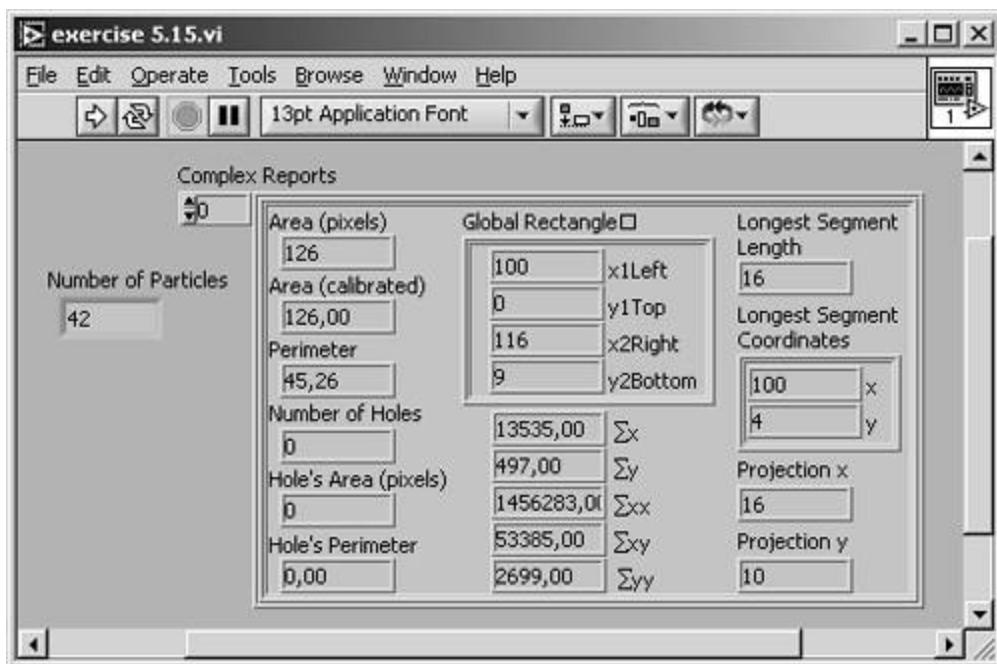
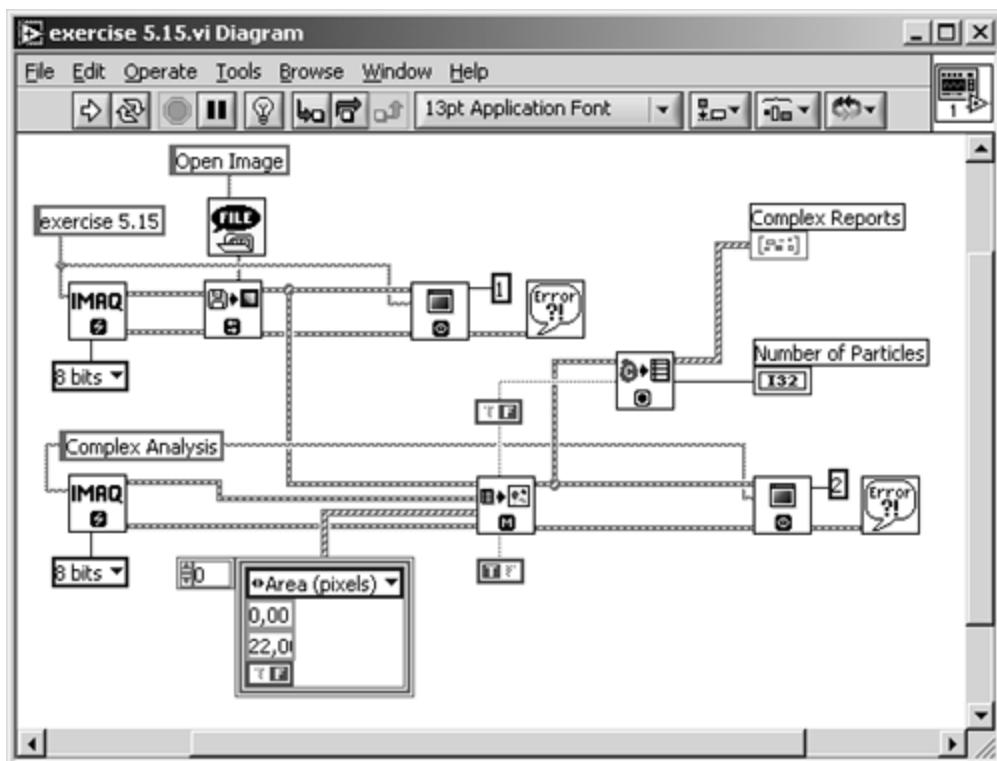


Figure 5.45. Diagram of [Exercise 5.15](#)



`IMAQ ChooseMeasurements`, another function of this group, filters the reports generated either by `IMAQ BasicParticle` or by `IMAQ ComplexParticle`, based on a selection parameter and minimum and maximum values. We need not try this function in an exercise here, because our VIs already provide particle filtering with the function `IMAQ ParticleFilter`.

To calculate the whole range of particle parameters, or at least one of the more complex ones we mentioned in [Chapter 4](#), you use the function `IMAQ ComplexMeasure`.

Add the IMAQ Vision function `IMAQ ChooseMeasurements` to [Exercise 5.15](#). Provide a control for choosing the parameter you want to measure, and provide a numeric 1D-array for the results. See [Figure 5.46](#) and [Figure 5.47](#).

Figure 5.46. Calculating Other Complex Particle Parameters

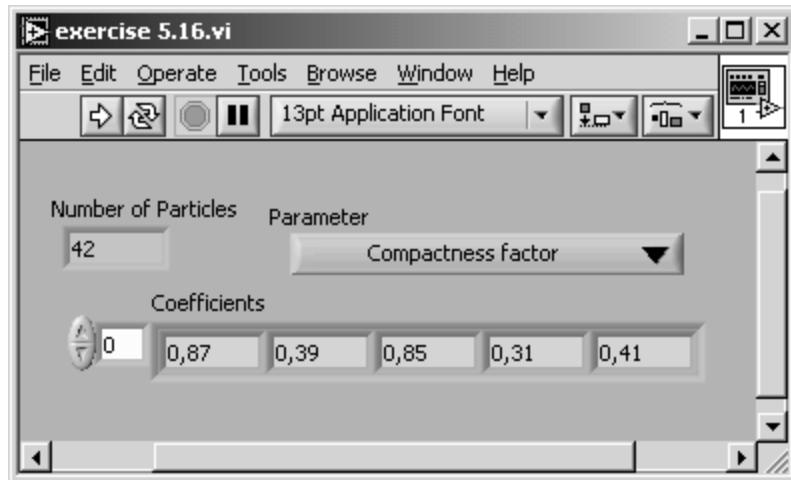
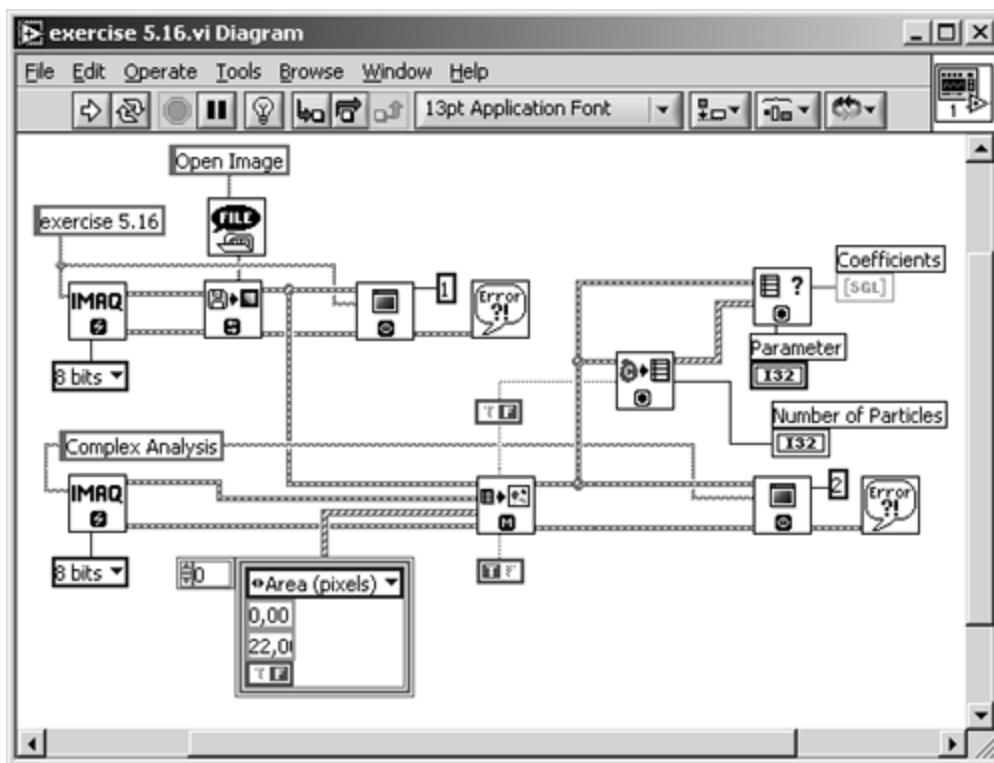


Figure 5.47. Diagram of [Exercise 5.16](#)



In the next pages we discuss in more detail the parameters that can be measured by `IMAQ ComplexParticle` or calculated by `IMAQ ChooseMeasurements`. You can also find these details in [4].

- *Area (pixels)*: Surface area of particle in number of pixels. Holes in particles are not included.

- *Area (calibrated) A*: Surface area of particle in user units. We learn about calibration of images later in this chapter.
- *Number of holes*: Number of holes within a particle.
- *Holes' area A_h*: Surface area of all holes within a particle.
- *Total area A_t*: Area of particle including areas of all holes:

Equation 5.1

$$A_t = A + A_h \quad .$$

- *Scanned area A_s*: Surface area of the entire image in user units:

Equation 5.2

$$A_s = (\text{Resolution } x \times x\text{Step}) \\ \times (\text{Resolution } y \times y\text{Step}) \quad .$$

- *Ratio area/scanned area R_p(%)*: Percentage of the image occupied by particles:

Equation 5.3

$$R_p = \frac{A}{A_s} \quad .$$

- *Ratio area/total area R_t(%)*: Percentage of particle surface in relation to the total area:

Equation 5.4

$$R_t = \frac{A}{A_t} \quad .$$

- *Center of mass (X)*: *x* coordinate of the center of gravity. In general, the center of gravity *G* of a particle, which consists of *n* pixels *P_i*, is defined by

Equation 5.5

$$\overline{OG} = \frac{1}{n} \sum_{i=1}^n \overline{OP_i} \quad ;$$

the *x* coordinate of *G* is therefore

Equation 5.6

$$x_G = \frac{1}{n} \sum_{i=1}^n x_{P_i} \quad ,$$

which is the average location of the central points of the horizontal segments in a particle [4].

- *Center of mass (Y)*: y coordinate of the center of gravity:

Equation 5.7

$$y_G = \frac{1}{n} \sum_{i=1}^n y_{P_i} .$$

- *Left column (X)*: Left x coordinate of bounding rectangle;
- *Upper row (Y)*: Top y coordinate of bounding rectangle;
- *Right column (X)*: Right x coordinate of bounding rectangle;
- *Lower row (Y)*: Bottom y coordinate of bounding rectangle. If x_i and y_i are the coordinates of a pixel P_i in a particle, there exists a horizontal rectangle defined by two points $(\min x_i, \min y_i)$ and $(\max x_i, \max y_i)$:

Equation 5.8

$$\begin{aligned} \text{left column (X)} &= \min x_i , \\ \text{upper row (Y)} &= \min y_i , \\ \text{right column (X)} &= \max x_i , \\ \text{lower row(Y)} &= \max y_i . \end{aligned}$$

- *Width l*: Width of bounding rectangle:

Equation 5.9

$$l = \max x_i - \min x_i .$$

- *Height h*: Height of bounding rectangle:

Equation 5.10

$$h = \max y_i - \min y_i .$$

- *Longest segment length*: Length of longest horizontal line segment.
- *Longest segment left column (X)*: Leftmost x coordinate of longest horizontal line segment.
- *Longest segment row (Y)*: y coordinate of longest horizontal line segment.
- *Perimeter p*: Length of the outer contour of a particle.
- *Hole perimeter*: Perimeter of all holes in user units.
- *SumX*: Sum of the x coordinates of each pixel in a particle.
-

- *SumY*: Sum of the y coordinates of each pixel in a particle.
- *SumXX*: Sum of the x coordinates squared for each pixel in a particle.
- *SumYY*: Sum of the y coordinates squared for each pixel in a particle.
- *SumXY*: Sum of the x and y coordinates for each pixel in a particle.
- *Corrected projection x*: Sum of the horizontal segments that do not superimpose any other horizontal segment.
- *Corrected projection y*: Sum of the vertical segments that do not superimpose any other vertical segment.
- *Moment of inertia I_{xx}*: All moments of inertia are a representation of the pixel distribution of a particle in relation to its center of gravity. If A is the particle area and x_G the x coordinate of the center of gravity, the moment of inertia I_{xx} is defined as

Equation 5.11

$$I_{xx} = \sum x^2 - A \cdot x_G^2 .$$

- *Moment of inertia I_{yy}*: If y_G is the y coordinate of the center of gravity, the moment of inertia I_{yy} is defined as

Equation 5.12

$$I_{yy} = \sum y^2 - A \cdot y_G^2 .$$

- *Moment of inertia I_{xy}*:

Equation 5.13

$$I_{xy} = \sum xy - A \cdot x_G \cdot y_G .$$

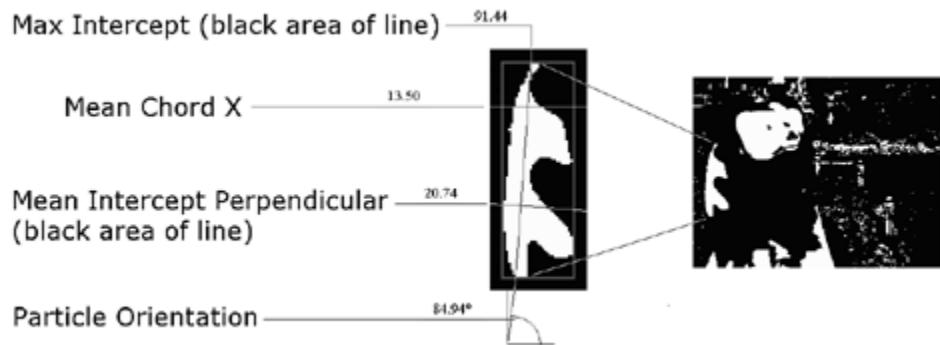
- *Mean chord X C_x*: Mean length of horizontal segments in a particle.
- *Mean chord Y C_y*: Mean length of vertical segments in a particle.
- *Max intercept S_{max}*: Length of longest segment in a particle. All possible directions have to be considered.
- *Mean intercept perpendicular C*: Mean length of the chords in an object perpendicular to its max intercept. Another description possibility is

Equation 5.14

$$C = \frac{A}{S_{\max}} .$$

- *Particle orientation*: Direction of the longest segment in a particle. [Figure 5.48](#) shows an example of the last five parameters.

Figure 5.48. Intercept and Chord Particle Measurements



- *Equivalent ellipse minor axis:* Imagine an ellipse having the same area as the particle and a major axis a equal to half the max intercept S_{\max} . The minor axis b of this ellipse is

Equation 5.15

$$b = \frac{4 \cdot A}{2\pi \cdot S_{\max}}$$

if the particle area A is

Equation 5.16

$$A = \pi a b$$

- *Ellipse major axis:* Now imagine another ellipse having the same area and the same perimeter as the particle. The ellipse major axis is the total length of the major axis of this ellipse.
- *Ellipse minor axis:* Total length of minor axis having the same area and perimeter as the particle.
- *Ratio of equivalent ellipse axis R_e :* Fraction of major axis to minor axis of the equivalent ellipse:

Equation 5.17

$$R_e = \frac{a}{b}$$

- *Rectangle big side:* Length of the large side of a rectangle having the same area and perimeter as the particle.
- *Rectangle small side:* Length of the small side of a rectangle having the same area and perimeter as the particle.
- *Ratio of equivalent rectangle sides:* Ratio of rectangle big side to rectangle small side.
- *Elongation factor F_e :* The ratio of the longest segment in a particle to the mean length of the perpendicular segments:

Equation 5.18

$$F_e = \frac{S_{\max}}{C} \quad .$$

- *Compactness factor* F_c : The ratio of the particle area A to the area of the bounding rectangle (always between 0 and 1; or equal to 1 if the particle is rectangular):

Equation 5.19

$$F_c = \frac{A}{h \cdot l} \quad .$$

- *Heywood circularity factor* F_H : The ratio of the particle perimeter p to the perimeter of a circle having the same area A as the particle:

Equation 5.20

$$F_H = \frac{p}{2\sqrt{\pi \cdot A}} \quad .$$

- *Type factor* F_t : A complex factor relating the surface area A to the moment of inertia:

Equation 5.21

$$F_t = \frac{A^2}{4\pi\sqrt{I_{xx} \cdot I_{yy}}} \quad .$$

- *Hydraulic radius* R_h : The ratio of the particle area A to the particle perimeter p :

Equation 5.22

$$R_h = \frac{A}{p} \quad .$$

- *Waddel disk diameter* R_d : Diameter of the disk having the same area as the particle:

Equation 5.23

$$R_d = 2 \cdot \sqrt{\frac{A}{\pi}} \quad .$$

- *Diagonal d*: Diagonal of the bounding rectangle:

Equation 5.24

$$d = \sqrt{l^2 + h^2} \quad .$$

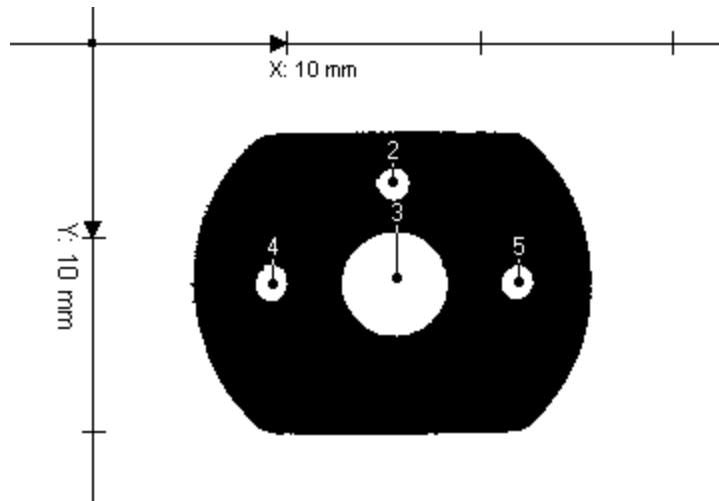
Image Calibration

Sometimes it is necessary to convert the x and y dimensions of an image into real units; for example, in our motor image, we can measure some "real" distances on the motor itself.

Before we start a new exercise, let us look at the relevant functions in IMAQ Vision Builder. Load a binary motor image (`dc_motor_bin3.png`) and select Image / Simple Calibration. Use the menu now displayed to define two main topics:

- *Origin and orientation:* You can draw a line in the image and thus define the origin and the x -axis orientation of a new coordinate system. If the object to measure is already in a correct orientation and you want to measure only distances, not absolute coordinates (which is true for our motor image), you can skip this menu item.
- *Pixel calibration:* Here, you can define a factor for the x and the y direction as well as the unit for the new dimensions (in our example, "millimeter" is appropriate). You can also set different scaling factors for x and y direction (see [Figure 5.49](#)).

Figure 5.49. Calibrating the Motor Image



With the motor image selected, draw a vertical line from the top to the bottom of the motor, thus specifying the smallest possible distance; this action returns an (image) distance of about $n = 150$ pixels. The real distance is about $d = 15.4$ mm, which you must enter in the appropriate control.

The new dimension is now displayed in the coordinate system. The image is now ready for simple or complex particle analysis, which is based on user units; for example, measuring height, width, area, perimeter, and some other parameters in real-world units.

Exercise 5.17: Image Calibration.

Here, we try image calibration in a LabVIEW exercise: In [Exercise 5.16](#), insert the function `IMAQ Set Simple Calibration` (you can find it in Motion and Vision / Vision Utilities / Calibration) between `IMAQ Particle Filter` (which is not really necessary here) and `IMAQ WindDraw`. Provide an additional control for setting the grid description. As mentioned above, it is not necessary here to change orientation and origin of the coordinate system. See [Figure 5.50](#) and [Figure 5.51](#).

Figure 5.50. Setting a Simple Pixel Calibration

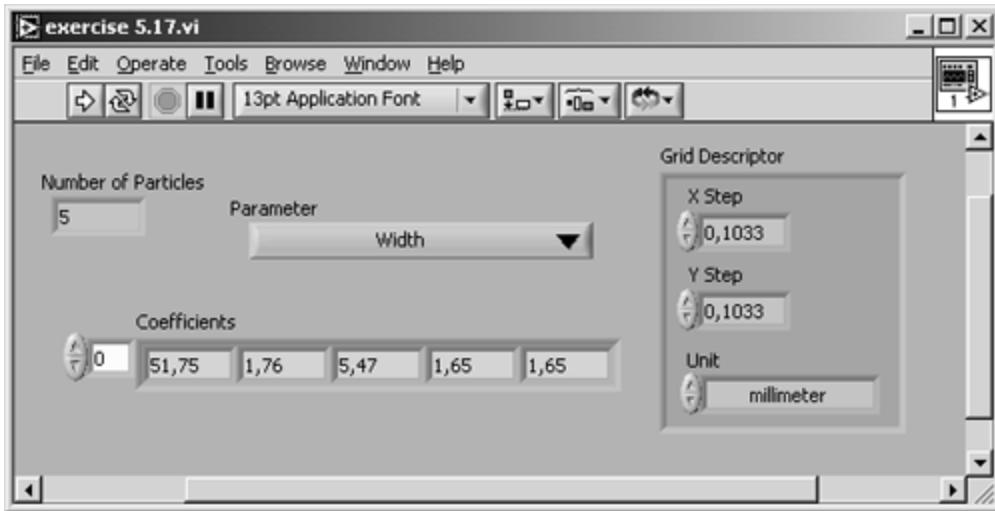
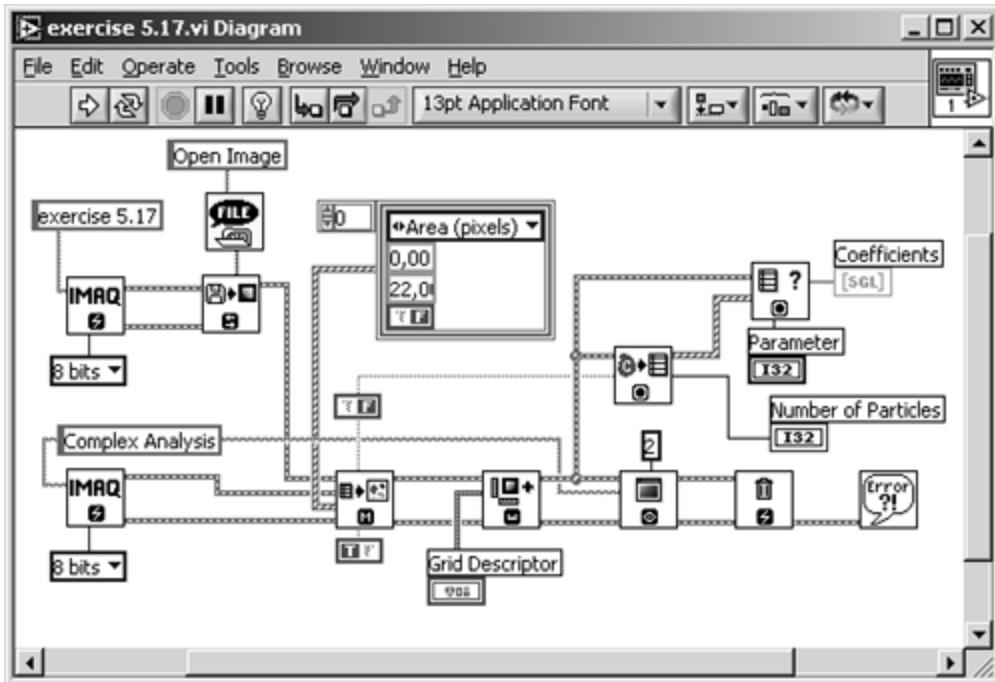


Figure 5.51. Diagram of [Exercise 5.17](#)



We can calculate the correct factors "X Step" and "Y Step" in [Exercise 5.17](#) by using the values we obtained from IMAQ Vision builder. The value f for both directions is

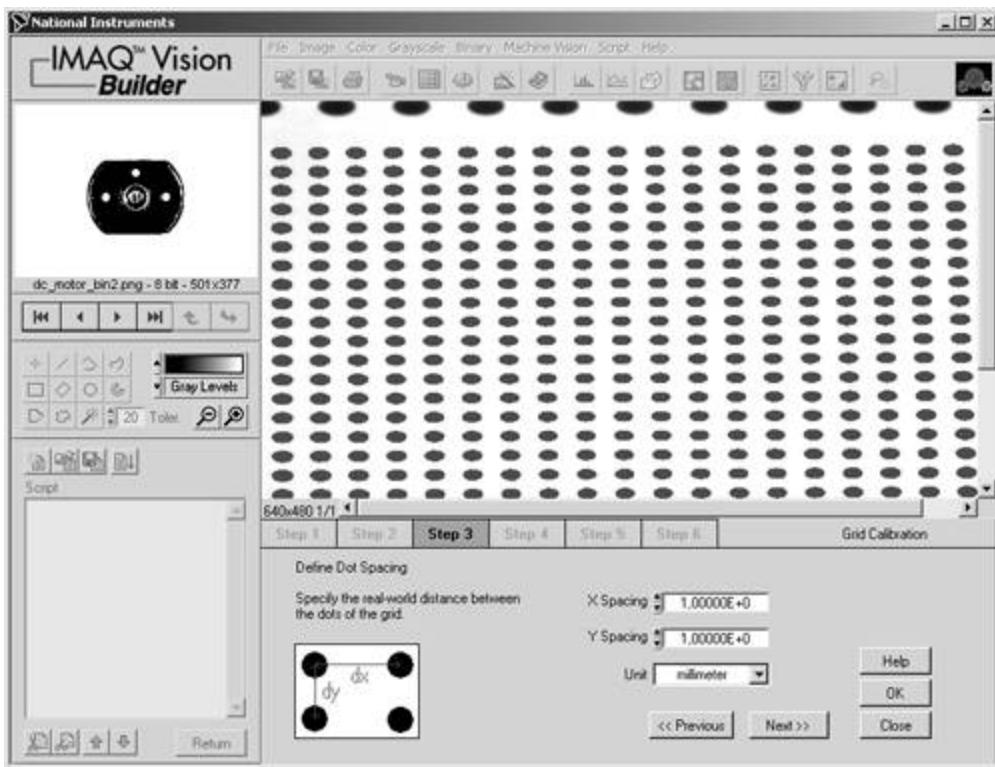
Equation 5.25

$$f = \frac{\text{real distance}}{\text{image distance}} = \frac{d}{n} = \frac{15.4 \text{ mm}}{150 \text{ pixels}} \approx 0.103 \text{ mm/pixels}$$

See also [Figure 5.50](#).

Another possibility is to use a *grid image*, showing circular dots specifying the distortion of an image ([Figure 5.52](#)). Calibration of the image (we use IMAQ Vision Builder here) is done by the following steps:

Figure 5.52. Grid Calibration with IMAQ Vision Builder



1. The template image, showing the calibration grid, is loaded.
2. The grid image is converted into a binary image. An appropriate threshold must be specified.
3. The real-world distance between the grid dots in x and y direction must be specified.
4. A new coordinate system can be defined, similar to the system in simple calibration example ([Figure 5.49](#)).
5. In case of a perspective or nonlinear distortion, these parameters must be specified.
6. The calibration data is now learned and can be used to display corrected images.

[\[Team LiB \]](#)

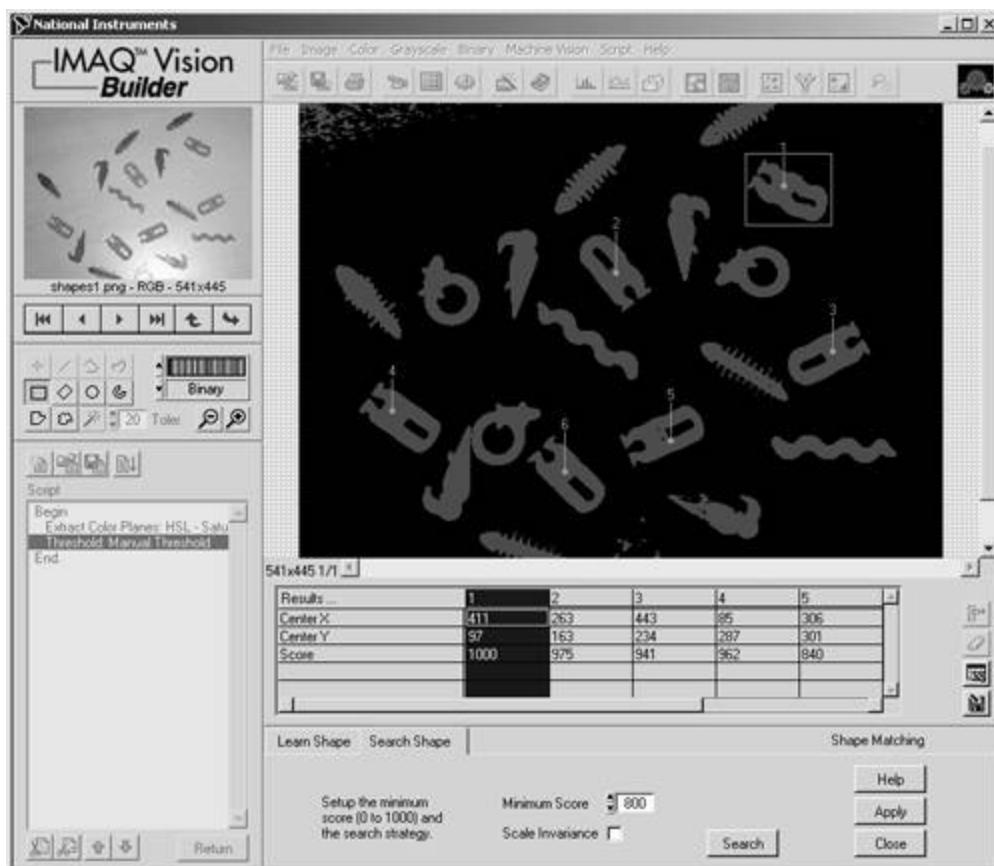
[◀ PREVIOUS](#) [NEXT ▶](#)

Shape Matching

(Binary) shape matching is a tool that can be used if a Machine Vision application has to detect corresponding or similar shapes; for example, if different parts in a production process have to be sorted or defective parts should be detected.

The image, which is the base for the shape matching process, has to be binary; so it has to be converted from a gray-level or color image. An example is on the attached CD-ROM; the color image is called `shapes1.png`. The following script can be used to convert the image to a binary one.

Figure 5.53. Shape Matching with IMAQ Vision Builder



IMAQ Vision Builder 6.0

List of functions generated :

Tuesday, 15. October 2002 14:15

STEP #1 Extract Color Planes : HSL - Saturation Plane

IMAQ Vision VI IMAQ ExtractSingleColorPlane

C Function imaqExtractColorPlanes

Visual Basic Methods CWIMAVision.ExtractColorPlanes

Parameters:

Color Plane Long (I32) 4

STEP #2 Threshold : Manual Threshold

```

IMAQ Vision VI IMAQ Threshold
C Function imaqThreshold
Visual Basic Methods CWIMAQVision.Threshold
Parameters:
Range.Lower value Float (SGL) 34,133858
Range.Upper value Float (SGL) 255,000000

IMAQ Vision VI IMAQ Cast Image
C Function imaqCast
Visual Basic Methods CWIMAQVision.Cast
Parameters:
Image Type Long (I32) 0

Connections:
Connect output "Image Dst Out" of "IMAQ Threshold"
    to input "Image Src" of "IMAQ Cast Image".
Connect output "error out" of "IMAQ Threshold"
    to input "error in (no error)" of
"IMAQ Cast Image".

Comment: Display your image with a Binary palette.

```

Figure 5.53 shows a shape matching result of the shapes image. The entire process is divided into two parts:

- First, we specify a template that defines the shape to be searched. We do this either by drawing a region of interest in the image or by loading a previously saved template file.
- For the search process itself (see Figure 5.53), we can specify the minimum score value, which is a number from 0 to 1000 and is used to indicate the quality of the matching process (a perfect match would score 1000), and we can specify whether objects with the same shape but of different size from the template should be detected (scale invariance).

You can try this example with a number of different templates. Note that the conversion from color to gray-level uses the saturation plane of the HSL color model, leading to optimal results in the subsequent threshold operation.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

Pattern Matching

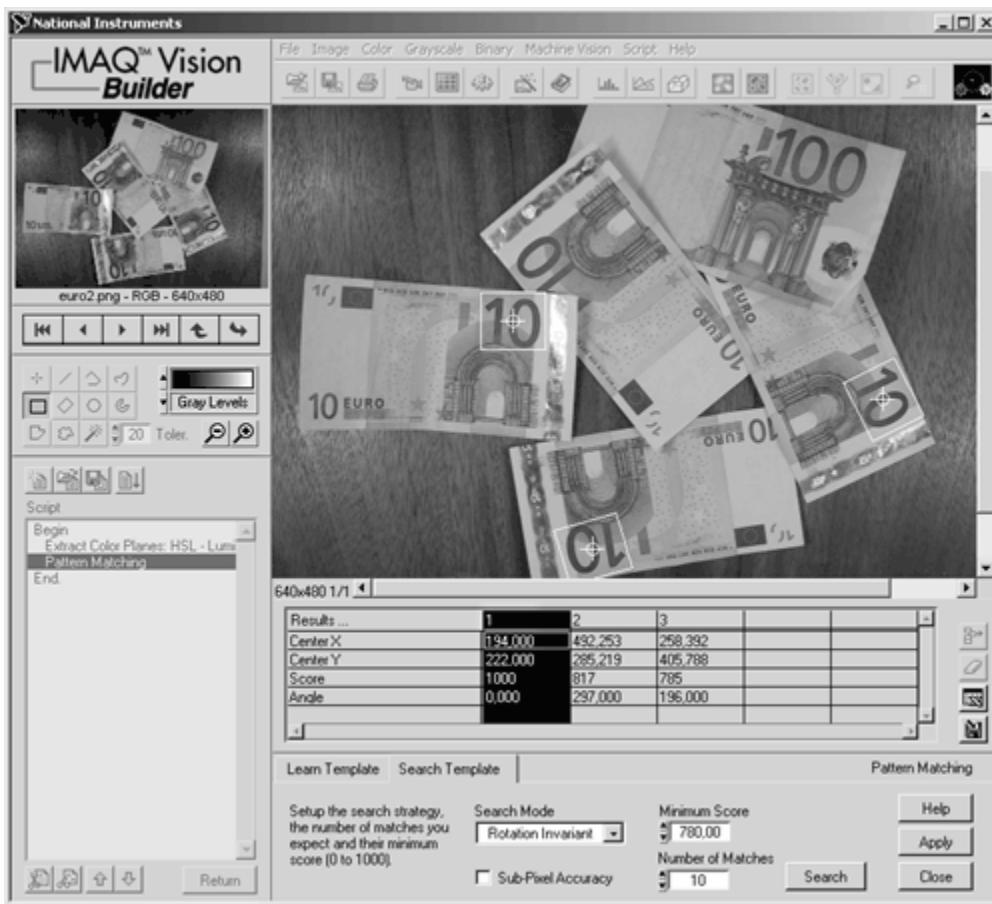
Pattern matching is arguably one of the most important image analysis tools and is very often the first step in a Machine Vision application. In general, pattern matching provides you with information about the position and the number of instances of a previously defined template (pattern).

Introduction

We can try a first pattern matching example in IMAQ Vision Builder. The procedure is similar to the previous shape matching example, with the main difference that the image has to be a gray-scaled one:

1. Define a template (ROI or file).
2. Search for instances of the template (see also [Figure 5.54](#)).

Figure 5.54. Pattern Matching with IMAQ Vision Builder



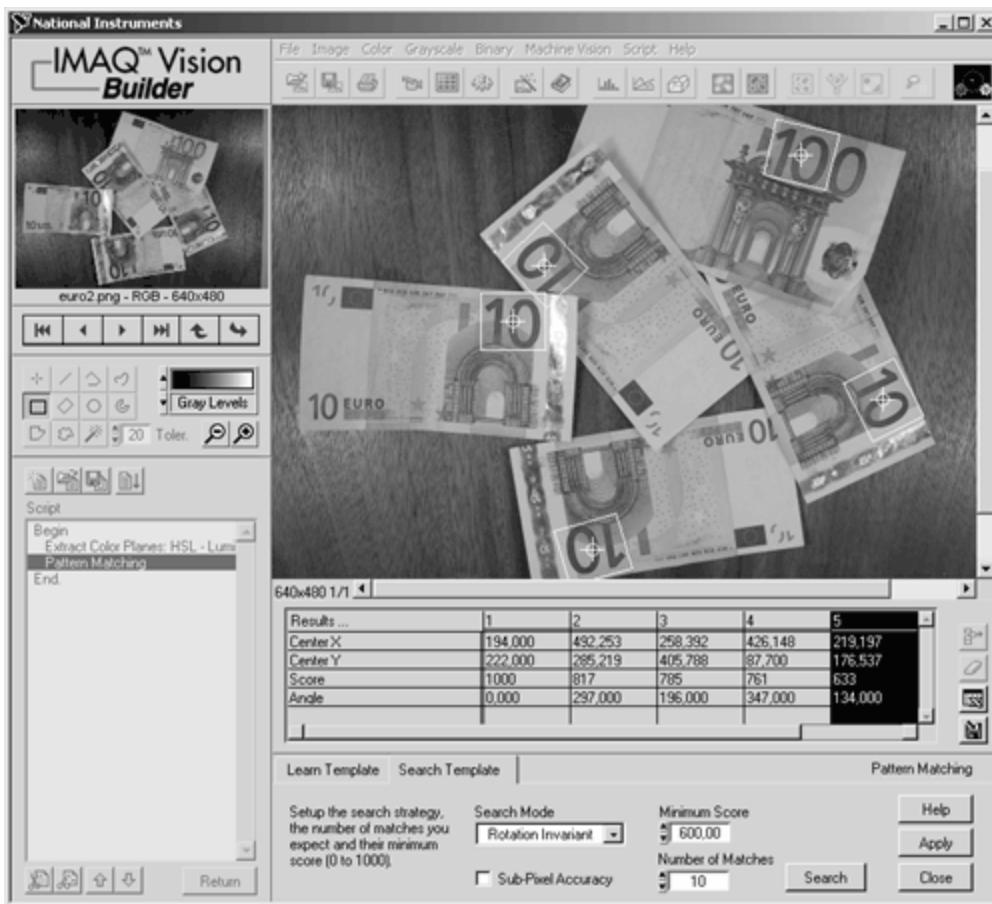
[Figure 5.54](#) also shows the parameters of the pattern matching process:

- the maximum number of instances that can be found;

- the minimum matching score (see also the shape matching example);
- the selection between *shift invariant* and *rotation invariant* mode;
- Shift invariant mode does not allow rotation of potential matches in relation to the template; this means that only matches with the same orientation as the template are found.
- Rotation invariant allows matches in any direction; this mode is significantly slower than the shift invariant mode.

[Figure 5.55](#) shows the result with the same image if the minimum matching score is decreased to 600; now the partly hidden 10 and the first two digits of the 100 note are found, although the 100 is a different color.

Figure 5.55. Pattern Matching: Decreasing Matching Score



Pattern Matching Techniques

The mathematical basis of pattern matching is the *cross correlation* function. The correlation $C(i,j)$ is defined as

Equation 5.26

$$C(i,j) = \sum_{x=0}^{l-1} \sum_{y=0}^{k-1} w(x,y) f(x+i, y+j) ,$$

where $w(x,y)$ is a subimage (template) of the size $k \times l$, $f(x,y)$ is the original image of the size $m \times n$ (where $k \leq m$ and $l \leq n$); $i = 0, 1, \dots, m-1$, $j = 0, 1, \dots, n-1$ [4].

[Figure 5.56](#) shows the front panel of a LabVIEW VI provided on the attached CD-ROM. The program is similar to the IMAQ Vision Builder functionality and allows definition of a template and the matching process itself. Moreover, the orientation of the (first) match is displayed with a gauge control. See also [Figure 5.57](#) for the diagram, showing the main loop of the pattern matching process.

Figure 5.56. Pattern Matching: Single Match

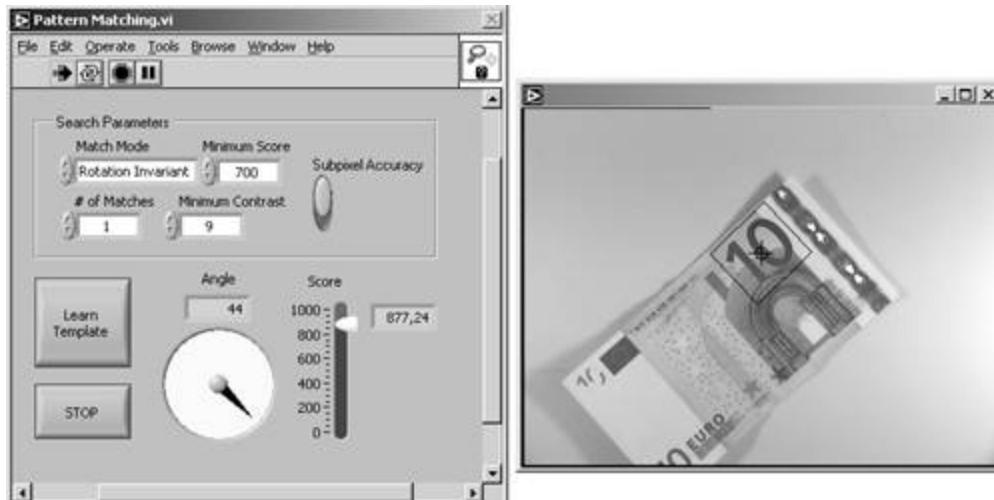
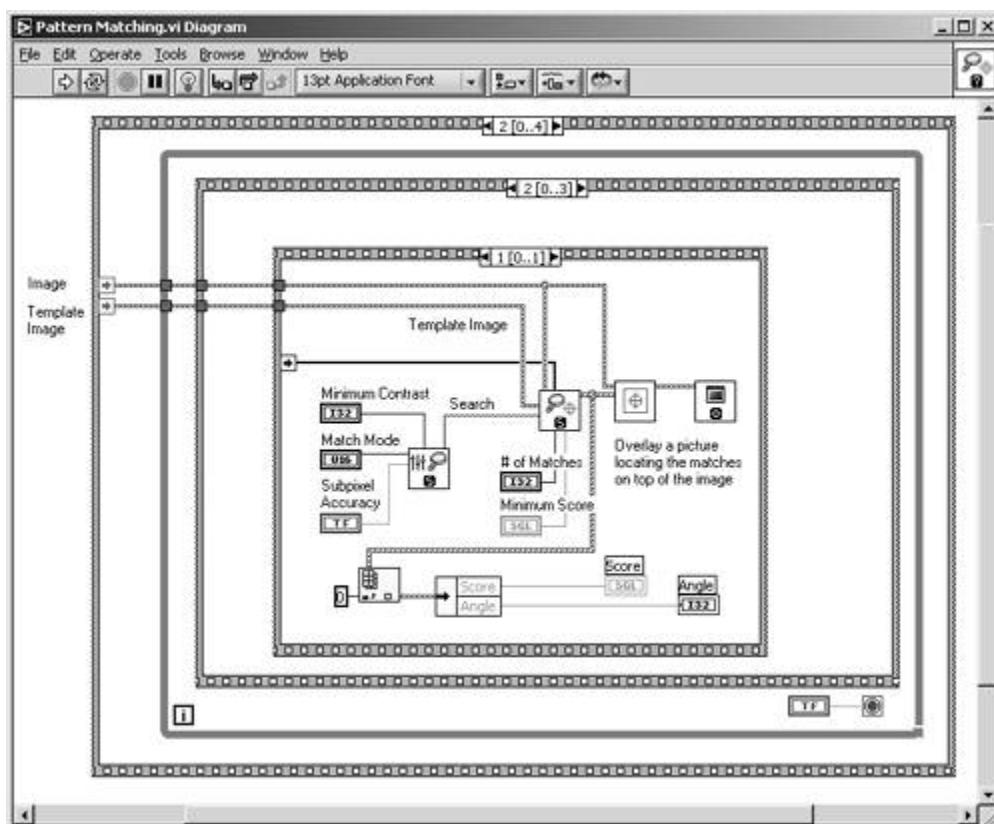


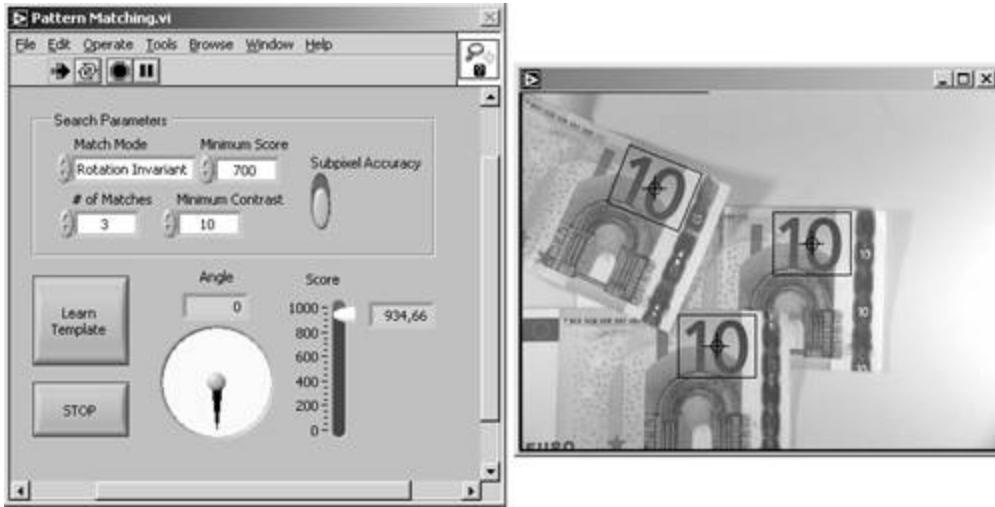
Figure 5.57. Part of the Pattern Matching Diagram



The same application finding multiple instances of the template can be seen in [Figure 5.58](#). The Number (#) of Matches control has to be increased to the desired maximum number of

instances. Note that the matches found in this example as well as in the Vision Builder are not scale invariant, which means that all instances have to be the same size as the template.

Figure 5.58. Pattern Matching: Multiple Match



Another problem with Eq. (5.26) is its high sensitivity to amplitude changes of the template as well as of the image, which has to be searched for matches. Amplitude changes are caused by different lighting conditions of new images or by brightness changes of the original image.

An normalized correlation coefficient $R(i,j)$, as in

Equation 5.27

$$R(i,j) = \frac{\sum_{x=0}^{l-1} \sum_{y=0}^{k-1} (w(x,y) - \bar{w}) (f(x+i, y+j) - \bar{f}(i,j))}{\sqrt{\sum_{x=0}^{l-1} \sum_{y=0}^{k-1} (w(x,y) - \bar{w})^2} \sqrt{\sum_{x=0}^{l-1} \sum_{y=0}^{k-1} (f(x+i, y+j) - \bar{f}(i,j))^2}}$$

solves this problem: \bar{w} is the average intensity value of the pixels in the template w , and \bar{f} is the average intensity value of the coincident image area f , which leads to a range for the value of R from -1 to 1 . R is therefore independent of intensity changes of f and w [4].

The result of the function **IMAQ Match Pattern** is a data set for all matches found in the original image. In particular, the center coordinates and the rotation angle (in the case of rotation invariant matching) can be used for further image analysis, for example, for the definition of a new coordinate system.

IMAQ Vision also contains functions for color pattern matching, for example, **IMAQ Color Match Pattern**. These functions combine the gray-level pattern matching functions with the matching of color information. The results of both algorithms are used to calculate the total matching score. You can try these functions, using IMAQ Vision Builder.

Reading Instrument Displays

A nice set of examples, prepared by National Instruments and almost ready for use, is the instrument-reading functions of IMAQ Vision. Two groups, the reading of an analog needle instrument and the reading of an LCD (liquid crystal display) meter, are provided.

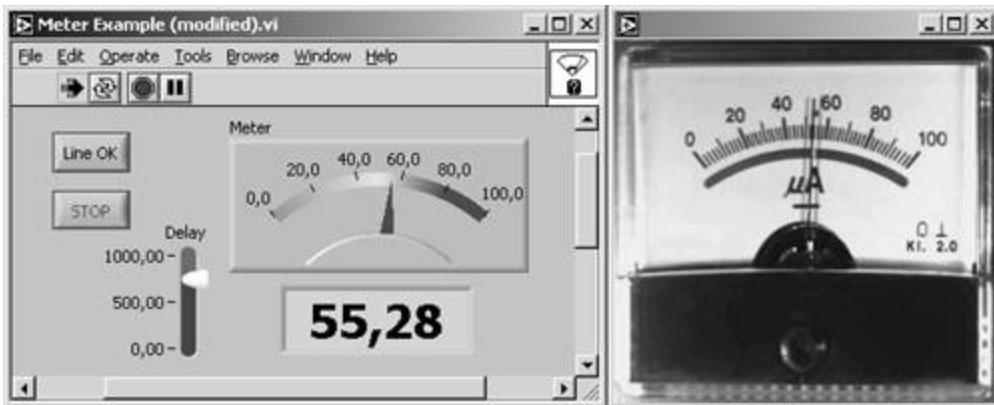
These examples are really self-explaining, so we need not build our own exercises. However, I added some small modifications; you can add your own images to these example programs. If you look at the block diagrams, you will find out that adding images is not difficult.

Analog Instrument Displays

Reading analog needle instruments is an interesting and challenging task. Unlike modern digital instruments, these devices do not have a microcontroller and an interface to a PC, so it is usually impossible to export their measurement data. The IMAQ Vision functions [IMAQ Get Meter](#) and [IMAQ Read Meter](#) make it possible to convert the image of an analog instrument, using the information of the needle position, into a measurement value that can be further processed by a PC.

[Figure 5.59](#) shows the (slightly modified) meter VI, which is part of the IMAQ Vision examples.

Figure 5.59. Reading an Analog Needle Instrument

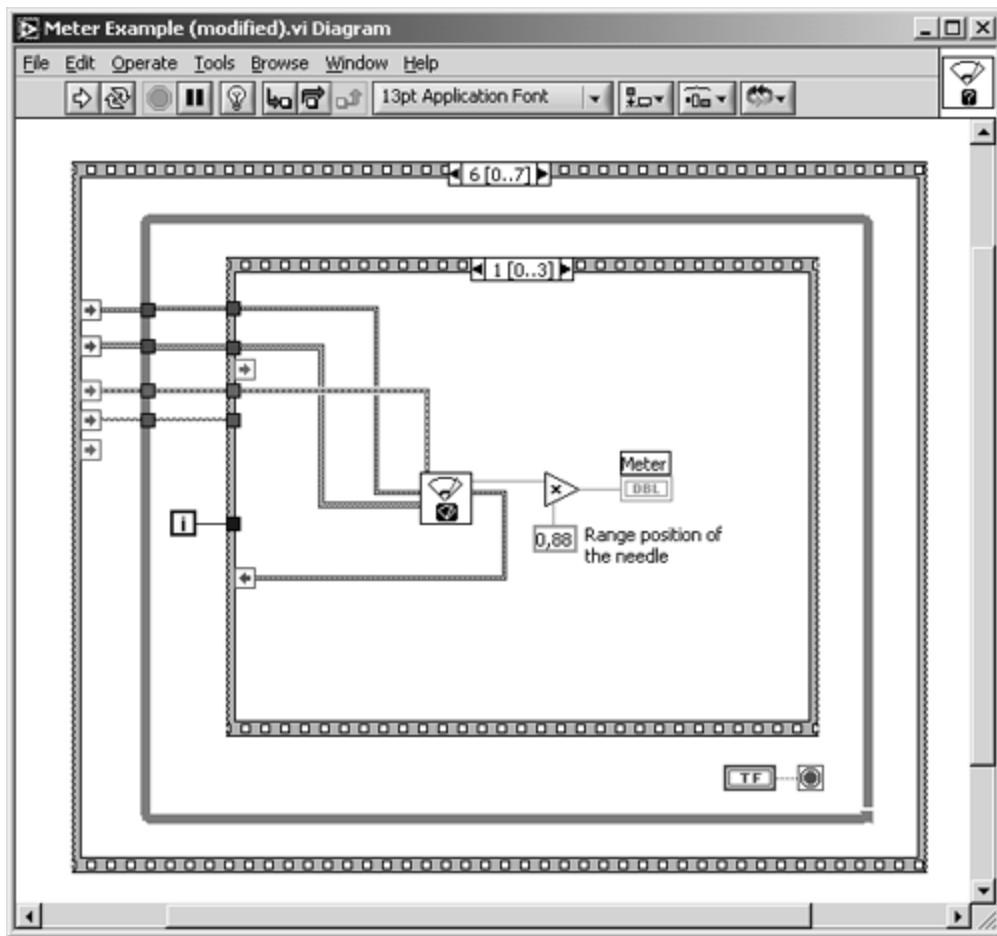


After the program is started, you have to run through the following procedure:

- One special image of the instrument with the needle in the minimum position is displayed. Draw a line on the image of the needle starting at the top and press the Line OK button.
- Another image with the needle in the maximum position is displayed. Repeat the previous step with this image.

The program is now ready to read various meter values if the scale range of the meter (the difference between maximum and minimum position) is provided. In the meter example, this value is hard-coded (see [Figure 5.60](#)).

Figure 5.60. Part of the Analog Meter Reading Diagram



A part of the diagram of the meter example, shown in [Figure 5.60](#), illustrates the use of the function **IMAQ Read Meter**. You can, of course, also use your own images with this example; just put them in the subdirectory Meter and modify the value range in the diagram.

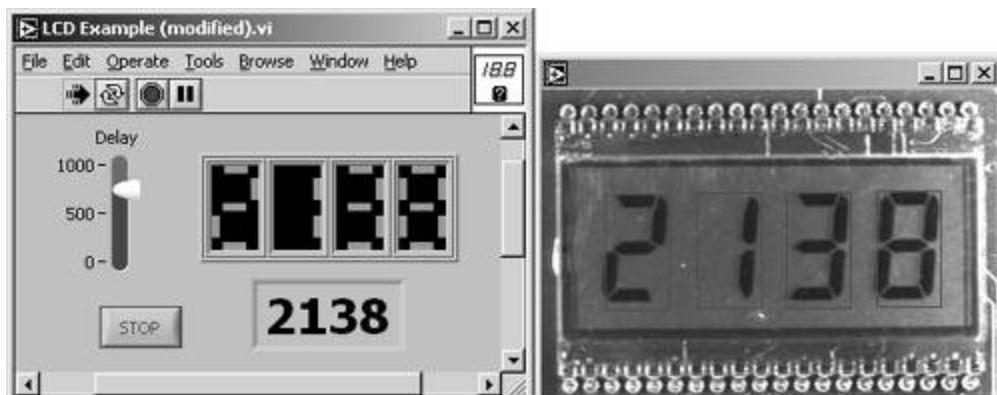
Digital Instrument Displays

Reading digital displays^[3] requires different algorithms. An example is also part of one of the application papers (Moving Camera; page 289) [\[40\]](#).

^[3] These are also known as liquid crystal displays (LCDs).

Again, I modified an IMAQ Vision example (see the front panel in [Figure 5.61](#)). The procedure the user has to go through is quite simple:

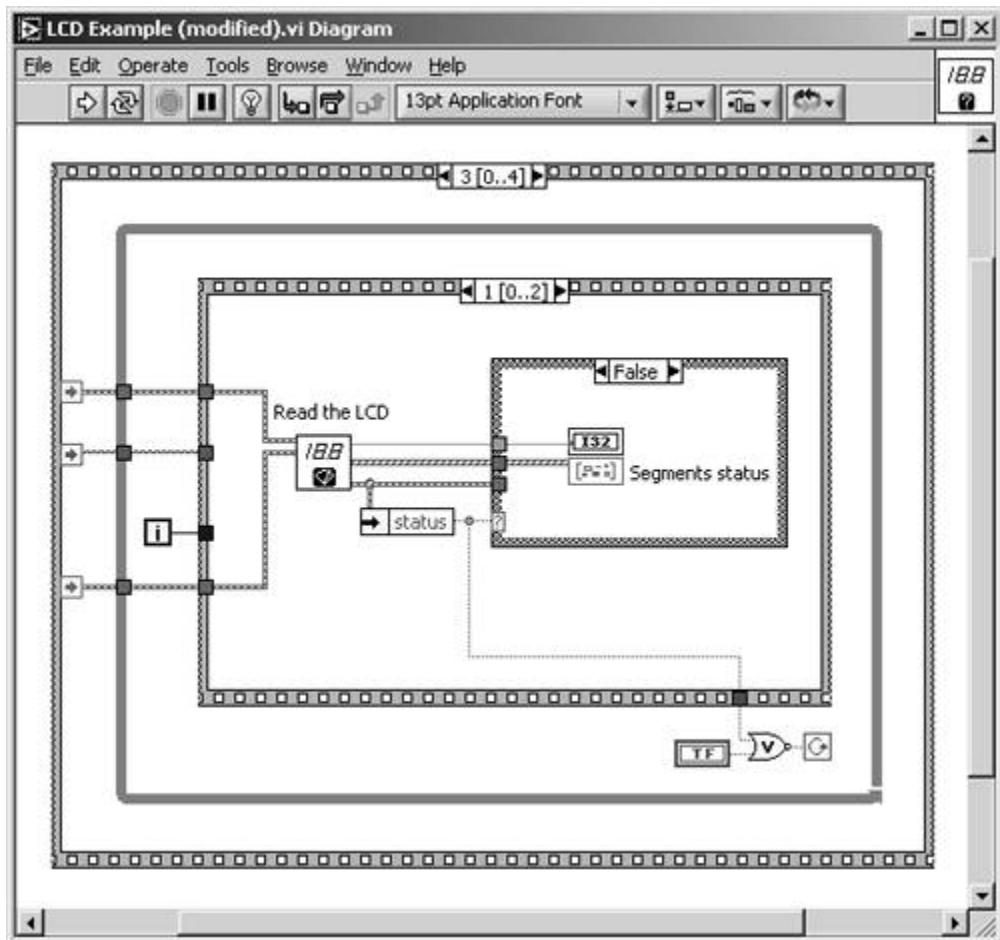
Figure 5.61. Reading a Digital LCD Instrument



- After the program is started, a single image of an LCD is displayed. Draw a rectangle around all four digits of the LCD and click the OK button.
- The program now reads all available LCD values, which are stored as images in the subdirectory LCD. Again, you can run the VI with your own images.

The example VI uses the functions `IMAQ Get LCD ROI` and `IMAQ Read LCD`, similarly to the meter example described in the previous section. See a part of the example diagram in [Figure 5.62](#).

Figure 5.62. Part of the Digital LCD Reading Diagram



The VI works also with a light emitting diodes (LED) display, which means light segments on a dark background in opposition to the dark segments on a light background of an LCD. To distinguish between these two types of displays, a boolean constant (LCD/LED) is used.

Note also the insensitivity of both meter reading programs to brightness changes of the image. To help you visualize this, the example images show various lighting and intensity situations.

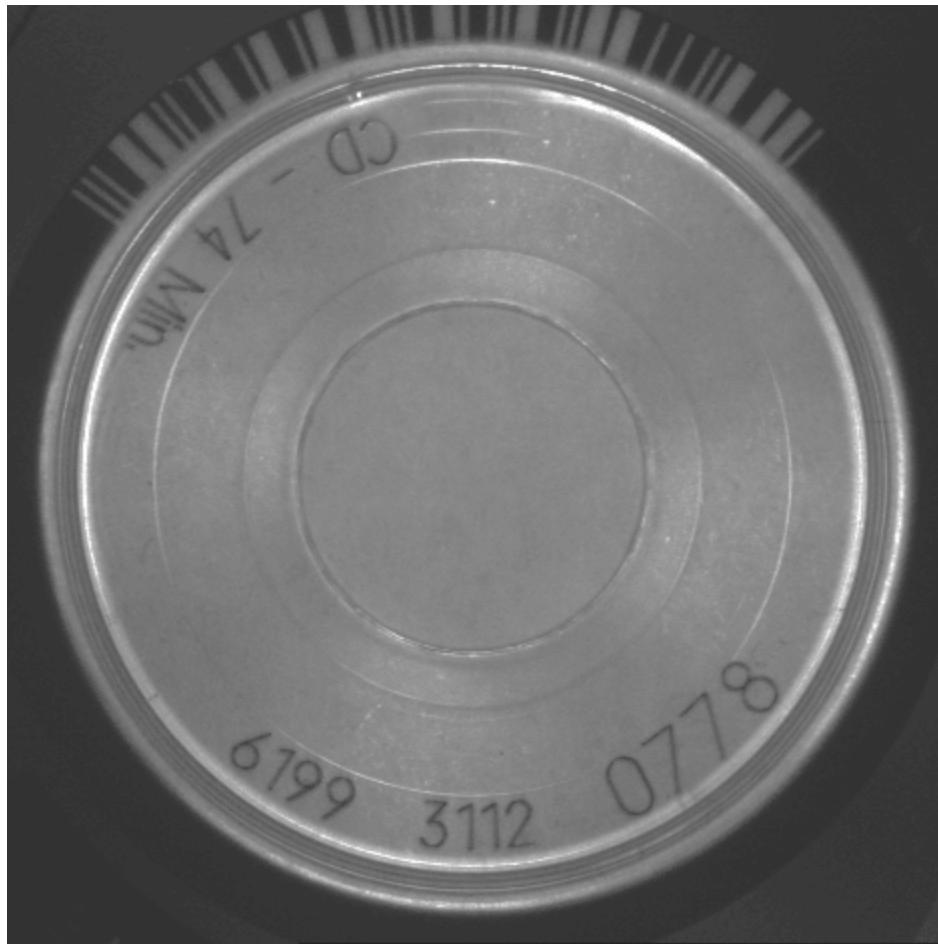
Character Recognition

IMAQ Vision also provides some tools for the machine recognition of characters, similar to the digital instrument example; in the basic version of the IMAQ Vision toolkit, only the functions for bar code reading are implemented. The reading of text characters, or optical character recognition (OCR), can only be implemented by addition of the IMAQ Vision OCR toolkit.

Text Reading (OCR)

OCR is a method that converts images containing text areas into computer editable text files. The IMAQ Vision OCR toolkit can read text in capital and printed letters. An additional group of functions can be used to "unwrap" characters that are not located in a straight line, for example, the number on a compact disc (see [Figure 5.63](#)), before the OCR process begins.

Figure 5.63. Characters to "Unwrap" on a Compact Disc



Bar Code Reading

As mentioned before, bar code reading is included in the basic IMAQ Vision functions. An example including some bar code images is part of the IMAQ Vision Examples set and is easy to modify for our purposes.

In the examples section on the attached CD-ROM, I provide four special images in two different bar code languages:

- [barcode0.png](#) and [barcode1.png](#) are the codes from the back side of the IMAQ Vision manuals ([4] and [5]). These codes are a 3 of 9 type (or Code 39); [Figure 5.64](#) shows an example.
- [barcode2.png](#) is an ISBN code from the book of E. R. Davies: *Machine Vision* [13] ([Figure 5.65](#)).
- [barcode3.png](#) is another ISBN code from the book *Computed Tomography* by W. A. Kalender [20].

Figure 5.64. Bar Code Reading Example (Code 39 Setting)

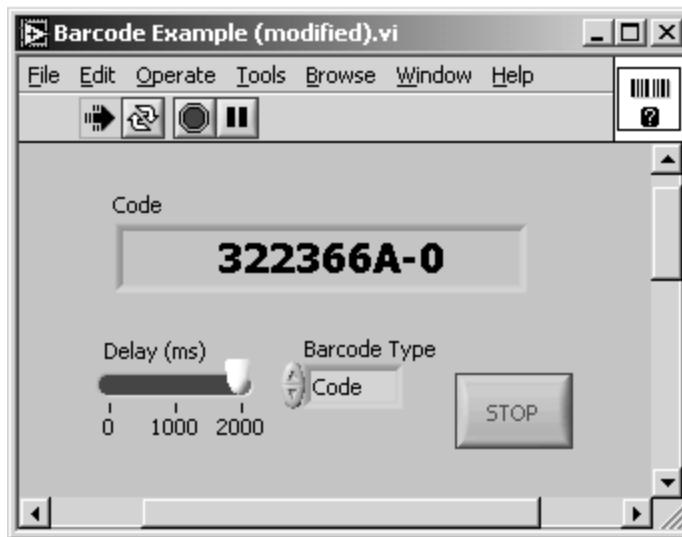


Figure 5.65. Bar Code Reading Example (EAN 13 Setting)



The function that performs the bar code reading is [IMAQ Read Barcode](#). In general, the program works with the following bar code languages:

- *UPC-A* (Universal Product Code-A) is the most widely used code for product identification in the United States. The code has a fixed length of 12 characters, with the twelfth digit being a check value calculated from the other 11.
- *MSI* or MSI-Plessey is a code containing check digits. It uses special symbols for start and stop characters.
- *Interleaved 2 of 5*: This is a high-density code of variable length for the coding of an even number of digits. Odd-positioned data is encoded in the bars, even-positioned data in the spaces.
- *EAN 13* (Europäische Artikel-Nummerierung) is mainly used in Europe instead of UPC-A. It consists of 13 digits instead of 12 (see [Figure 5.65](#)).
- *EAN 8*: EAN code with 8 digits.
- *Code 128* allows the coding of the entire 128 ASCII (American Standard Code of Information Interchange) character set. Code 128 is self-checking and contains features that improve scanner reading.
- *Code 3 of 9 (Code 39)* is a common code that can also encode alphanumeric characters. The start and the stop character is always an asterisk (*; not shown in the display). National Instruments uses this code for its manuals (see [Figure 5.64](#)).
- *Code 9 of 3 (Code 93)* provides higher density and higher security than does Code 39.
- *Codabar* is a self-checking language that can encode the digits 0-9 and 6 special characters.

To calculate the ISBN bar code of books, skip the last number of the ISBN; for example, the ISBN 0-12-206092-X is turned into the number 012206092. After that, add 978 to the beginning. These two steps lead to the 12-digit number 978012206092 (compare with [Figure 5.65](#)).

The procedure of the VI is very similar to that of the LCD example: just draw an ROI in the first image (as long and as narrow as possible) and the program is ready to convert bar codes according to the type setting control.

[[Team LiB](#)]

 PREVIOUS  NEXT 

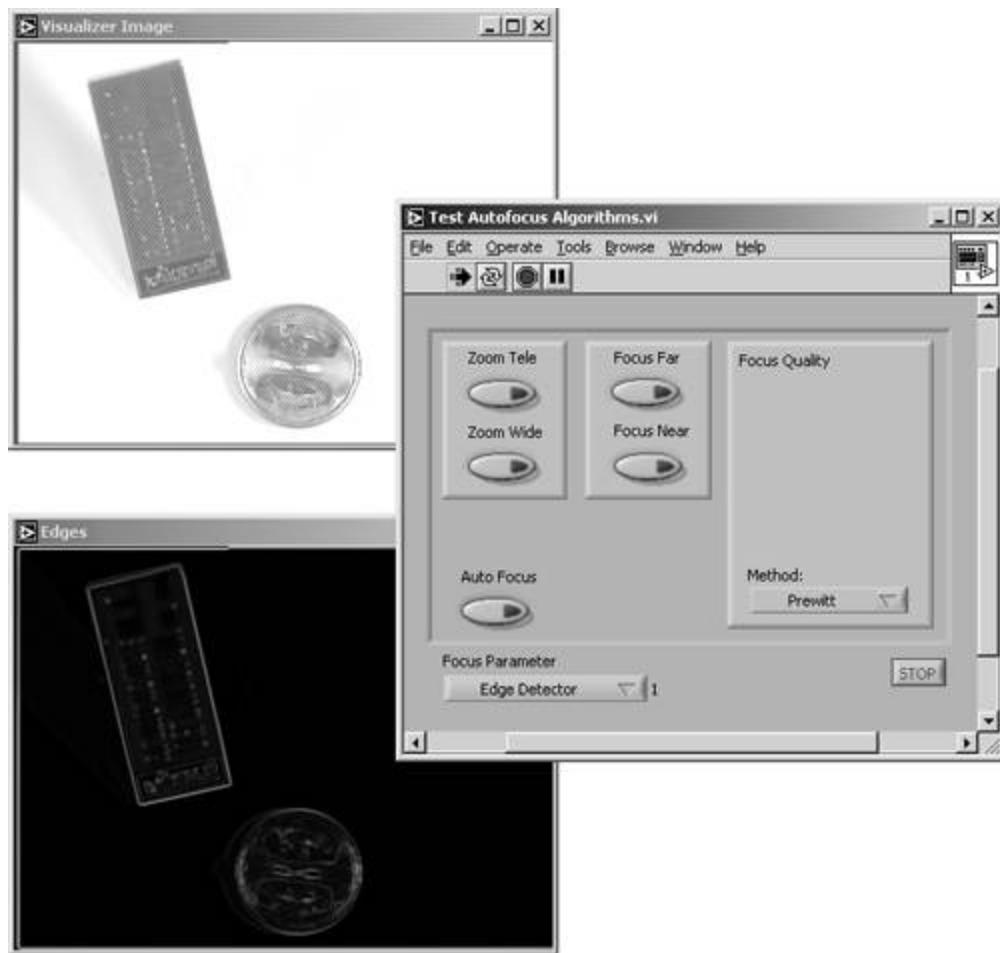
Image Focus Quality

The idea for this example section came to me when I visited NIWeek 2001 and listened to a presentation called "Emerging Technologies in Vision," which contained a section about autofocusing algorithms; with the following example we can try them ourselves.

Autofocusing (the adjustment of camera focus to obtain optimal image focus) can be divided into two more or less independent tasks:

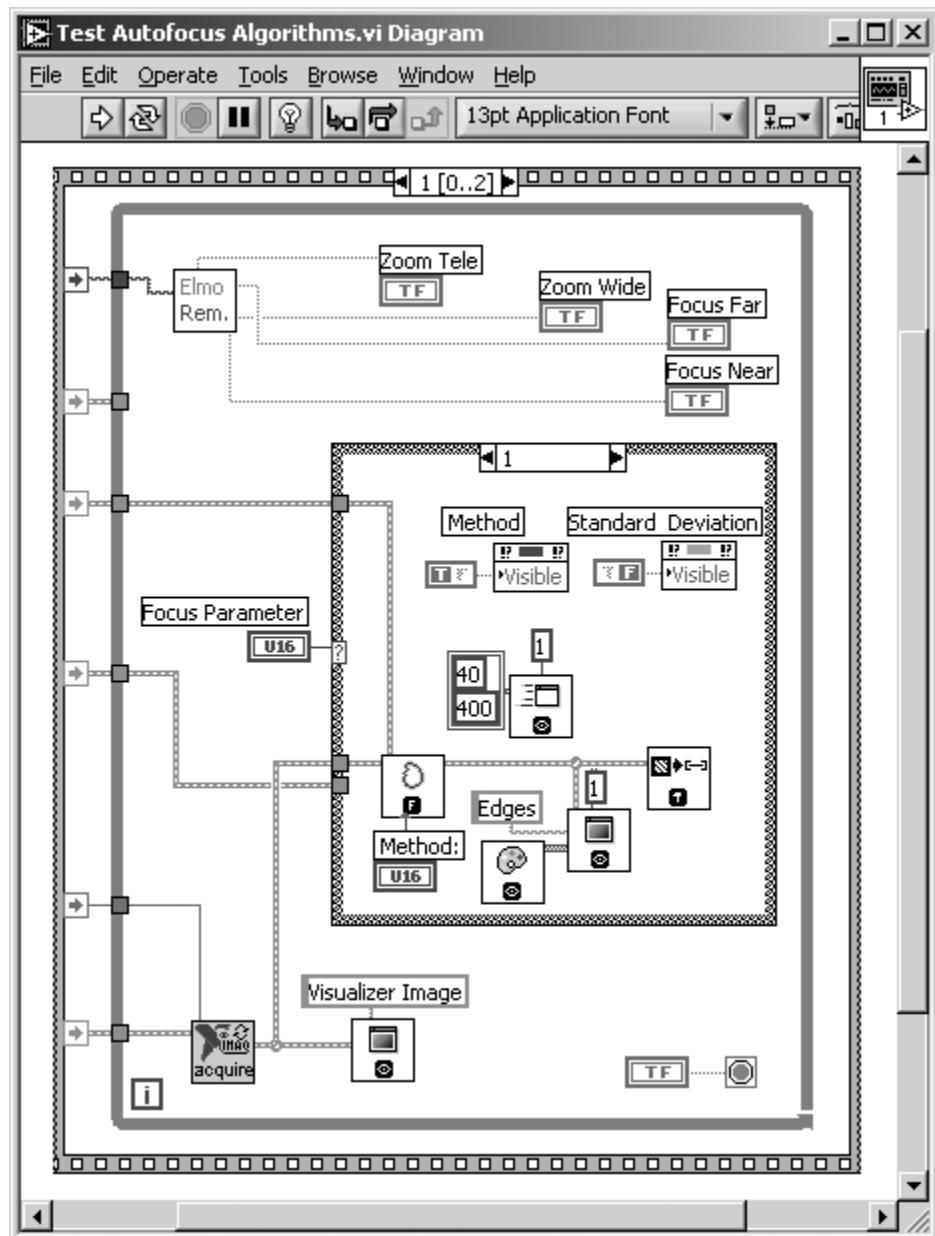
- determine the optimum focus by analyzing image quality (for example, [Figure 5.66](#));

Figure 5.66. Focus Quality Rating with Edge Detection



- adjust the camera focus setting. We do not talk about this part, but if you look at one of the diagrams (for example, [Figure 5.67](#)), you will notice zoom and focus setting controls, which are inputs of a subVI called Elmo Remote. You can use your own routines instead.

Figure 5.67. Focus Quality Diagram (Edge Detection)

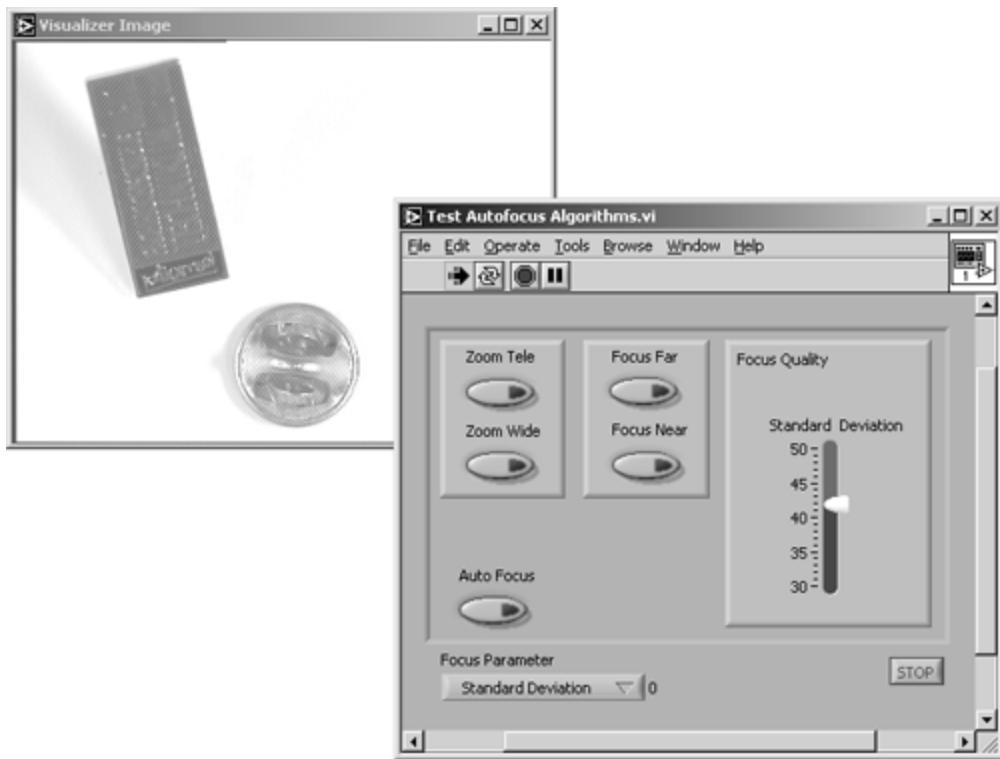


Back to the first task: *focus quality* is defined by sharp differences between object edges and the background. Therefore, the simplest method for quantifying the image focus is obviously a classic edge detection, for example, with a Sobel or a Prewitt filter kernel. [Figure 5.66](#) shows a result, and [Figure 5.67](#) shows the corresponding diagram. It turns out that the classic edge detection method is quite accurate and fast. The sum of all pixel values of the filtered image can be used to specify a value for the focus quality; it gets a maximum at optimal focus.

Note that the example program is designed for the testing of three (or more) methods; a case structure selects the method and enables active controls or disables elements that are not required by the current method. All cases are prepared for further programming.

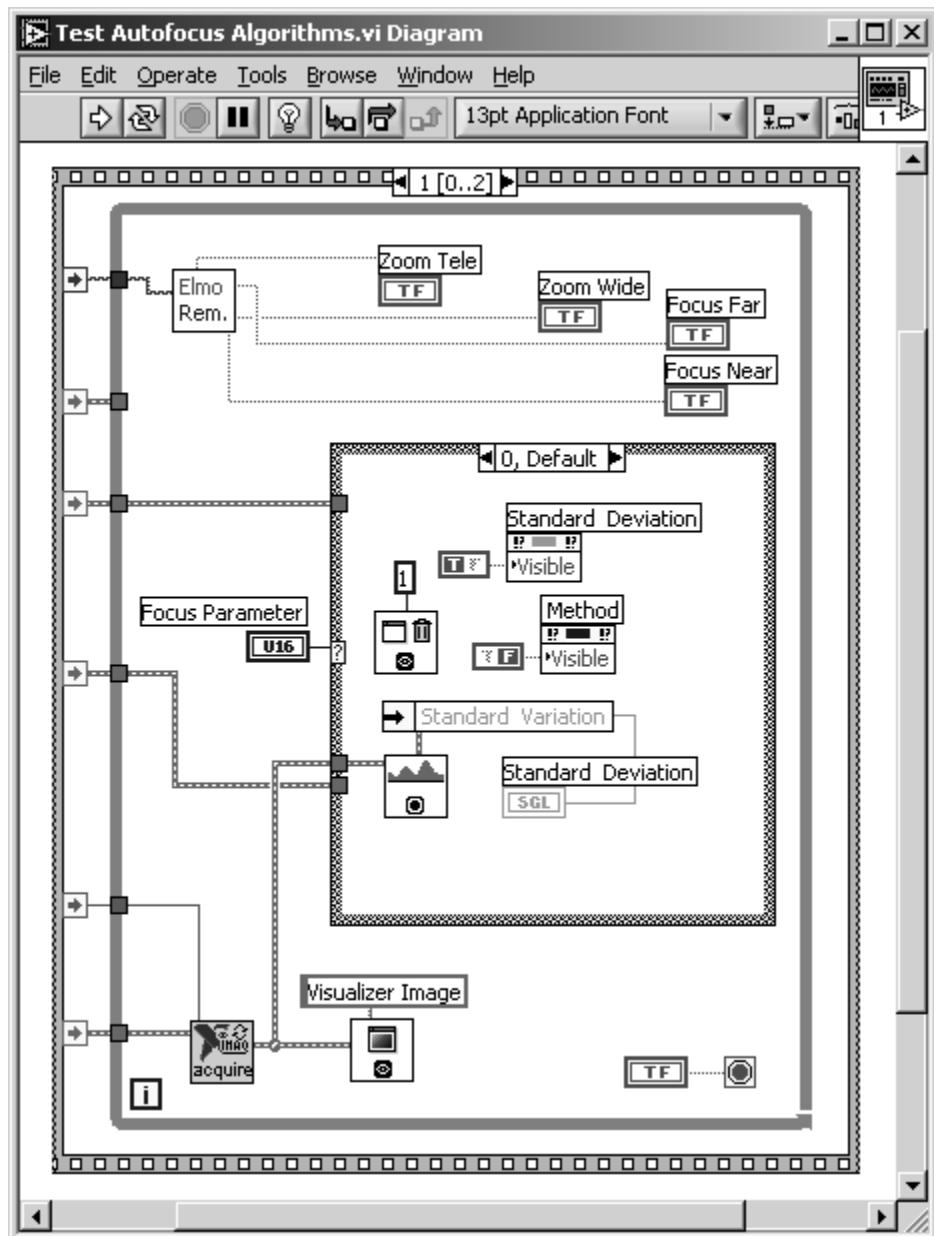
[Figure 5.68](#) shows the result of another possible method. Here, the standard deviation of the image histogram is used as an indicator of focus quality. At optimum focus, the value of the standard deviation gets a maximum.

Figure 5.68. Focus Quality Rating with Histogram Analysis



The corresponding block diagram can be seen in [Figure 5.69](#). The function `IMAQ Histogram` is used to calculate the standard deviation of the image's pixel values. Although this method is very fast, it is less accurate than the edge detector.

Figure 5.69. Focus Quality Diagram (Histogram Analysis)



The third method we discuss here is based on the frequency domain of the image. Remember that sharp edges in images are represented by medium to high frequencies in the frequency domain image. [Figure 5.70](#) shows the calculated FFT (fast Fourier transform) of our test image.

You already know how to build the diagram shown in [Figure 5.71](#): Simply use the function [IMAQ FFT](#) to display the fast Fourier transform; you have to add functions to quantify the amount of high-frequency components yourself.

Another approach for the frequency domain could be the use of wavelets, as discussed in [Chapter 3](#). All frequency domain methods are very accurate, but also very time consuming (wavelet methods are a little faster than FFT methods). [Table 5.1](#) compares the discussed methods; obviously each of them has its advantages and drawbacks.

Figure 5.70. Focus Quality Rating with FFT

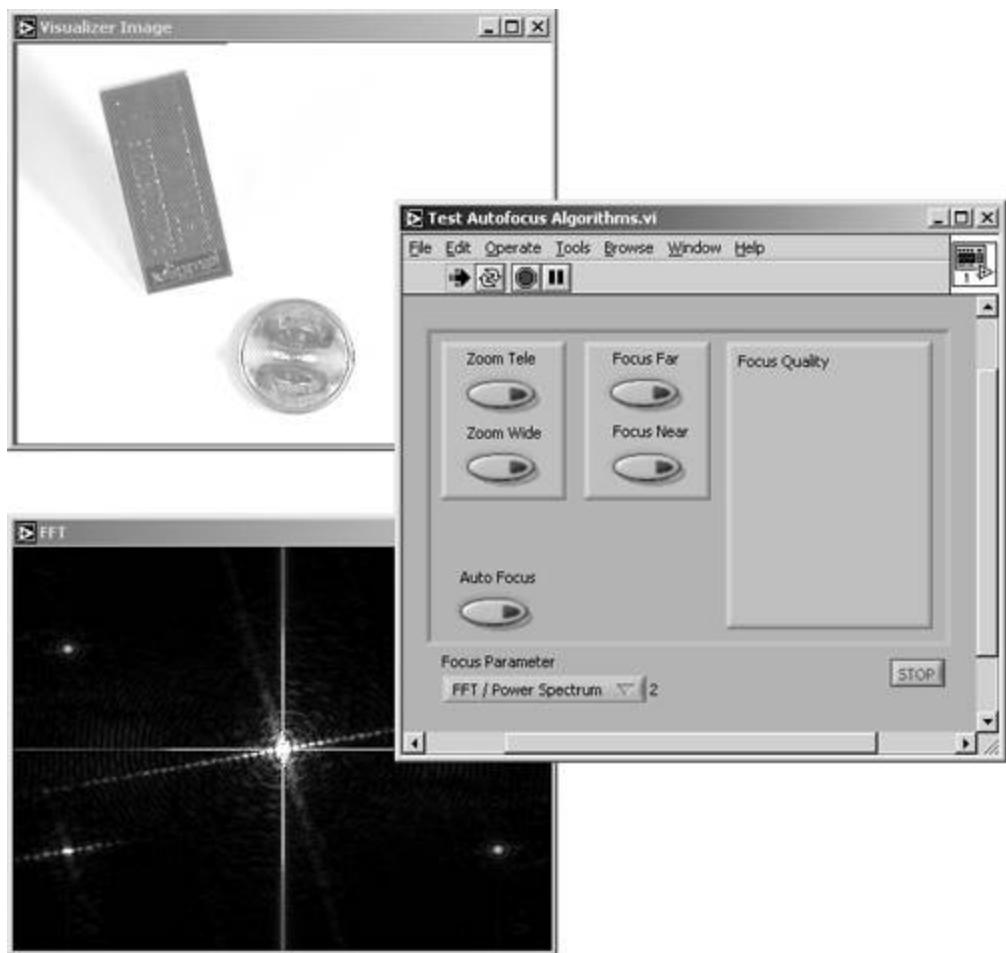


Figure 5.71. Focus Quality Diagram (FFT)

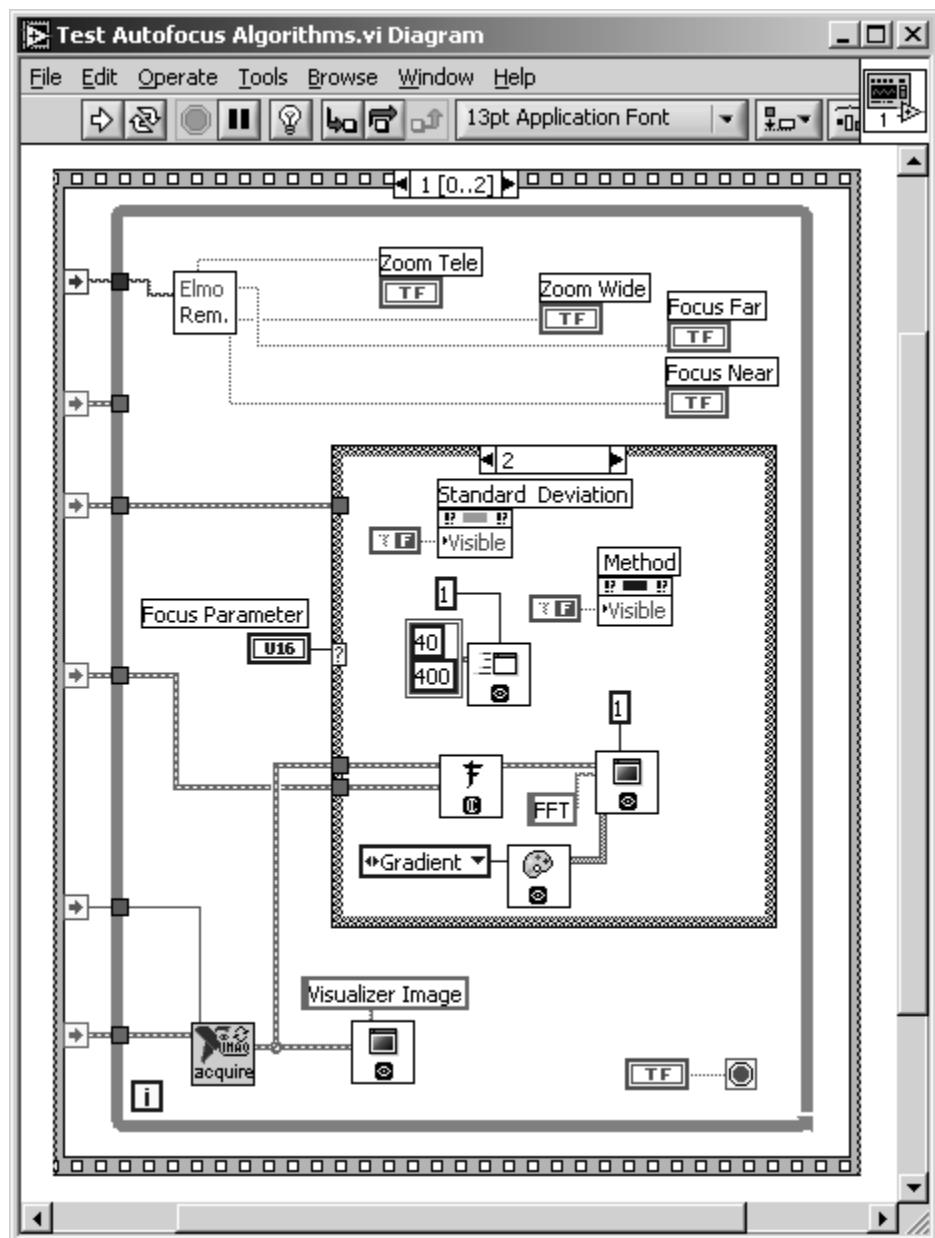


Table 5.1. Comparison of Focus Quality Rating Methods

Method	Accuracy	Speed
<i>Classic edge detector:</i>	+	+
<i>Statistical (standard deviation):</i>	o	++
<i>Frequency domain (FFT):</i>	++	o

Application Examples

The following examples show some applications my colleagues and I developed over the past few years. They were papers we presented at various NI conferences (mostly NIWeek, but also VIP conferences in Europe); that is why you will find some redundancies, especially regarding FireWire. These papers are reproduced by permission of National Instruments.

You can find related software on the attached CD; when certain hardware is needed, parts of the application or demo samples of the software are available.

Moving Camera Supports EMC Test Automation

by Thomas KLINGER, Christian MADRITSCH, and Hermann STERNER (Paper presented at NIWeek 1999) [\[40\]](#)

Category

- Industrial Automation

Products Used

- LabVIEW Version 5.0
- IMAQ Vision for LabVIEW

The Challenge

EMC testing is usually a long and intensive procedure. The test object has to be visually checked and supervised by a test engineer, who has to record the time when the test object fails to operate.

The Solution

A Moving Camera, controlled by a LabVIEW application, using IEEE 1394 (FireWire) technology for video transmission and control signals, is able to detect test object failures by means of image processing.

Introduction

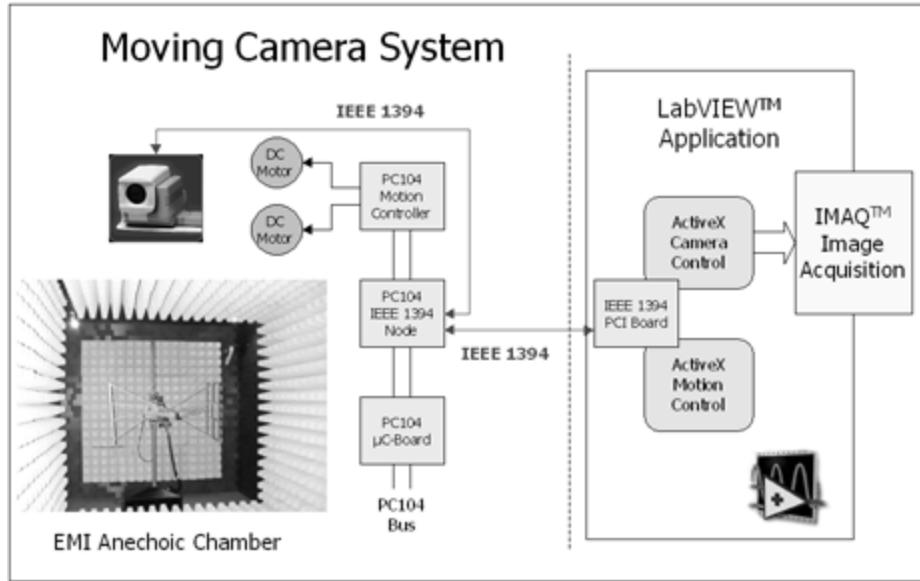
CTI (Carinthia Tech Institute, or Technikum Kärnten) is an educational institute with several research projects supported by the Austrian government. Two of the main topics of CTI are serial bus systems (with focus on the IEEE 1394 technology) and EMC. The Moving Camera project gave us the opportunity to combine these two topics.

Normally, a camera is installed inside an EMI anechoic chamber to monitor the test object

during the test period. Current solutions use a video monitor in combination with a VCR, but by using a digital video camera with IEEE 1394 bus, the video can be displayed on a PC.

The Moving Camera system described in this paper allows not only setting the camera properties (e.g. brightness, focus, zoom), but also tilt and pan adjustment of the camera.

Figure 5.72. Block Diagram of the Moving Camera System



IEEE 1394 (FireWire)

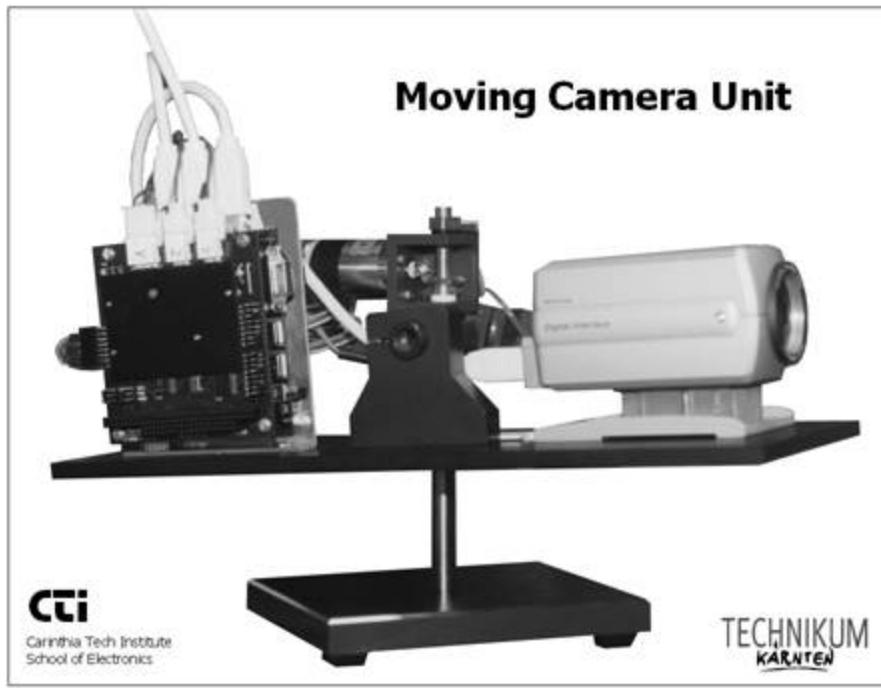
IEEE 1394 is a serial bus system invented by Apple in 1985 (at this time known as ADB; Apple Desktop Bus) with the goal to have a standard for a high-speed serial bus that is able to replace all existing interfaces on PC or multimedia environments. In 1995 the bus was standardized under the name IEEE 1394-1995.

The primary features of FireWire are:

- Transfer rates of 100 Mb/s, 200 Mb/s, and 400 Mb/s (up to 3.2 Gb/s planned).
- Support for isochronous applications (guaranteed bandwidth) as well as asynchronous applications (guaranteed delivery).
- Hot plugging: Devices can be attached to or removed from the bus dynamically.
- Cable power: Power available from the bus can be either sourced or sunked by a given node.
- Daisy-chaining: The serial bus can be extended by connecting new serial devices to ports provided by serial bus nodes.

Currently, FireWire is becoming common for high-end multimedia applications, such as digital camcorders or VCRs. On the other hand, the features mentioned above enable FireWire to establish itself as a standard for industrial automation applications.

Figure 5.73. Prototype of the Moving Camera Unit



The Moving Camera System

[Figure 5.72](#) shows a block diagram of the whole system. The left hand part of the figure represents the Moving Camera unit, which is located inside the EMI chamber and connected via one single serial IEEE 1394 cable with a PC equipped with an IEEE 1394 PCI interface board. The IEEE 1394 cable provides the digital video signal as well as the motor control signals and the power supply for both the camera and the DC motors.

The hardware is controlled by two ActiveX controls; one for displaying the video data on the PC screen and the other for positioning the digital camera using the motor controller and the DC motors.

Moving Camera Hardware

The Moving Camera unit (inside the anechoic chamber) is shown in [Figure 5.73](#) and consists of

- a digital IEEE 1394 video camera (Sony) with a resolution of 640 x 480 pixels;
- a PC104 microcontroller board (386SX microprocessor, 24 MHz clock, 1 MB DRAM);
- a PC104 IEEE 1394 node for asynchronous transfers (200 Mb/s, developed by CTI);
- a PC104 Motion Controller board (developed by CTI) driving two DC motors for horizontal and vertical movement;
- aluminum ground plate and DC motor support with gear units (prototype in [Figure 5.73](#) shown without housing).

The PC104 bus allows for a modular design of the system. For further upgrading, it is possible to add other components, e.g., ADC-boards.

Figure 5.74. Screenshot of a Typical Image Processing Application



Moving Camera Software

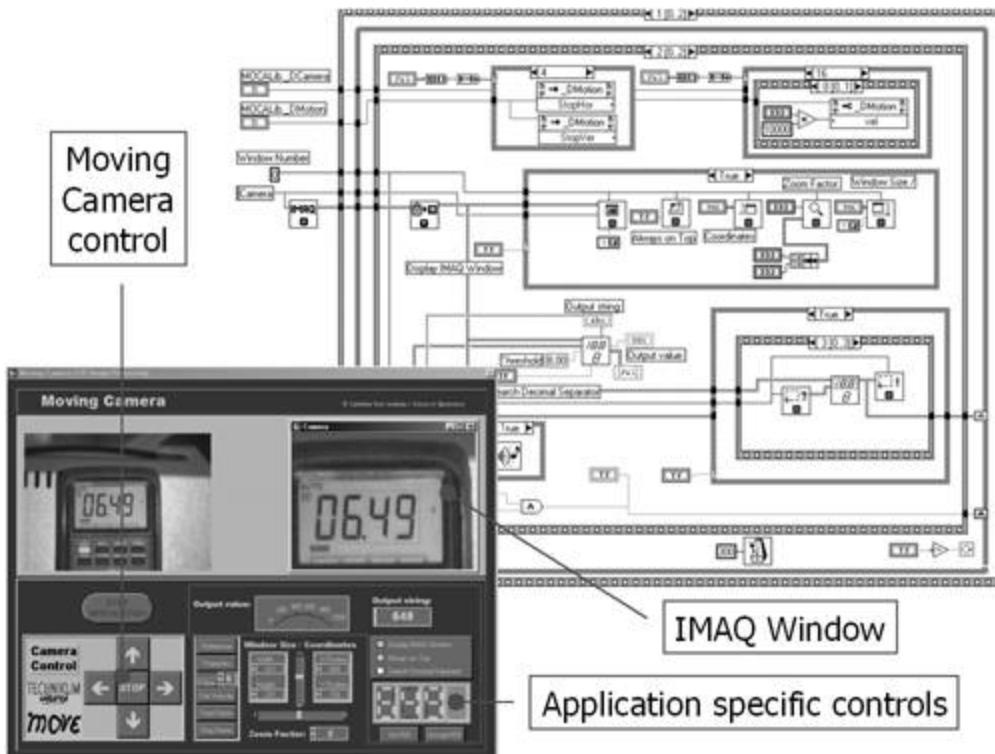
The Moving Camera software structure is shown on the right hand side of [Figure 5.72](#). The software base is two ActiveX controls, which are layered above the driver for the IEEE 1394 PCI card.

The first ActiveX, called Camera Control, provides the functionality to control the camera properties and to display the video stream. The second is called Motion Control and provides the functionality to control the movement unit. Both controls are embedded within a LabVIEW application. For example, the Motion Control ActiveX exposes in its COM interface the methods MoveRight, MoveLeft, MoveUp, MoveDown. These methods can be invoked by pressing a control element of the LabVIEW GUI, using LabVIEW's ActiveX capabilities.

[Figure 5.74](#) shows the screenshot of a typical LabVIEW user interface. The upper part of the screen displays the live video on the left and the IMAQ window used for further image processing on the right. The lower half consists of the control elements for the moving camera, position and zoom elements for the IMAQ window, and application specific controls (shown in [Figure 5.75](#)).

One of our sample applications detects the change of brightness in a previously specified region. For example, this feature allows for the detection of LED failure ([Figure 5.74](#)). In [Figure 5.75](#) the value of a digital display is being read and certain values can be detected.

Figure 5.75. LabVIEW Diagram and Front Panel Details



Conclusion

Due to the ActiveX capability of LabVIEW, it is very easy to enhance the functionality of LabVIEW with custom-made functions and as a result, it is possible to include digital video functions, DC motor control, IEEE 1394 bus handling, and image processing in a single application.

Object Detection and Counting in Public Places

by Thomas KLINGER, Christian MADRITSCH, and Hermann STERNER

(Paper presented at NIWeek 2000) [[41](#)]

(similar paper presented at VIP 2000, Munich)

Category

- R&D

Products Used

- LabVIEW Version 5.1
- IMAQ Vision for LabVIEW
- AT-485/2 RS-485 Serial Interface
- 6B Series Distributed Signal Conditioning and I/O Modules

The Challenge

The new fountain in the Villach city hall square should be a "situative reflection" of the urban situation of the square and the surrounding area; the presence and the movement of people in the square should result in variations of the fountain water height.

The Solution

Using two digital video cameras, image processing algorithms, and a LabVIEW control program, it is possible to determine the number of persons as well as their speed vectors and to control the fountain height by an analog output voltage.

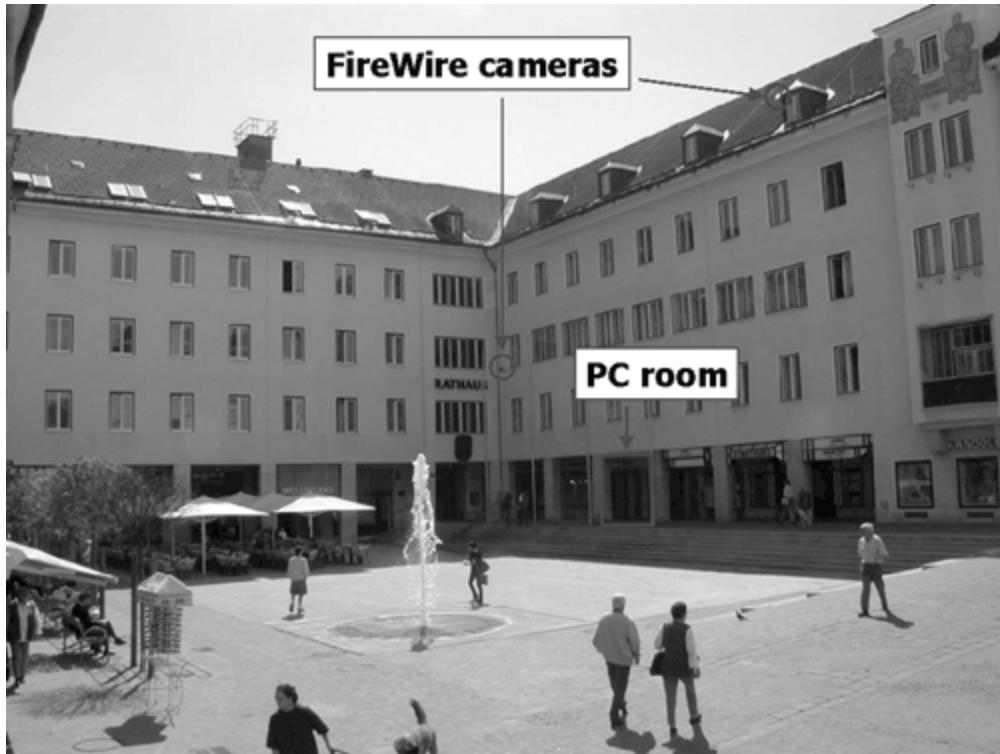
Introduction

The definite detection of objects (here, people) in public places is quite challenging. The processed image is affected by the variation of shadows, vehicles, and other large objects, which normally cause a significantly higher detection rate than the objects of interests. This paper shows how specific IMAQ Vision image processing algorithms provide the separation of the desired information. The two digital video cameras are connected to a PC using the serial bus system IEEE 1394 via plastic optical fiber. Due to the relatively large distance between the PC and the fountain control, 6B modules were used for the voltage output.

Fountain Control Hardware

[Figure 5.76](#) shows the Villach city hall square with the new interactive fountain in the front. The position of the two FireWire cameras is indicated by the circles; one of them has its focus on the surrounding of the fountain, the other one is able to watch the activities on the whole square (see also [Figure 5.79](#)).

Figure 5.76. Villach City Hall Square with Interactive Fountain



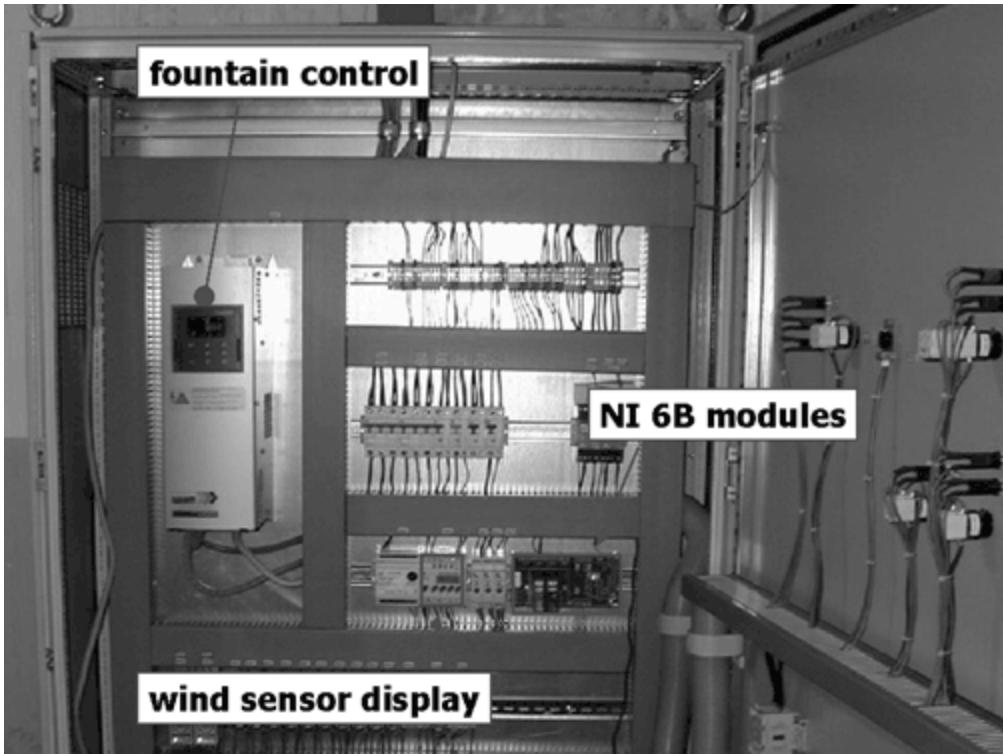
The hardware used for the interactive fountain control consists of:

- the two 1394 digital cameras (Sony DFW-VL500), including cables and repeaters (wire and plastic optical fibre);
- a PC (Pentium III/550) equipped with a 1394 FireBoard 400 (Unibrain) and an AT-485/2 RS-485 serial interface card (National Instruments);
- a 6B 4-channel backplane with RS-485 interface for the two required 6B modules:
 - a 6B21 current output module provides the control output signal for the fountain;
 - a 6B12 voltage and current input module. This module is used as input for a wind sensor; in case of strong wind, the software reduces the fountain water height.

Figure 5.77 shows the fountain control box with the control unit itself on the left side (installed by a third company). The current output of the 6B21 (0 ... 20 mA) is converted into a 0 ... 10 V voltage by means of a 500 μ m resistor and connected to the control unit. Also shown in Figure 5.77 is the wind sensor connected to the 6B12 module.

The camera pictures are imported into IMAQ Vision by a 1394 ActiveX control (version 1.4) written by CTI. In case the PC is equipped with 1394 functionality, no further hardware is required for the capturing of the live images.

Figure 5.77. Fountain Control and NI 6B Modules



Object Detection Algorithms

The software part of the project solution has to fulfill the following requirements:

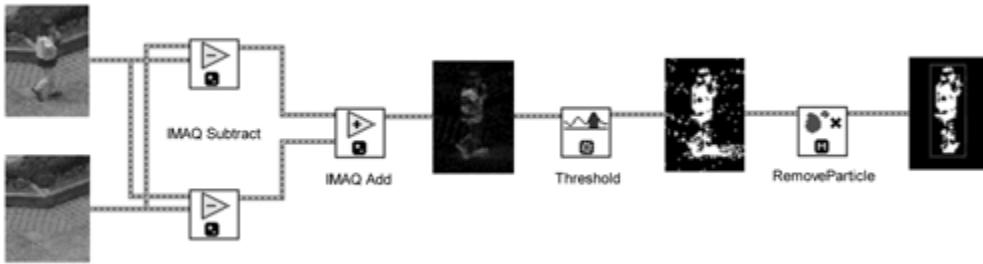
- supervision of the Villach city hall square (two focus areas);
- detection and counting of "objects" (people) on the square;
- calculation of their speed vectors;
- electrical output of values representing the number of people on the square (absolute fountain water height) and their movement (frequency of fountain water height variation).

[Figure 5.78](#) shows the principle of the object detection algorithm. The actual picture (top-left side of [Figure 5.78](#)) is subtracted from a so called "idle picture" using the function [IMAQ Subtract](#) twice, in order to get significant difference values for light and dark areas of the image. The two differences are added to one single picture using the [IMAQ Add](#) function.

The idle picture, representing the empty city hall square, is generated by calculating the mean pixel value out of a number (here, 8) of actual pictures, taken in distances from some seconds up to a few minutes. This algorithm guarantees a dynamic generation of the idle picture, resulting in insensitivity to moving shadows, parking vehicles, and other large objects, which otherwise would cause a significantly higher detection rate than the objects of interests.

Further image processing is done by using the functions [IMAQ Threshold](#) (creates a binary image) and [IMAQ RemoveParticle](#) (keeps particles resistant to a specified number of 3×3 erosions). The remaining particles are marked in the original picture (see also [Figure 5.79](#)).

Figure 5.78. Principle of the Object Detection Algorithm



Another possibility is the use of the function `IMAQ_BCGLookup` instead of `IMAQ_Threshold` and further processing with the appropriate `IMAQ_GrayMorphology` functions. This will keep particles more consistent because the image is not converted to binary values.

[Figure 5.79](#) shows the user interface of the fountain control software. Two VIs are running; one of them (left side of [Figure 5.79](#)) displays the images, executes the detection algorithms, and calculates the values for the number of objects and for their speed, respectively. The other one (right side of [Figure 5.79](#)) determines the fountain height value (represented by the tank indicator) and controls the 6B modules.

Figure 5.79. User Interface of the Fountain Control Software



Extracting GIS Data for Telecom Networks from City Maps

by Thomas KLINGER, Peter BACHHIESL, Gernot PAULUS, Joachim WERNER, and Herbert STOEGNER

(Paper presented at NIWeek 2002) [\[45\]](#)

(similar paper presented at VIP 2003, Munich)

Category

- Communication

Products Used

- LabVIEW Version 6i
- IMAQ Vision for LabVIEW Version 6.0.1

The Challenge

For the use of a previously developed tool (NETQUEST), which represents a digital workflow for the computation of cost-optimized layouts for fiber optic access networks, very little geographic data is present. A tool was needed which extracts rough geodata out of city maps.

The Solution

We developed an IMAQ Vision program that is able to read in city maps (from paper scans or the World Wide Web), to extract relevant classes (streets, buildings, or similar), and to transfer the result into an output format which can be used by NETQUEST.

Motivation

During the last two years, European network-carriers have invested 7.5 billion Euro in the implementation of fiber optic networks—mainly in the expansion of the core—and the distribution net domain (backbones, city backbones, and metropolitan area networks).

Investigations have shown that about 95 percent of the total costs for the implementation of a three-layer fiber optic network may be expected for the areawide realization of the last mile (access networks). Therefore, the expansion of the access net domain represents the bottleneck of modern network architectures. In order to achieve a return on investment, carriers will be forced to link access net nodes, like corporate clients, private customers, or communication nodes for modern mobile services (e.g., UMTS) to their city backbones.

In this context we have focused on the cost-intensive underground work for the implementation of fiber optic cables under real-world conditions. The two stage planning tool NETQUEST represents a full digital workflow for the computation of cost-optimized layouts for fiber optic access networks. This tool combines real-world geo-information data (land-use classes and the according specific implementation costs per meter) with sophisticated methods from mathematical optimization.

NETQUEST is based on digital geodata representing land-use classes (i.e., street network, parcels, buildings, rivers, etc.) which are relevant for network planning. NETQUEST uses this geodata for an appropriate network optimization in two representations: as vector and image format as well.

High-resolution digital geodata is not always available. For any large network domain, high resolution data capture using ground surveying techniques is extremely expensive. On the other hand, raster maps like digital city guide maps are available at comparable low cost and can be used for strategic planning purposes on a general level.

Using the image processing functionality of LabVIEW and IMAQ Vision, we have developed an efficient workflow to extract land-use classes relevant for network planning from low-cost image data sources. These data sets provide an important, cost-efficient input for a GIS in order to assign real-world construction costs (\$/m) to each of the land classes. A case study from Leipzig, Germany is presented.

The Layer Extraction Algorithm

In our algorithm, each land-use class is described by a layer represented by a binary bitmap file or a corresponding text file, respectively. The algorithm itself contains the following steps:

1. Reduce the map image to a single color plane (hue plane; IMAQ function `IMAQ ExtractSingleColorPlane`);
2. Extract a single color using threshold functions (`IMAQ Threshold`, `IMAQ Cast Image`);
3. Remove small objects, which may be unwanted results of thresholding (IMAQ function `IMAQ RemoveParticle`);
4. Close resulting areas, especially holes resulting from map text (IMAQ function `IMAQ Morphology`/Close);
5. (optional) Extract object borders (IMAQ function `IMAQ Morphology`/ Gradient in).

The LabVIEW VI is intended for use in presentations, too; this is why every step of the algorithm can be executed separately. In step 1 it is absolutely necessary to use the hue plane of the image and not the intensity or brightness information. The hue information can be calculated from the RGB values using the following formula:

$$H = \cos^{-1} \left[\frac{\frac{1}{2}[(R - G) + (R - B)]}{\sqrt{(R - G)^2 + (R - B)(G - B)}} \right]$$

and is represented by a color angle starting from the red axis.

The threshold function (step 2) can be executed in two different ways; if the color scheme of the city map is always the same, fixed values can be used for the extraction of land-use classes. If the city map uses different colors for land classes, it is possible to define the upper and lower threshold values manually, using cursors in a histogram display. The threshold step of the algorithm is exemplarily shown in [Figure 5.80](#).

Postprocessing Using NETQUEST

In an additional sixth step, the layer is saved as a bitmap or text file. The (previously developed) tool NETQUEST combines all layers, assigning the respective costs per meter for the construction of a fiber optic network. In this case, the classes shown in [Table 5.2](#) were used.

Figure 5.80. Diagram Window of the Layer Extraction Algorithm (Threshold Step)

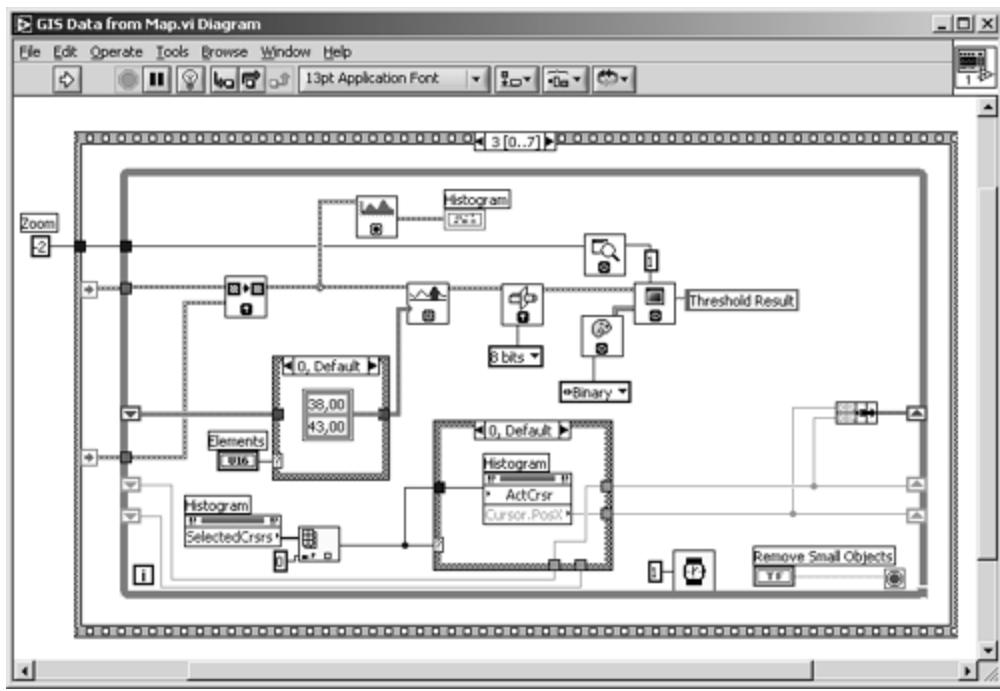
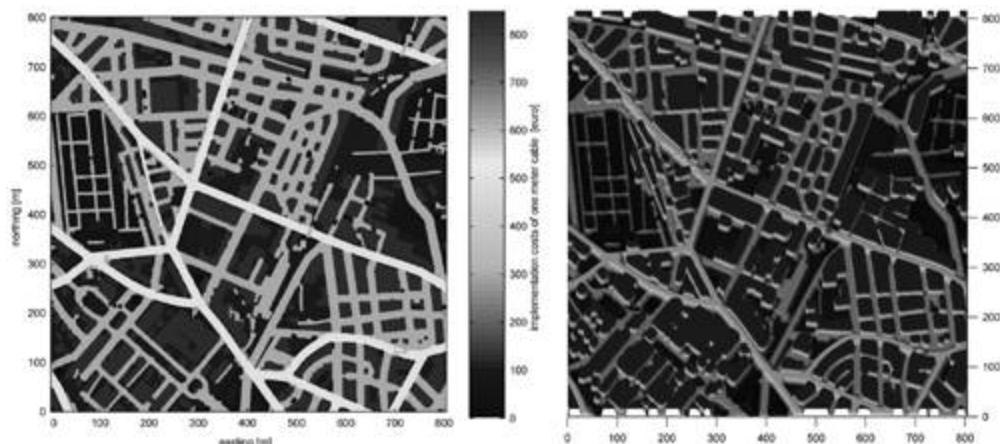


Table 5.2. Classes Used in NETQUEST

Land Class	Costs per m in Euro	Map Color
Built-up area	5000 (infinity)	Red
Private property	100	Green
Public property	50	Light green
Main streets	500	Yellow
Minor streets	300	White

The visualization of this information can be displayed in 2D and 3D images, shown in [Figure 5.81](#). For the final results, NETQUEST uses this information for the calculation of a cost-optimized fiber optic network.

Figure 5.81. Visualization of NETQUEST Results in 2D and 3D View



Feedback Form Reader Using IMAQ Vision

by Thomas KLINGER

(Paper presented at NIWeek 2002) [[46](#)]

(similar paper presented at VIP 2003, Munich)

Category

- Academic

Products Used

- LabVIEW Version 6i
- IMAQ Vision for LabVIEW Version 6.0

The Challenge

The students' feedback for the courses at the Carinthia Tech Institute is realized with a feedback form. The evaluation of this form is quite expendable, because it has to be done manually. Afterwards, these values have to be manually transferred into a PC software to get valuable statistical results.

Figure 5.82. Feedback Form Prepared for Automatic Reading

Feedback Form MEDICAL IT

_____. Semester

Lecturer: _____

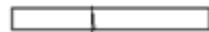
**A. The Course**

The meaning and role of the course in the curriculum is	<i>clearly visible</i> <input checked="" type="checkbox"/>	<i>not visible</i>
The structure of the content was	<i>clearly visible</i>	<i>not visible</i>
I understood the contents of the course	<i>yes</i>	<i>no</i>
My prerequisites for the understanding were	<i>sufficient</i>	<i>insufficient</i>
Overall, I am rating the entire course with:	<i>very good</i>	<i>very bad</i>

B. The Lecturer

... uses a lot of examples during the course	<i>yes</i> <input checked="" type="checkbox"/>	<i>no</i>
... realizes the course very interesting	<i>yes</i>	<i>no</i>
... aspires comprehensive lecturing	<i>yes</i>	<i>no</i>
... is open minded to student's proposals	<i>yes</i>	<i>no</i>
Overall, I am rating the lecturer with:	<i>very good</i> <input checked="" type="checkbox"/>	<i>very bad</i>

Place your mark with a vertical line inside the red framed box; e.g.:
 Do not use red pen or pencil !!!
 For suggestions and proposals please use back side of this form.

**The Solution**

A LabVIEW and IMAQ Vision program is able to read the images generated by a flat-bed scanner to extract the relevant data using color information, to measure the value of each answer, and to transfer the data into Microsoft Excel.

Students' Feedback Procedure

In each course at the Carinthia Tech Institute, the students have to fill in a feedback form consisting of ten questions dealing with the course itself and the lecturer. [Figure 5.82](#) shows an example of a form sheet, which is already prepared for automatic reading and evaluation.

Formerly, the students as well as the lecturers had two possibilities:

1. Fill out a feedback form on paper. Usually, the students' acceptance is very good for this method; however, manually obtaining a statistical result out of over 1,000 feedback forms is a lot of work to do.
2. Use a computer program for the questions and the evaluation as well; this leads to statistical results immediately. On the other hand, students do not prefer this method, because it might be possible to identify the origin of the votes.

Therefore, the combination of both methods is ideal; if paper feedback forms can be read automatically, the students can fill in their feedback forms where- and whenever they want.

Modification of the Feedback Form

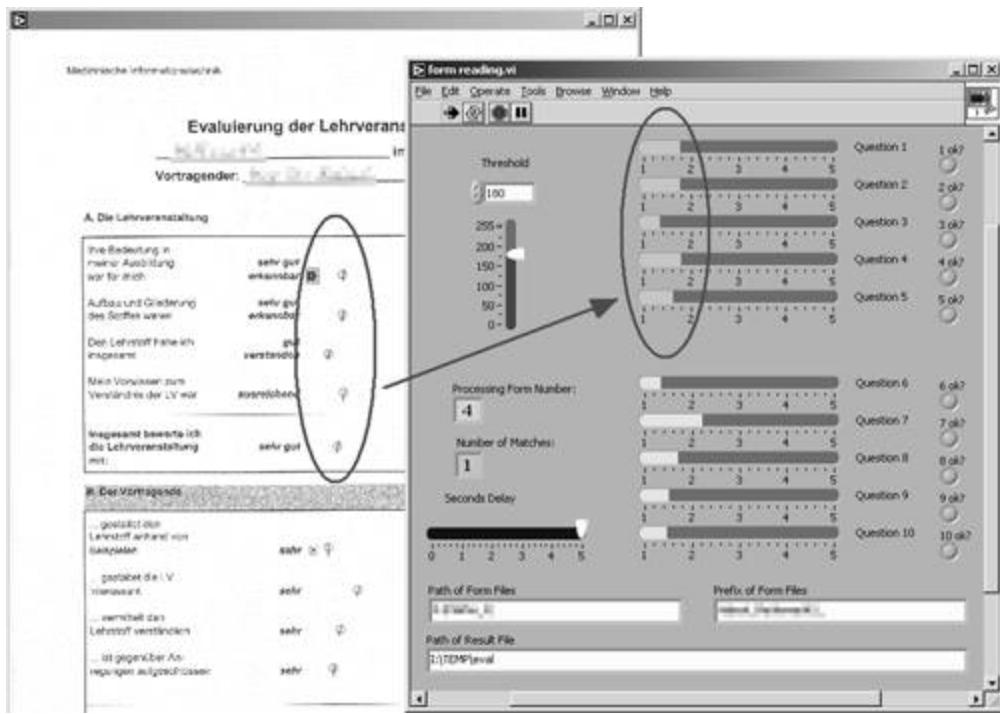
First of all, some modifications were made to the original feedback form:

1. The area for the students' voting mark is indicated by a red frame (see also [Figure 5.82](#)). When the frame is read by the application, only the red plane of the color scan is used, in order to make the red frame disappear. This is the reason why the students are not allowed to use red pens or pencils.
2. Three marks are positioned at the beginning of three voting areas, vertically distributed over the entire page; they are used for the location of a starting position, using pattern matching.

Functionality of the Form Reader Software

The results of the algorithm are displayed in the front panel of the Feedback Form Reader (shown in [Figure 5.83](#)) and afterwards written to a spreadsheet file, which later can be used for statistical evaluation. The 10 LEDs at the right side of [Figure 5.83](#) indicate whether the mark detection algorithm was successful or not. In order to watch the entire process and to verify the results, a delay of some seconds can be adjusted with a slider control. If the delay is set to 0, it takes about 5 seconds for the scanning of 30 feedback forms.

Figure 5.83. Results of the Form Reader Compared with Original Values



[Figure 5.83](#) also shows the coherence of the marks in the feedback form with the obtained results of the Feedback Form Reader. If the results do not match at all because the marks cannot be detected, it is possible to adjust the threshold level for the edge detection function.

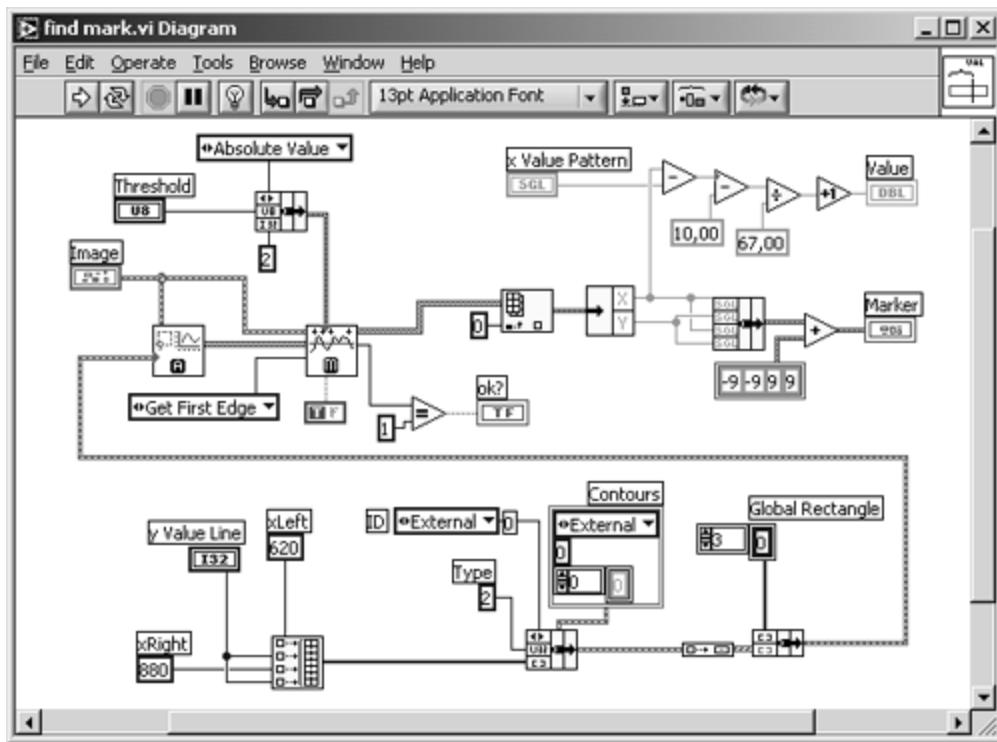
Mark Detection Algorithm

The software has to perform the following functions:

- Detect the exact position of the form sheet using pattern matching and—if necessary—calculate a correction factor for the x coordinates of the mark areas.
- Using the x coordinate of the pattern and the correction factor, find out if there is a mark located in the mark area. If a mark is detected with a line profile edge detector, the distance can be calculated and transformed into a value from 1 (very good) to 5 (very bad).
- Collect the calculated values in a $n \times 10$ array, where n is the number of feedback forms, and write this array to disk.

[Figure 5.84](#) shows the diagram of the subroutine `find mark.vi`. This VI uses the x coordinate from the pattern matching algorithm as input and detects the mark line using the functions `IMAQ ROIProfile` and `IMAQ SimpleEdge`. If an edge is found, the Boolean output `ok?` is set to true.

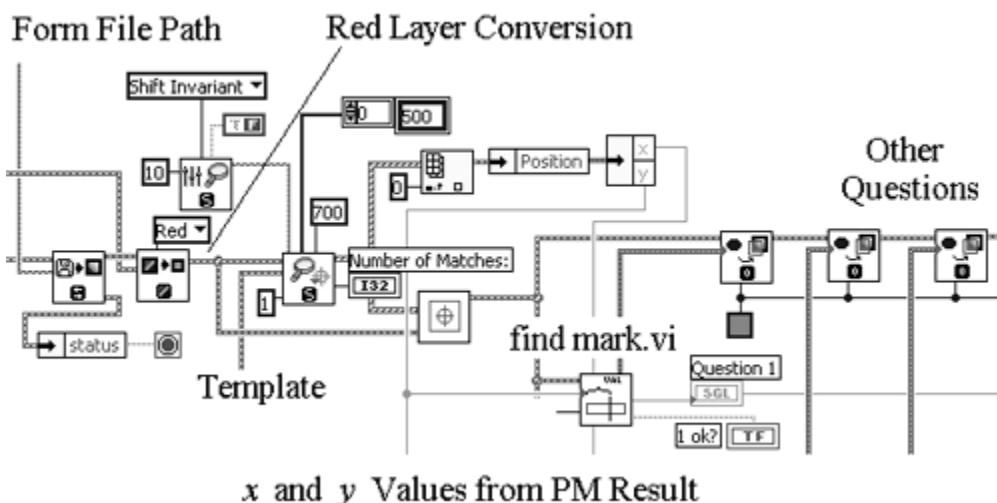
Figure 5.84. Block Diagram of `find mark.vi`



The x coordinate of the mark and the x coordinate of the pattern are used for the calculation of the final value (see top-right corner of [Figure 5.84](#)). The x and y coordinates of the mark are used to draw a small green circle in the original image at the place the mark was detected (see also [Figure 5.83](#)).

Finally, [Figure 5.85](#) shows the integration of `find mark.vi` into the main program. The entire block diagram is too big to show it here, so only the pattern matching (PM) function and one call of `find mark.vi` (the evaluation of one question) is shown.

Figure 5.85. Block Diagram of the Main Program



Conclusion

With this program, we found a nearly perfect solution for the combination of the use of handwritten feedback forms and automatic evaluation. Since Autumn 2001, this system is used at the Carinthia Tech Institute with success.

[\[Team LiB \]](#)

[!\[\]\(74d4b616ab5dd96b19ba36ee28efef9c_img.jpg\) PREVIOUS](#) [!\[\]\(ded7621f32c9032dea44f8f08190e19f_img.jpg\) NEXT !\[\]\(2414b9b0d0f6548e39641b3c90a51181_img.jpg\)](#)

Bibliography

[Chapter 1:](#)

[Chapter 2:](#)

[Chapter 3:](#)

[Chapters 4 and 5:](#)

[Application Papers:](#)

Bibliography

Chapter 1:

1. *LabVIEW™ User Manual* . Part Number 320999C-01, National Instruments, 2000.
2. *LabVIEW™ Measurements Manual* . Part Number 322661A-01, National Instruments, 2000.
3. *G Programming Reference Manual* . Part Number 321296B-01, National Instruments, 1998.
4. *IMAQ™ Vision User Manual* . Part Number 322320B-01, National Instruments, 1999.
5. *IMAQ™ Vision for G Reference Manual* . Part Number 322366A-01, National Instruments, 1999.
6. *NI-IMAQ™ for IEEE-1394 Cameras User Manual* . Part Number 370362A-01, National Instruments, 2001.
7. *IMAQ™ Vision Builder Tutorial* . Part Number 322228C-01, National Instruments, 2000.
8. *IMAQ™ Vision Builder Concepts Manual* . Part Number 322916A-01, National Instruments, 2000.
9. *IMAQ™ Vision Builder Release Notes* . Part Number 322604B-01, National Instruments, 2000.
10. *NI Vision Builder for Automated Inspection Tutorial* . Part Number 323379A-01, National Instruments, 2002.
11. *NI Vision Builder for Automated Inspection Release Notes* . Part Number 323518A-01, National Instruments, 2002.
12. Haberaecker P. *Praxis der Digitalen Bildverarbeitung und Mustererkennung* . Hanser, 1998.
13. Davies E. R. *Machine Vision; Theory, Algorithms, Practicalities* . Academic Press, 1997.

Bibliography

Chapter 2:

14. Reisch M. *Elektronische Bauelemente: Funktion, Grundschaltungen, Modellierung mit SPICE*. Springer, 1998.
15. Ott E. "Wissenswertes über CCD-Kameras." *PCO CCD Imaging*, 1997.
16. Gonzalez R. C., Woods R. E. *Digital Image Processing*. Addison-Wesley, 1993.
17. Litwiller D. "CCD vs. CMOS: Facts and Fiction." *Photonics Spectra*, Laurin Publishing, January 2001.
18. Theuwissen A., Roks E. "Building a Better Mousetrap." *SPIE's oeMagazine*, January 2001.
19. Krestel E. *Imaging Systems for Medical Diagnostics*. Publicis MCD, 1990.
20. Kalender W. A. *Computed Tomography*. Publicis MCD, 2000.
21. Beutel J., Kundel H. L., Van Metter R. L. *Handbook of Medical Imaging, Volume 1: Physics and Psychophysics*. SPIE Press, 2000.
22. Beutel J., Kundel H. L., Van Metter R. L. *Handbook of Medical Imaging, Volume 2: Medical Image Processing and Analysis*. SPIE Press, 2000.
23. Beutel J., Kundel H. L., Van Metter R. L. *Handbook of Medical Imaging, Volume 3: Display and PACS*. SPIE Press, 2000.
24. Bankman I. N. (Editor) *Handbook of Medical Imaging*. Academic Press, 2000.
25. Hajnal J. V., Hill D. L. G., Hawkes D. J. (Editors) *Medical Image Registration*. CRC Press, 2001.

Bibliography

Chapter 3:

26. Anderson D. *FireWire System Architecture, Second Edition* . Mindshare, Inc. (Addison-Wesley), 1999.
27. Anderson D. *Universal Serial Bus System Architecture* . Mindshare, Inc. (Addison-Wesley), 1998.
28. Bhaskaran V., Konstantinides K. *Image and Video Compression Standards: Algorithms and Architectures* . Kluwer, 1997.
29. Hoffman R. *Data Compression in Digital Systems* . Chapman & Hall, 1997.
30. Born G. *Dateiformate—Die Referenz* . Galileo Press, 2000.

Bibliography

Chapters 4 and 5:

31. Jain A. K. *Fundamentals of Digital Image Processing* . Prentice Hall, 1989.
32. Castleman K. R. *Digital Image Processing* . Prentice Hall, 1996.
33. Crane R. *A Simplified Approach to Image Processing* . Prentice Hall PTR, 1997.
34. Russ J. C. *The Image Processing Handbook, Third Edition* . CRC, Springer, IEEE Press, 1999.
35. Jaehne B. *Digitale Bildverarbeitung* . Springer, 1997.
36. Jaehne B., Haussecker H., Geissler P. *Handbook of Computer Vision and Applications, Volume 1: Sensors and Imaging* . Prentice Hall, 1999.
37. Jaehne B., Haussecker H., Geissler P. *Handbook of Computer Vision and Applications, Volume 2: Signal Processing and Pattern Recognition* . Prentice Hall, 1999.
38. Jaehne B., Haussecker H., Geissler P. *Handbook of Computer Vision and Applications, Volume 3: Systems and Applications* . Prentice Hall, 1999.
39. Myler H. R., Weeks A. R. *The Pocket Handbook of Image Processing Algorithms in C* . Prentice Hall PTR, 1993.

Application Papers:

40. Klinger T., Madritsch Ch., Sterner H. "EMC Test Automation using Motion and Vision." *NIWeek Paper*, 1999.
41. Klinger T., Madritsch Ch., Sterner H. "Object Detection and Counting in Public Places." *NIWeek Paper*, 2000.
42. Madritsch Ch., Klinger T., Sterner H. "Synchronous Video and Data Acquisition." *NIWeek Paper*, 2000.
43. Sterner H., Klinger T., Madritsch Ch. "Measurement System for Infrared Detectors." *NIWeek Paper*, 2000.
44. Haselberger M., Klinger T., Sterner H. "LabVIEW and IMAQ Vision enable PL EEG including Synchronous Video Recording." *NIWeek Paper*, 2001.
45. Klinger T., Bachhiesl P., Paulus G., Werner J., Stoegner H. "Extracting GIS Data for Telecommunication Networks from City Maps." *NIWeek Paper*, 2002.
46. Klinger T. "Students' Feedback Form Reader using LabVIEW and IMAQ Vision." *NIWeek Paper*, 2002.

About the Author



Thomas Klinger received his M.Sc. degree in 1989 and his Ph.D. in 1992 in electronic engineering from the University of Technology, Vienna, Austria. The major topics of his studies were biomedical engineering and biosensors. He then spent six years in the development department of Philips Domestic Appliances and Personal Care products in Klagenfurt, Austria. At Philips he became familiar with LabVIEW and other National Instruments' products.

In 1998 he joined the Carinthia Tech Institute, University of Applied Sciences (Carinthia, Austria) as professor of electronics, electromagnetic compatibility, and electronic system integration. Two years later he started to manage the new CTI course in Medical Information Technology; currently he holds lectures about medical image processing at CTI.

Thomas is a certified trainer for LabVIEW basic courses and a member of the Institute of Electrical and Electronics Engineers and the Engineering in Medicine and Biology Society. For several years he has published papers using LabVIEW and IMAQ Vision at various events, including NIWeek.

About the CD-ROM

The CD-ROM included with *Image Processing with LabVIEW and IMAQ Vision* contains the following:

Sections

Code Examples: You may copy the desired examples to your hard drive or open them directly from the CD menu.

Color Images: The CD-ROM contains most of the book's images in .jpg file format. If color can support the understanding of the image, this image is provided in a color version.

Application Examples: [Chapter 5](#) of *Image Processing with LabVIEW and IMAQ Vision* contains some of the author's programming examples, which resulted in papers presented at various NIWeek conferences. Most of this software is included on the CD.

Evaluation Software

IMAQ™ Vision Multimedia Demo: Upon selection, run Setup from the CD to install the demo software on your hard drive.

NI Vision Builder for Automated Inspection Evaluation Software: Upon selection, run Setup from the CD to install the demo software on your hard drive.

Information Sources

Links to National Instruments, MediaChance, 1394 Trade Organization, Basler Cameras, Pegasus Software, University of Mannheim (Germany), Aware, DICOM/NEMA, AccuSoft, and OTech.

The CD-ROM can be used on Microsoft Windows 98/NT/Me/2000/XP. Note that some of the open source VIs included are platform independent.

License Agreement

Use of the software accompanying *Image Processing with LabVIEW and IMAQ Vision* is subject to the terms of the License Agreement and Limited Warranty, found on the previous two pages.

[\[Team LiB \]](#)

 PREVIOUS

Technical Support

Prentice Hall does not offer technical support for any of the programs on the CD-ROM. However, if a CD-ROM is damaged, you may obtain a replacement copy by sending an email that describes the problem to disc_exchange@prenhall.com.

[\[Team LiB \]](#)

 PREVIOUS