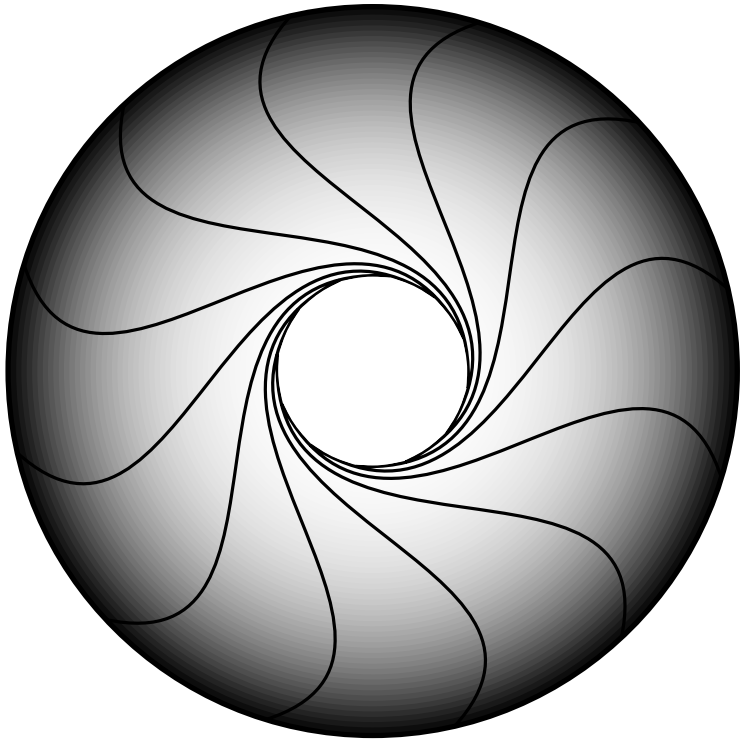
AdeptVision

Reference Guide

Version 12.1



Part Number 00962-01300, Rev. A
August 1997

adept technology, inc.

150 Rose Orchard Way • San Jose, CA 95134 • USA • Phone (408) 432-0888 • Fax (408) 432-8707

Otto-Hahn-Strasse 23 • 44227 Dortmund • Germany • Phone (49) 231.75.89.40 • Fax(49) 231.75.89.450

41, rue du Saule Trapu • 91300 • Massy • France • Phone (33) 1.69.19.16.16 • Fax (33) 1.69.32.04.62

1-2, Aza Nakahara Mitsuya-Cho • Toyohashi, Aichi-Ken • 441-31 • Japan • (81) 532.65.2391 • Fax (81) 532.65.2390

The information contained herein is the property of Adept Technology, Inc., and shall not be reproduced in whole or in part without prior written approval of Adept Technology, Inc. The information herein is subject to change without notice and should not be construed as a commitment by Adept Technology, Inc. This manual is periodically reviewed and revised.

Adept Technology, Inc., assumes no responsibility for any errors or omissions in this document. Critical evaluation of this manual by the user is welcomed. Your comments assist us in preparation of future documentation. A form is provided at the back of the book for submitting your comments.

Printed in the United States of America

Table of Contents

List of Figures

List of Tables

1 Introduction

Compatibility

This reference guide is for use with Adept controllers equipped with the AdeptVision VXL option and V+ operating system version 12.1 or later. See **Appendix D** for details on using existing program code with 12.1 systems.

Conventions

Many operating system options require you to hold down the keys marked "Shift" or "Ctrl" and then press another key. These types of key combinations are shown as "Ctrl+C", which means you should hold down the "Ctrl" key and press the "C" key.

Related Publications

This reference guide details the extensions added to the V+ language with the AdeptVision VXL option. It is a companion to the *V+ Language Reference Guide* and *V+ Operating System Reference Guide,* which cover the basic V+ language and operating system.

The following manuals may be required by your system:

Table 1-1. Related Publications

Manual	Material Covered
AdeptVision User's Guide	Enhancements to the V+ operating system that are added when the AdeptVision option is installed
Instructions for Adept Utility Programs	Instructions for running various setup and configuration software utilities
Adept MV Controller User's Guide	Instructions for setting up, configuring, and maintaining the controller that runs V+
Robot, motion, or force device user's guides (if connected to your system)	Instructions for installing and maintaining the motion device connected to your system
Manual Control Pendant User's Guide (if connected to your system)	Using the manual control pendant to move the robot and interact with motion control programs
V+ Operating System User's Guide	Details on the V+ operating system
V+ Operating System Reference Guide	Detailed descriptions of the V+ monitor commands

Table 1-1. Related Publications (Continued)

Manual	Material Covered
V+ Language User's Guide	V+ is a complete high-level language as well as an operating system. This manual covers programming principles for creating V+ programs.
V+ Language Reference Guide	Detailed descriptions of the keywords in the V+ language
User's guides for any AIM software that may be installed on your system	Details on using AIM applications such as MotionWare or VisionWare
AIM software reference guides	Details on the structure of AIM software and AIM applications

AdeptVision Keyword Summary

Table 1-2 summarizes the keywords added with the AdeptVision VXL option. **Appendix A** is a quick reference guide to the AdeptVision keywords, listing their arguments and basic functions.

Table 1-2. AdeptVision Keyword Summary

Keyword	Description
V.2ND.MOMENTS	Enable computation of the best-fit ellipse for each region in the image.
V.2ND.THRESH	Set a second threshold for use during binary image processing.
VABORT	Abort any active vision processing and all pending vision operations associated with the given task number.
VADD	Add two binary or grayscale images.
VAUTOTHR	Determine good thresholds for binary image processing based on the gradients in a grayscale frame store.
V.BACKLIGHT	Establish the color (black or white) of the background.
V.BINARY	Enable or disable automatic edge-image generation at VPICTURE time.
V.BORDER.DIST	Define an image border reduction (in pixels) to mask out regions clipped by the image border.
V.BOUNDARIES	Enable or disable boundary analysis by the vision system.
V.CENTROID	Enable computation of the centroid of each region in the image.
VCONVOLVE	Perform an image convolution on a grayscale frame, possibly storing the result in a different frame store.
VCOPY	Copy all or part of an image from one operation region definition to another.

Table 1-2. AdeptVision Keyword Summary (Continued)

Keyword	Description
VCORRELATE	Perform a normalized correlation, comparing a predefined template with a rectangular image window, or searching for a closest match within the window to a predefined template.
VDEF.AOI	Define an area-of-interest. Areas-of-interest are used by most vision tools to specify the tool placement within an image.
VDEF.CONVOLVE	Define an image convolution.
VDEF.FONT	Define, replace, or modify an Optical Character Recognition (OCR) font.
VDEFGRIP	Define the shape and position of a robot gripper for clear-grip tests.
VDEF.LUT	Define a user look-up table for mapping graylevel values.
VDEF.MORPH	Define a binary morphological operation.
VDEF.SUBPROTO	Define a subprototype.
VDEF.TRANS	Define a transformation to apply to the location of all vision tools placed until the next VDEF.TRANS instruction.
VDELETE	Delete a specified prototype, subprototype, Optical Character Recognition (OCR) font, or correlation template in the vision system.
V.DISJOINT	Determine whether or not prototypes may be matched to multiple disjoint regions.
VDISPLAY	Select the current vision display mode or the display mode to be used when the vision system performs its normal image processing functions.
V.DRY.RUN	Enable simulation of various vision measurement and inspection operators.
VEDGE	Compute edges in the grayscale image and threshold the edges, replacing the binary image, using either a cross gradient or Sobel algorithm.
VEDGE.INFO	Retrieve information about the edges and corners of a prototype or of a region in the image.
V.EDGE.INFO	Enable saving of information about edges in the image for recall via the VEDGE.INFO instruction.
V.EDGE.STRENGTH	Set the edge threshold for grayscale image processing and fine-edge rulers.
V.EDGE.TYPE	Determine the type of edge operator to use, cross gradient or Sobel, when a VPICTURE or VEDGE instruction is performed.
VFEATURE	Return specified information about the object most recently VLOCATEd or the prototype most recently displayed by the VSHOW program instruction.
VFIND.ARC	Fit a circular arc to an image edge bounded by a window that is shaped like a ring or a ring segment.
VFINDER	Perform ObjectFinder recognition using the planning associated with the given virtual camera.

Table 1-2. AdeptVision Keyword Summary (Continued)

Keyword	Description
VFIND.LINE	Fit a straight line to an image edge within a window.
VFIND.POINT	In a search window, find an edge point that is nearest to one side of the window.
V.FIRST.COL	Set the number of the first column of frame store data to be processed.
V.FIRST.LINE	Set the number of the first line of frame store data to be processed.
V.FIT.ARCS	Enable or disable the fitting of circular arcs when performing boundary analysis.
V.GAIN	Set the gain for the incoming video (camera) signal.
VGAPS	Find the unverified gaps in a match with a prototype or subprototype.
VGET.AOI	Return the definition of an area-of-interest.
VGETCAL	Ask the system to fill in arrays with the previously defined vision calibration data for a given virtual camera.
VGETPIC	Read all or part of an image into a string array.
VGET.TRANS	Return the value of the current vision transformation.
VHISTOGRAM	Compute the histogram for a grayscale frame store.
V.HOLES	Enable or disable the accounting of interior features in all objects.
V.IO.WAIT	Enable the synchronization of taking pictures (VPICTUREs) with an external event that triggers the fast digital-input interrupt line.
VISION	Enable the entire vision system.
V.LAST.COL	Set the number of the last column of frame store data to be processed.
V.LAST.LINE	Set the number of the last line of frame store data to be processed.
V.LAST.VER.DIST	Enable an extra verification of prototype-to-image matches and specify the pixel tolerance to use when determining boundary coincidence.
VLOAD	Load vision prototypes, Optical Character Recognition (OCR) fonts, or correlation templates from a disk file.
VLOCATE	Identify and locate an object in the scene.
V.MAX.AREA	Set the maximum area, above which the vision system ignores regions.
V.MAX.PIXEL.VAR	During a VFIND.LINE or VFIND.ARC operation, this parameter specifies the maximum pixel distance from the fit edge beyond which edge points may be filtered out. During boundary analysis, this parameter sets the maximum pixel deviation allowed when fitting lines and arcs to region edges.
V.MAX.SD	Set the distance (in units of standard deviation) from the fit line or arc beyond which edge points should be filtered out.

Table 1-2. AdeptVision Keyword Summary (Continued)

Keyword	Description
V.MAX.TIME	Set the maximum time allowed for the vision system to analyze a region during prototype recognition or OCR.
V.MAX.VER.DIST	Set the pixel tolerance for determining boundary coincidence during the verification of prototype-to-image matches.
V.MIN.AREA	Set the minimum area, below which the vision system ignores regions.
V.MIN.HOLE.AREA	Set the minimum area, below which the vision system ignores holes.
V.MIN.LEN	Set the minimum length of features to be used for feature pairs.
V.MIN.MAX.RADII	Enable the feature that, for each region in the image, finds the two points on the perimeter that are closest to and farthest from the region centroid.
VMORPH	Perform a morphological transform on a binary image frame.
VMORPH	Set the offset for the incoming video signal (that is, program the zero reference for the A / D converter).
VOCR	Perform Optical Character Recognition (OCR) or text verification in a rectangular image window.
V.OVERLAPPING	Determine whether or not objects may be overlapping in the image and still be recognized.
V.PERIMETER	Enable computation of the lengths of region perimeters.
VPICTURE	Grab an image for processing.
VPLAN.FINDER	Set up the type of “planning” used by the Finder when locating models.
VPUTCAL	Give the system arrays filled with vision calibration data.
VPUTPIC	Store into a frame store an image saved previously with VGETPIC.
VQUEUE	Display object information for any objects queued in the vision system, awaiting retrieval by VLOCATE instructions.
VQUEUE	Return the number of objects in the vision system queue.
V.RECOGNITION	Enable or disable prototype recognition by the vision system.
VRENAME	Rename a prototype or subprototype.
VRULERI	Obtain edge information or graylevels along a line or circular arc in the current image.
VSELECT	Select one of the two grayscale frame stores for processing or display and, optionally, select a virtual camera and its calibration to be associated with the frame store.
VSHOW	List the defined prototypes or display a vision prototype, a subprototype, or a specific prototype edge in the Vision display window. Edge numbers are optionally shown.

Table 1-2. AdeptVision Keyword Summary (Continued)

Keyword	Description
VSHOW	Display a vision prototype or subprototype and make information about it available through the VFEATURE real-valued function.
V.SHOW.BOUNDS	Enable the special display of the lines and arcs fit to the boundaries of regions.
V.SHOW.EDGES	Enable the special display of edges—both the primitive edges that are fit to the boundaries of regions, and the edge points that are found by the finders VFIND.LINE, VFIND.ARC, and VFIND.POINT.
V.SHOW.FEATS	Enable the special display of features used for ObjectFinder recognition.
V.SHOW.GRIP	Enable the special display of clear-grip tests.
VSHOW.MODEL	Display a model—either a correlation template or an Optical Character Recognition (OCR) font—and return information about it; or return information about all the defined templates or OCR fonts.
V.SHOW.RECOG	Enable the special display of the objects recognized.
V.SHOW.VERIFY	Enable the special display of the verification step in the recognition process.
VSTATUS	Display vision system status information in the Monitor display window.
VSTATUS	Return vision system status information in a real array.
VSTORE	Store in a disk file selected (or all) vision prototypes (and their subprototypes), Optical Character Recognition (OCR) fonts, or correlation templates.
V.STROBE	Enable the firing of a strobe light in synchronization with taking pictures (VPICTUREs).
VSUBPROTO	Determine the percentage of an edge or subprototype that was verified during recognition. Also, this instruction can have the prototype position refined, based on only a subprototype or a single edge, producing an adjusted location for the prototype.
VSUBTRACT	Subtract two binary or grayscale images.
V.SUBTRACT.HOLE	Determine whether or not hole areas are to be subtracted from region areas.
V.SYNC.STROBE	Select synchronous or asynchronous firing of a strobe light when a picture is taken (that is, when a VPICTURE is executed).
VTHRESHOLD	Threshold a grayscale image, producing a binary image.
V.THRESHOLD	Set the camera grayscale value that separates black pixels from white pixels.
V.TOUCHING	Determine whether or not objects may be touching in the image and still be recognized.
VTRAIN	Initiate training of the object whose name is specified.
VTRAIN.FINDER	Initiate training of the finder model whose name is specified

Table 1-2. AdeptVision Keyword Summary (Continued)

Keyword	Description
VTRAIN.MODEL	Train on a vision "model"—a correlation template or an Optical Character Recognition (OCR) font. For correlation, this instruction defines the template. For OCR, this instruction trains the vision system to recognize characters in a font, or causes the vision system to plan the recognition strategy for a fully trained font.
VWAIT	Delay program execution until processing of a VPICTURE or VWINDOW operation is complete.
VWINDOW	Perform processing (blob finder, prototype recognition, or ObjectFinder) within a rectangular window in the image.
VWINDOWB	Extract basic image information from within a rectangular window.
VWINDOWI	Extract image information from within a window with any of the following shapes: rectangle, circle, pie cut, ring, or ring segment.

How Can I Get Help?

The following section tells you who to call if you need help.

Within the Continental United States

Adept Technology maintains a Customer Service Center at its headquarters in San Jose, CA. The phone numbers are:

Service Calls

(800) 232-3378 (24 hours a day, 7 days a week)
(408) 433-9462 FAX

> **NOTE:** When calling with a controller-related question, please have the serial number of the controller. If your system includes an Adept robot, also have the serial number of the robot. The serial numbers can be determined by using the ID command (see the *V+ Operating System User's Guide*).

Application Questions

If you have an application question, you can contact the Adept Applications Engineering Support Center for your region:

Adept Office	Phone #, Hours	Region
San Jose, CA	Voice (408) 434-5033 Fax (408) 434-6248 8:00 A.M. – 5:00 P.M. PST	Western Region States: AR, AZ, CA, CO, ID, KS, LA, MO, MT, NE, NM, NV, OK, OR, TX, UT, WA, WY
Cincinnati, OH	Voice (513) 792-0266 Fax (513) 792-0274 8:00 A.M. – 5:00 P.M. EST	Midwestern Region States: AL, IA, IL, IN, KY, MI, MN, MS, ND, West NY, OH, West PA, SD, TN, WI
Southbury, CT	Voice (203) 264-0564 Fax (203) 264-5114 8:00 A.M. – 5:00 P.M. EST	Eastern Region States: CT, DE, FL, GA, MD, ME, NC, NH, MA, NJ, East NY, East PA, RI, SC, VA, VT, WV

Applications Internet E-Mail Address

If you have access to the Internet, you can send application questions by e-mail to:

```
adeptinfo@infolab.com
```

This method also enables you to attach a file, such as a portion of V+ program code, to your message.

NOTE: Please attach only information that is formatted as text.

Training Information

For information regarding Adept Training Courses in the USA, please call (408)474-3246 or send a FAX message to (408)474-3226.

Within Europe

Adept Technology maintains a Customer Service Center in Dortmund, Germany. The phone numbers are:

(49) 231 / 75 89 40 from within Europe (Monday to Friday, 8:00 A.M. to 5:00 P.M.)
(49) 231/75 89 450 FAX

France

For customers in France, Adept Technology maintains a Customer Service Center in Massy, France. The phone numbers are:

(33) 1 69 19 16 16 (Monday to Friday, 8:30 A.M. to 5:30 P.M., CET)
(33) 1 69 32 04 62 FAX

Outside Continental United States or Europe

For service calls, application questions, and training information, call the Adept Customer Service Center in San Jose, California USA:

1 (408) 434-5000
1 (408) 433-9462 FAX (service requests)
1 (408) 434-6248 FAX (application questions)

Adept Fax on Demand

Adept maintains a fax back information system for customer use. The phone numbers are (800) 474-8889 (toll free) and (503)207-4023 (toll call). Application utility programs, product technical information, customer service information, and corporate information is available through this automated system. There is no charge for this service (except for any long-distance toll charges). Simply call either number and follow the instructions to have information faxed directly to you.

Adept on Demand Web Page

If you have access to the Internet, you can view Adept's web page at the following address:

```
http://www.adept.com
```

The web site contains sales, customer service, and technical support information.

2 Descriptions of Vision Keywords

This chapter presents detailed descriptions of all the V+ system keywords related to vision. These keywords are available only to systems with the AdeptVision VXL option.

The global system switch VISION enables and disables all vision processing. If this switch is DISABLEd, the system will behave as if the vision option is not installed. The VISION switch default value is ENABLEd.

The vision system includes the following types of keywords:

- Program instructions
- Functions
- Monitor commands
- System parameters
- System switches

The keywords are presented in alphabetical order, with the description for each keyword starting on a new page. Note that some keywords may belong to more than one keyword type. The keyword is shown at the top of each page. The sections described on pages 18 and 19 are included for each keyword, as appropriate.

Syntax

This section presents the syntax of the keyword. The keyword is shown in uppercase and the arguments are shown in lowercase. The keyword must be entered exactly as shown.[1] Parentheses must be placed exactly as shown. Required keywords, parameters, and marks such as equal signs and parentheses are shown in bold type; optional keywords, parameters, and marks are shown in regular type. In the example:

KEYWORD req.param1 = req.param2 OPT.KEYWORD opt.param

KEYWORD	must be entered exactly as shown[1]
req.param1	must be replaced with a value, variable, or expression
=	the equal sign must be entered
req.param2	must be replaced with a value, variable, or expression
OPT.KEYWORD	can be omitted but must be entered exactly as shown if used
opt.param	may be replaced with a value, variable, or expression but assumes a default value if not used

The keyword type (function, program instruction, and so on) is shown at the top of the page.

An abbreviated syntax is shown for some keywords. This is done when the abbreviated form is the most commonly used variation of the complete syntax.

Function

This section gives a brief description of the keyword.

Usage Considerations

This section lists any restrictions on the keyword's use. If specific hardware or other options are required, they will be listed here.

[1] In the SEE editor, instructions can be abbreviated to a length that uniquely identifies the keyword. The SEE editor will automatically expand the instruction to its full length.

Parameters

The requirements for input and output parameters are explained in this section. If a parameter is optional, it will be noted here. When an instruction line is entered, optional parameters do not have to be specified and the system will assume a default. Unspecified parameters at the end of an argument list can be ignored. Unspecified parameters in the middle of an argument list must be accounted for by commas. For example, the following keyword has four parameters—the first and third are used, and the second and fourth are left unspecified:

```
SAMPLE.INST var_1,,"test"
```

String and numeric input parameters can be constant values (3.32, “part_1”, etc.) or any legitimate variable name (see Chapter 1 of the ***V+ Language User's Guide*** for the requirements of a variable name). The data type of the constant or variable must agree with the type expected by the keyword. String variables must be preceded by a “$”. Precision-point variables must be preceded by a “#”. Belt variables must be preceded by a “%”. String constants must be enclosed in quotes. Real and integer constants can be used without modification. (V+ keywords cannot be used as variable names—see **Appendix A** for a complete list of keywords.)

Details

This section describes the function of the keyword in detail.

Examples

Examples of correctly formed instruction lines are presented in this section.

Related Keywords

Additional keywords that are similar or are frequently used in conjunction with this instruction are listed here.

Syntax

```
... V.2ND.MOMENTS [camera]
```

Function

Enable computation of the best-fit ellipse for each region in the image.

Usage Considerations

The system switches V.CENTROID and V.BOUNDARIES must also be enabled in order for the second moments of inertia to be computed.

A change to this switch takes effect when the next VPICTURE command or instruction, or VWINDOW instruction, is executed.

This is an array of switches—one for each virtual camera. (See the general description of switches in ***AdeptVision User's Guide*** for syntax details.)

Details

V.2ND.MOMENTS enables the computation of the second moments of inertia for regions. Based on the second moments and centroid of each region, a "best-fit" ellipse is fit to the region.

The dimensions of the best-fit ellipse are available from the VFEATURE function after a VLOCATE instruction has succeeded. VFEATURE provides the ellipse major and minor radii and the angle of the ellipse major axis. The center of the ellipse is at the centroid of the region, which is also available from VFEATURE.

The areas and positions of holes are not considered during the computation of moments. (The V.SUBTRACT.HOLE system switch does not affect moments.) However, if you want these statistics for holes, you should enable V.HOLES, disable V.DISJOINT, and do VLOCATEs in get-hole mode.

The best-fit ellipse is computed as follows. The ellipse is centered on the region centroid. The major axis of the ellipse is aligned with the region axis of least inertia. The minor axis of the ellipse is perpendicular to the major axis. The radii of the ellipse axes are chosen so that the ellipse area and region area are equal, and their ratios of major / minor axes of inertia are equal.

Example

The following program segment draws a cross on each region in the image, showing the axes of the best-fit ellipses.

```
ENABLE V.BOUNDARIES, V.CENTROID, V.2ND.MOMENTS ;Necessary switches
ATTACH (vlun, 4) "GRAPHICS"            ;Attach to the vision window
FOPEN (vlun) "Vision /MAXSIZE 640 480" ; and select graphics scaling
GTRANS (vlun, 1)                       ; in real-world millimeters
vf.cx = 42                             ;Indexes of VFEATURE function
vf.cy = 43                             ; for centroid and best-fit
vf.major.axis = 48                     ; ellipse dimensions
vf.major.radius = 49
vf.minor.radius = 50

VDISPLAY 3                             ;Special display
VPICTURE ,0                            ;Take a picture with virtual
                                       ; camera 1 and no recognition

VWAIT                                  ;Wait for image processing to
                                       ; complete for graphics instr.
VLOCATE () $nam                        ;Locate anything in the image

WHILE VFEATURE(1) DO                   ;If a region was found...
     cx = VFEATURE(vf.cx)              ;Get centroid: Cx,Cy
     cy = VFEATURE(vf.cy)
     maj = VFEATURE(vf.major.radius)   ;Get min/max radii of ellipse
     min = VFEATURE(vf.minor.radius)
     ang = VFEATURE(vf.major.axis)     ;Angle of major axis
     dx = COS(ang)*maj                 ;Draw major axis
     dy = SIN(ang)*maj
     GLINE (vlun) cx+dx, cy+dy, cx-dx, cy-dy

     ang = ang+90                      ;Draw minor axis
     dx = COS(ang)*min
     dy = SIN(ang)*min
     GLINE (vlun) cx+dx, cy+dy, cx-dx, cy-dy

     VLOCATE () $nam                   ;Locate next region in the image
END
```

Related Keywords

VFEATURE (real-valued function)
V.BOUNDARIES (system switch)
V.CENTROID (system switch)
V.DISJOINT (system switch)
V.HOLES (system switch)
VLOCATE (program instruction)
V.MIN.MAX.RADII (system switch)
V.PERIMETER (system switch)

Syntax

```
... V.2ND.THRESH [camera]
```

Function

Set a second threshold for use during binary image processing.

Usage Considerations

Changing this parameter immediately affects the video input to the binary frame buffers, assuming the VISION switch is enabled.

This is an array of parameters—one for each virtual camera. (See the general description of parameters in *AdeptVision User's Guide* for syntax details.)

Details

This is an optional second threshold that may be used with the primary threshold V.THRESHOLD to separate "black" pixels from "white" ones. V.2ND.THRESH and V.THRESHOLD define a range of grayscale values between which camera pixels are considered to be black. Pixel values outside this range are considered to be white. For example, if the value of V.2ND.THRESH is 50 and the value of V.THRESHOLD is 70, camera pixels in the range 50 to 70 are interpreted as being black, and pixels in the ranges 0 to 49 and 71 to 127 are interpreted as being white. (The V.BACKLIGHT system switch determines whether white or black is the background color.)

Normally, one threshold is sufficient for binary image processing. V.2ND.THRESH should then be assigned the value 0 to disable its use. Dual thresholds are sometimes very useful, however. For example, suppose the task is to recognize a black and white checkerboard on a middle-gray background. By setting the V.THRESHOLD and V.2ND.THRESH values so that the grayscale range includes the middle-gray background, the checkerboard will appear as one large square in the scene. As an another example, the object to be recognized may be a uniform middle-gray color that is lying on a checkerboard background.

In general, the correct values for V.THRESHOLD and V.2ND.THRESH depend on the particular application. The program instruction VAUTOTHR may be used to automatically determine thresholds.

This parameter must be assigned an integer value in the range 0 to 127, inclusive. The parameter is set to 0 (that is, its effect is disabled) when the V$^+$ and AdeptVision systems are loaded into memory from disk.

Examples

Set dual thresholds for a region of interest with intensity range 45 to 67:

```
PARAMETER V.2ND.THRESH = 45
PARAMETER V.THRESHOLD = 67
```

Related Keywords

VAUTOTHR (monitor command and program instruction)
V.BINARY (system parameter)
V.THRESHOLD (system parameter)

Syntax

VABORT task_id

Function

Abort any active vision processing and all pending vision operations associated with the given task number.

Parameter

task_id	Optional real-valued expression specifying the number of a V+ program execution task. The special task number –2 indicates all tasks. The special task number –1 indicates "self", meaning the task issuing the VABORT. Otherwise, "task_id" is an integer value in the range 0 to N, where N is the largest task number supported by your V+ system. The default value is –1, meaning "self".

Details

Most users will never need to use VABORT. When a user aborts a V+ program that is executing a vision instruction, or when the user aborts a vision monitor command by typing Ctrl+C, V+ automatically aborts the associated vision processing. In fact, V+ automatically sends the vision processor a VABORT request (for the respective execution task) whenever a task is primed for execution, starts executing, or completes execution. (This VABORT is sent to cancel any pending VPICTURE that might be waiting for a fast digital-input interrupt signal.)

The one vision instruction that can wait forever is a VPICTURE that waits for a fast digital-input interrupt signal. The V.IO.WAIT system parameter enables this wait mode. If the input signal never occurs, the VPICTURE request will never complete. VABORT, however, may be used to cancel the request.

In multitasking applications, one task may issue a VABORT to abort a vision instruction that another task is waiting on, thereby letting that task resume.

Only pending or executing vision instructions associated with the given task number are aborted. However, if the given task number is –2, all pending or executing vision instructions are aborted.

Example

Abort executing or pending vision instructions issued from task #0:

```
VABORT 0
```

Syntax

VADD (cam, type, dmode) **dest_ibr = src1_ibr, src2_ibr**

Function

Add two binary or grayscale images.

Parameters

cam	Selects the threshold parameters to apply to the result.
type	Optional integer value indicating the type of addition: 1 - for binary (default value) 2 - for average grayscale 3 - for summed grayscale
dmode	Optional real-valued expression specifying the display mode for this operator. The choices are: –1 = no draw, 0 = erase, 1 = draw solid, 2 = complement, 3 = draw dashed, and 4 = complement dashed. The default is 1 (draw solid).
dest_ibr	Integer value specifying the image buffer region that will receive the result of adding the source image buffer regions. Image buffer regions specify both a virtual frame buffer and an area of interest (see the description of VDEF.AOI).
src1_ibr	Integer values specifying the image buffer regions to add. The AOIs **src2_ibr** must have been defined with a VDEF.AOI instruction. The virtual frame stores specified for these two image buffer regions must be in different physical frame stores.

Details

The VADD operation adds two images. The images considered are identical areas of interest within two frame stores. The areas are defined using VDEF.AOI instructions. If the areas of interest are not the same size, the larger areas of interest are shrunk to the same dimensions as the smallest, but each keeps its original center location.

Each pixel in the area of interest for one image is added to the corresponding pixel in the other image, and the result is stored in the corresponding pixel of the destination image buffer region. The smaller the area of interest to be processed, the faster the addition executes.

VADD type #1 is binary addition. The two binary images are simply OR'd together, pixel by pixel. (In comparison, when the VSUBTRACT instruction is used in binary mode, the images are **exclusive** OR'd.) The underlying grayscale image in the destination image buffer region is left unmodified.

VADD type #2 adds the graylevel pixels in the two images together and divides each pixel sum by 2 to produce an average. The division operation rounds up. (That is, 1÷2 is 1, 2÷2 is 1, 3÷2 is 2, 4÷2 is 2, etc.) Averaging images may be useful for filtering out noise or for increasing pixel precision.

VADD type #3 adds the graylevel pixels in the two images, in effect increasing the image intensity. Pixel sums are clipped at the maximum graylevel of 127. That is, if the sum of two pixels exceeds 127, the sum is reduced to 127. Summing images may be useful for increasing the brightness of dim images. (Note, however, that the system parameters V.GAIN and V.OFFSET provide a more conventional method of increasing the dynamic range of images when acquiring them from the camera.)

With VADD types #2 and #3, the binary image associated with the destination image buffer region is created by applying the threshold parameters for the given virtual camera.

When the two source image buffer regions can be the same, the following special considerations apply:

- VADD type #1 is not performed. (This would simply OR the image with itself with no effect, and copy the result to the destination image buffer region.)
- VADD type #2 averages the image with itself (again an operation with no effect), but the image is copied to the destination image buffer region.
- VADD type #3 is the only meaningful operation. It doubles the intensity of the image buffer region and puts the result in `dest_ibr`.

Example

Add two image buffer regions and place the result in the first image buffer region:

```
aoi2 = 2000      ;AOI number 2
cx = 4.5         ;Center of AOI
cy = 8           ;  "
size = 5         ;Length of side of AOI
rect = 1         ;Use shape definition 1
ang = 0
VDEF.AOI aoi2 = rect, cx, cy, size, size, ang
dest = aoi2+11   ;AOI 2, virt. frame buffer 11
```

```
src1 = aoi2+11   ;AOI 2, virt. frame buffer 11
src2 = aoi2+12   ;AOI 2, virt. frame buffer 12
VADD dest = src1, src2
```

Related Keywords

VCOPY (program instruction)
VDEF.AOI (program instruction)
VEDGE (program instruction)
VGET.AOI (program instruction)
VSUBTRACT (program instruction)
VTHRESHOLD (program instruction)

Syntax

```
VAUTOTHR (dmode, start, end) array[index] = ibr
```

Function

Determine good thresholds for binary image processing based on the gradients in a grayscale frame store.

Parameters

`dmode`	Optional real-valued expression specifying the display mode: 1 to display the histogram and average contrast curve or 0 for no display. The default is 1 (display).
`start, end`	Optional integer values specifying the range of intensities in which to limit the search for good thresholds. These values must be in the range 0 to 127, inclusive. The value of "start" must be less than the value of "end". By default, "start" is 0 and "end" is 127.
`array[]`	Optional array into which the threshold values are placed. The first value in the array is the number of good thresholds found ("n"). The next "n" values in the array are the thresholds, in order of "goodness" (see below for details).
`index`	Optional integer value that identifies the first array element to be defined in "`array[]`". Zero is assumed if the index is omitted. If a multiple-dimension array is specified, only the right-most index is incremented as the values are assigned.
`ibr`	Optional real value, variable, or expression interpreted as an integer that specifies the image buffer region to threshold. The image buffer region specifies both an AOI and a frame store. See the definition of VDEF.AOI.

Details

VAUTOTHR computes a histogram for the image currently in memory and automatically determines good threshold values. With automatic threshold determination, the vision system can adapt to gradual changes in lighting over time. The sample program given below, for example, could be called once every 30 minutes to adjust the threshold.

VAUTOTHR finds "optimum" thresholds by computing the average contrast in the image produced by every threshold. The average contrast for every possible threshold (0 to 127) is shown as a red curve overlaying the gray-level histogram, which is shown in blue. Peaks in the red curve are the good thresholds—the higher the peak, the better the threshold.

The highest peak in the average contrast curve is not always the best threshold, however. For example, specular reflections on shiny parts may produce a high peak in the curve. The associated threshold would separate only the specular reflections from the rest of the image, instead of separating the object from the background. Since these specular reflections are usually the brightest part of the image, their intensity will be located at the high end of the histogram.

To automate the filtering out of high average contrasts near the extreme ends of the histogram, VAUTOTHR uses the system parameter V.MIN.AREA to trim the ends. That is, if the number of pixels in the histogram above or below a peak is less than V.MIN.AREA, the peak is discarded. This is done because there would be regions in the image with areas less than V.MIN.AREA if one of these discarded peaks were used as a threshold.

An effective strategy for auto-thresholding is first to choose manually the best threshold, and then later have VAUTOTHR search for a threshold within a limited range of the current threshold. To choose manually the best threshold, execute VAUTOTHR as a monitor command with no parameters. The entire intensity range is considered and the results are displayed. Then, overlay the results on the live binary image by executing the monitor command “VDISPLAY 0, 1”. Finally, try each of the “optimum thresholds” shown by changing the V.THRESHOLD parameter, and decide on the best setting.

The frame store must contain a valid picture. Otherwise, an error results. (See the VSELECT program instruction.)

Example

The following code computes a new threshold for the given virtual camera. The search range is (t–20) to (t+20), where “t” is the current threshold:

```
     LOCAL t, tmin, tmax
     VPICTURE (cam) 2
;Take a quick picture
     t = PARAMETER(V.THRESHOLD[cam])
;The current threshold
; Search range is +/-20 around the current threshold
     tmin = MAX(0, t-20)              ;Make sure it's >= 0
     tmax = MIN(127, t+20)            ;Make sure it's <= 127
     VWAIT
;Wait for image processing
     VAUTOTHR (0, tmin, tmax) thrs[]  ;Compute best threshold
     IF thrs[0] > 1 THEN              ;If got one, use it
          PARAMETER V.THRESHOLD[cam] = thrs[1]
     END
```

Related Keywords

VDEF.AOI (system parameter)
VHISTOGRAM (monitor command and program instruction)
V.2ND.THRESH (system parameter)
V.MIN.AREA (system parameter)
V.THRESHOLD (system parameter)

Syntax

```
... V.BACKLIGHT [camera]
```

Function

Define which color (black or white) is to be considered the background.

Usage Considerations

A change to this switch takes effect when the next VPICTURE command or instruction, or VWINDOW, VWINDOWB, or VWINDOWI instruction, is executed.

This is an array of switches—one for each virtual camera. (See the general description of switches in ***AdeptVision User's Guide*** for syntax details.)

V.BACKLIGHT affects binary operations only and has no effect on grayscale processing.

Details

If the V.BACKLIGHT switch is enabled, the vision system considers the visual background to be white. Otherwise, the background is considered to be black. The objects to be recognized are in the foreground, the opposite color of the background.

The borders of the image (that is, beyond the image limits set by the parameters V.FIRST.LINE, V.LAST.LINE, V.FIRST.COL, and V.LAST.COL) are always treated as background.

The state of V.BACKLIGHT is critically important for prototype training, for clear grip tests, and for "blob" analysis. You may train prototypes only on foreground objects. Gripper positions over foreground regions are never clear. V.MIN.AREA applies to outermost foreground regions. V.MIN.HOLE.AREA applies to all inner regions, both foreground and background.

However, the state of V.BACKLIGHT is not critical for prototype recognition. Prototypes are either black or white. In fact, one prototype may be black while another is white, and both may be found in the same image, regardless of the state of V.BACKLIGHT.

Example

Enable processing of backlit objects for virtual camera 3:

```
ENABLE V.BACKLIGHT[3]
```

Get the current setting for virtual camera 1:

```
val = SWITCH(V.BACKLIGHT[1])
```

Related Keywords

V.BINARY (system switch)
VWINDOW (program instruction)
VWINDOWB (program instruction)
VWINDOWI (program instruction)

Syntax

```
... V.BINARY [camera]
```

Function

Enable or disable automatic edge-image generation at VPICTURE time.

Usage Considerations

This switch may become obsolete in future versions. Its functionality is identical to performing a VEDGE instruction after a VPICTURE (*,1) 2 instruction.

A change to this switch takes effect when the next VPICTURE command or instruction is executed.

This is an array of switches—one for each virtual camera. (See the general description of switches in ***AdeptVision User's Guide*** for syntax details.)

Details

Normally, this switch is enabled.

If disabled, it will affect the operation of VPICTURE modes –1, 1, and 2, as described below.

> For VPICTURE modes 1 and 2, it will start a VEDGE operation immediately following the completed acquisition into the virtual frame buffer.
>
> For VPICTURE mode –1, a VEDGE operation is performed prior to processing of the image. In this case, the VPICTURE instruction will not complete until after the first stage of processing (the computation of run-lengths) is complete. Therefore, the run-lengths are computed on the binary edge image which is the result of VEDGE (see **Appendix B** for details on how vision run-lengths are generated).

In each case above, the choice of edge operation to be performed (cross-gradient or Sobel) is determined by the value of the system parameter V.EDGE.TYPE. And, the edge strength threshold is given by the V.EDGE.STRENGTH system parameter.

Examples

Enable binary image processing for virtual camera #1, but not for virtual camera #2:

```
ENABLE V.BINARY[1]
DISABLE V.BINARY[2]
```

Related Keywords

VAUTOTHR (monitor command and program instruction)
VPICTURE (monitor command and program instruction)
VEDGE (program instruction)
V.EDGE.STRENGTH (system parameter)
V.EDGE.TYPE (system parameter)
V.THRESHOLD (system parameter)
V.2ND.THRESH (system parameter)

Syntax

```
... V.BORDER.DIST [camera]
```

Function

Define an image border reduction (in pixels) to mask out regions clipped by the image border.

Usage Considerations

This parameter takes effect when the next VPICTURE command or instruction, or VWINDOW instruction, is executed.

This is an array of parameters—one for each virtual camera. (See the general description of parameters in the *AdeptVision User's Guide* for syntax details.)

Details

This parameter is useful for depalletizing applications where the camera sees only a portion of the pallet at one time. In that case, the four sides of the image will cut across (that is, "clip") some of the objects in the pallet into two portions—a visible portion and an invisible portion. (The four sides of the image are defined by the parameters V.FIRST.COL, V.LAST.COL, V.FIRST.LINE, and V.LAST.LINE or by an AOI definition.)

If an object has been cut by the edge of the image and more than half of the object must be visible for it to be recognized, the vision system may spend a lot of time proposing prototype matches to that object. Only a portion of an object needs to be visible in order to propose a match, but most or all of the object (depending on the prototype's verify percentage and edge weights) needs to be visible to verify the match.

V.BORDER.DIST defines an "inner border" within the "outer border" defined by the parameters V.FIRST.COL, V.LAST.COL, V.FIRST.LINE, and V.LAST.LINE or by AOI definition. Only the reduced image area (inside the inner border) is used during the match-proposal step of image processing. The inner border affects match proposals as described below.

If an object is completely outside the inner border and completely inside the outer border, a match proposal is made because the object is completely visible. That is, the inner border has no effect.

If an object is completely outside the inner border and partially outside the outer border, no match proposals are made to that object.

If an object is partially inside the inner border and partially outside the outer border, a correct match will probably still be proposed—based on features in the reduced image area. The match will be verified based on all the visible boundary of the object.

As a general rule, V.BORDER.DIST should be set to one half the image width of the object being located. You may want to experiment with different values of V.BORDER.DIST. To do that, you should enable the V.SHOW.VERIFY system switch, select a special display mode (VDISPLAY mode #3), and process an image (VPICTURE –1, VPICTURE 2 followed by VWINDOW). All the attempts to verify match proposals are displayed in the Vision display window. Then, if the vision system attempts to verify matches to objects cut off by the sides of the image, you should increase V.BORDER.DIST. However, you should not make V.BORDER.DIST so big that the system does not recognize objects that are completely visible.

This parameter must be assigned an integer value in the range 0 to 100, inclusive. The value is interpreted as the number of pixels between the outer border and the inner border. The parameter is set to 0 (that is, no border reduction will occur) when the V+ and AdeptVision systems are loaded into memory from disk.

CAUTION: V.BORDER.DIST works with orthogonal (VWINDOW) windows, but not with rotated windows. V.BORDER.DIST is ignored if the window is rotated.

Example

Figure 2-1 shows an image of a pallet containing round objects. Many of the objects have been clipped by the image border. To speed up image processing, the V.BORDER.DIST parameter was assigned the value 25. This defines an inner border that is depicted by dashed lines in the figure. Features outside the inner border are not used to propose or confirm matches. (The features may still be used to verify matches, however.)

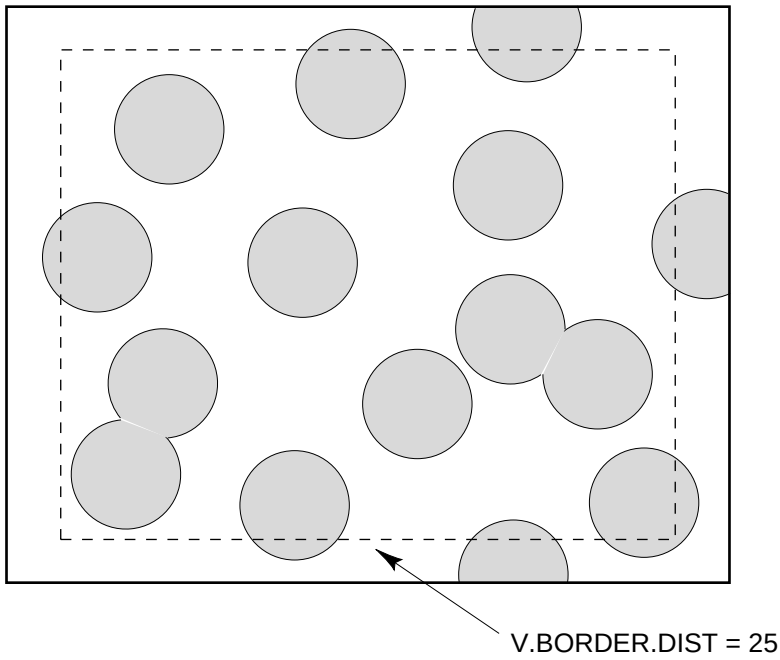


Figure 2-1. Example of V.BORDER.DIST

Related Keywords

VDEF.AOI (program instruction)
V.FIRST.COL (system parameter)
V.FIRST.LINE (system parameter)
V.LAST.COL (system parameter)
V.LAST.LINE (system parameter)

Syntax

```
... V.BOUNDARIES [camera]
```

Function

Enable or disable boundary analysis by the vision system.

Usage Considerations

A change to this switch takes effect when the next VPICTURE command or instruction, or VWINDOW instruction, is executed.

This is an array of switches—one for each virtual camera. (See the general description of switches in *AdeptVision User's Guide* for syntax details.)

Details

This switch allows the user to disable boundary analysis and, therefore, to disable the entire object recognition process when a "full" VPICTURE (type # –1 or #0) or a VWINDOW instruction is performed. Connectivity will be performed to extract regions from the image, but their centroids and moments will not be calculated. Only the more primitive region features such as area and bounding box are computed.

Example

Minimize vision processing (for all cameras):

```
DISABLE V.BOUNDARIES
```

Related Keywords

VFEATURE (program instruction)
V.FIT.ARCS (system switch)
V.RECOGNITION (system switch)
VWINDOW (program instruction)

Syntax

```
... V.CENTROID [camera]
```

Function

Enable computation of the centroid of each region in the image.

Usage Considerations

The V.BOUNDARIES system switch must also be enabled in order for centroids to be computed.

A change to this switch takes effect when the next VPICTURE command or instruction, or VWINDOW instruction, is executed.

This is an array of switches—one for each virtual camera. (See the general description of switches in the ***AdeptVision User's Guide*** for syntax details.)

Details

V.CENTROID enables the computation of the centroids of regions. When it has been computed, the centroid of a region is available, after a VLOCATE instruction has succeeded, by using the VFEATURE function.

The positions and areas of holes are not considered in the centroid computation. (The V.SUBTRACT.HOLE system switch does not affect this.) However, if you want the centroids and areas of holes, you should enable V.HOLES, disable V.DISJOINT, and do VLOCATEs in get-hole mode. If prototypes are being recognized in the image, V.CENTROID does not have to be enabled to get the *locations* of the recognized objects. Region centroids are different from recognized object locations, although both are available with the VFEATURE function. If region centroids are not needed, V.CENTROID should be disabled for efficiency. If the V.2ND.MOMENTS or V.MIN.MAX.RADII system switch is enabled, the V.CENTROID switch must also be enabled in order for the desired computation to be performed.

Example

The following program segment draws a dot at the centroid of each region in the image.

```
ENABLE V.BOUNDARIES, V.CENTROID       ;Enable necessary switches
VDISPLAY 3                            ;Special display
VPICTURE ,0                           ;Take a picture with virtual
                                      ; camera #1 and no recognition
ATTACH(vlun, 4) "GRAPHICS"            ;Attach to the vision window
FOPEN(vlun) "Vision /MAXSIZE 640 480" ; and select graphics scaling
```

```
GTRANS(vlun, 1)
; in real-world millimeters
vf.cx = 42                              ;Indexes of VFEATURE function
vf.cy = 43                              ; for centroid
VWAIT                                   ;Wait for image processing to
                                        ; complete for graphics instr.
VLOCATE() $nam                          ;Locate anything in the image

WHILE VFEATURE(1) DO                    ;If a region was found...
     cx = VFEATURE(vf.cx)               ;Get centroid: Cx,Cy
     cy = VFEATURE(vf.cy)

; Draw a dot on the region centroid

     GARC(vlun) cx, cy, 1
     VLOCATE () $nam                    ;Locate next region in image
END
```

Related Keywords

V.BOUNDARIES (system switch)
VFEATURE (real-valued function)
V.2ND.MOMENTS (system switch)
V.HOLES (system switch)
VLOCATE (program instruction)
V.MIN.MAX.RADII (system switch)
V.PERIMETER (system switch)
V.SUBTRACT.HOLE (system switch)

Syntax

```
VCONVOLVE (cam, type, dmode) dest_ibr = src_ibr
```

Function

Perform an image convolution on a grayscale frame, possibly storing the result in a different frame store.

Usage Considerations

The AdeptVision Enhanced VXL Interface option is required for convolutions larger than 3x3. On standard vision systems, only the center 3x3 section of larger convolutions is used (for both user-defined and predefined convolutions). Thus, standard systems may use larger convolutions, but the results may not be as expected (see the description of VDEF.CONVOLVE for a list of predefined convolutions).

With the AdeptVision Enhanced VXL Interface option, VCONVOLVE will execute faster.

Parameters

`cam`	Selects the threshold parameters to apply to the convolved result.
`type`	Integer value in the range 1 to 32, inclusive. A value in the range 1 to 16 performs a predefined convolution, and a value in the range 17 to 32 performs a user-defined convolution (see the description of VDEF.CONVOLVE).
`dmode`	Optional real-valued expression specifying the display mode for this operator. The choices are: –1 = no draw, 0 = erase, 1 = draw solid, 2 = complement, 3 = draw dashed, and 4 = complement dashed. The default is 1 (draw solid).
`dest_ibr`	Optional integer value specifying the image buffer region that will receive the result of the convolution. Image buffer regions specify both a virtual frame buffer and a size (see the description of VDEF.AOI).
`src_ibr`	Optional integer value specifying the image buffer region to convolve. The image buffer region's AOI must have been defined with a VDEF.AOI instruction.

NOTE: If only one image buffer region is specified, both image buffer regions default to the specified image buffer region. If neither image buffer region is specified, both default to the currently selected full frame.

The AOIs for convolutions greater than 3x3 are limited to 504 pixels in width. If an AOI wider than 504 is specified, the image buffer region is reduced based on the center of the image buffer region.

Details

A convolution is a neighborhood pixel operator that may be used to smooth an image or extract edges. The effect of a convolution is visible in VDISPLAY mode #1. As a first experiment with the VCONVOLVE instruction, type “VDISP 1” and then try each of the predefined convolutions by repeatedly doing “VPIC 2”, followed by “VCON(, n)”, with n equal to 1, 2, ..., 16.

Convolutions types 1 to 16 are predefined, and types 17 to 32 can be defined by the user. See the VDEF.CONVOLVE instruction in this manual for a description of the predefined convolutions and an explanation of how to define new ones.

The image buffer region defines the convolved area. The smaller the area of the image to be processed, the faster VCONVOLVE executes.

If the destination buffer is different from the source buffer, the destination buffer assumes some attributes of the source buffer. In particular, the virtual camera that was associated with the source buffer (when the image was acquired via VPICTURE) becomes associated with the destination buffer. This affects subsequent measurement or inspection operations on the frame, such as VRULERI or VWINDOW, because conversion of pixel units to millimeters is determined by the camera calibration.

Example

Apply a low-pass (averaging) filter to the currently selected full frame buffer:

```
VCONVOLVE (, 1)
```

Related Keywords

VDEF.AOI (program instruction)
VDEF.CONVOLVE (program instruction)

Syntax

```
VCOPY (cam, scale, dmode, lut) dest_ibr = src_ibr
```

Function

Copy the image from one image buffer region to another.

Usage Considerations

The scaling and look-up-table features require the AdeptVision Enhanced VXL Interface option.

Parameters

`cam` Optional integer value specifying a virtual camera. The V.THRESHOLD and V.2ND.THRESH values associated with the virtual camera are used when the scale parameter is 0.5 (see Details).

`scale` Optional real-valued expression specifies magnification of the copy. The acceptable values are 0.5, 1, and 2. The default is 1. This feature requires the AdeptVision Enhanced VXL Interface option.

`dmode` Optional real-valued expression specifying the display mode for this operator. The choices are: –1 = no draw, 0 = erase, 1 = draw solid, 2 = complement, 3 = draw dashed, and 4 = complement dashed. The default is 1 (draw solid).

`lut` Optional integer in the range 1 to 133 maximum (the actual maximum value is set with the DEVICE instruction) that specifies a defined “look-up table”. The table is defined with a VDEF.LUT instruction and specifies a replacement value for each possible graylevel value. See the description of VDEF.LUT. This feature requires the AdeptVision Enhanced VXL Interface option.

`dest_ibr` Integer value specifying the image buffer region to receive the copied image. Image buffer regions specify both a size and a frame store (see the description of VDEF.AOI).

`src_ibr` Integer values specifying the image buffer region to copy. The image buffer regions’s AOI must have been defined with a VDEF.AOI instruction.

NOTE: If only one image buffer region is specified, both image buffer regions default to the specified image buffer region. If neither image buffer region is specified, both default to the currently selected full frame which is selected with a VSELECT or VPICTURE instruction—for example, VPICTURE (1,,1011).

Details

VCOPY copies one image buffer region to another and optionally zooms the image. Both the grayscale and binary images are copied.

If the scale parameter is 0.5, the image is averaged with a 2x2 convolution. The result is then subsampled and thresholded using the threshold parameters for the virtual camera specified in "cam".

If the scale parameter is 2, the source frame store must be the display frame store (`src_ibr` must have the form nnn009).

After a frame is copied, the destination frame assumes some attributes of the source frame. In particular, the virtual camera that was associated with the source frame becomes associated with the destination frame. This affects subsequent measurement or inspection operations on the frame, such as VRULERI or VWINDOW, because conversion of pixel units to millimeters is determined by the camera calibration.

NOTE: When copying from one area to another within the same frame buffer, do not copy areas of interest that would overlap areas that will be copied out of later in the operation. Specifically, you cannot VCOPY a 100x100 pixel region down 10 pixels. By the time the tenth row is read out from the source area of interest, it has already been written over by line 1 of the source area of interest.

Examples

Copy one image buffer region in virtual frame store 11 to another image buffer region in the same virtual frame store:

```
VCOPY 4011 = 5011
```

Related Keywords

VDEF.AOI (program instruction)
VDEF.LUT (program instruction)
VADD (program instruction)
VSELECT (program instruction)
VSUBTRACT (program instruction)

Syntax

```
VCORRELATE (cam, mode, dmode, max_depth, accept, give_up)
                                          data[i], act_depth = tplnum, ibr

VCORRELATE  (cam, mode, dmode) data[i] = tplnum, shape,
                                                    cx, cy, dx, dy, ang
```

Function

Perform a normalized grayscale or binary correlation, comparing a predefined template with a rectangular image window, or searching for a closest match within the window to a predefined template.

Usage Considerations

Binary correlation requires the AdeptVision Enhanced VXL Interface option. Templates are defined with a VTRAIN.MODEL instruction.

Parameters

`cam` Optional real-valued expression indicating the virtual camera number to use. (This parameter is currently not used.)

`mode` Optional real-valued expression used to specify a skip pattern, as follows:

- 0 default, use the mode determined by the vision system
- 1 nonskipping mode
- 2 skip rows only
- 3 skip columns only
- 4 skip both rows and columns

`max_depth` Optional real-valued expression indicating the maximum depth of a hierarchical search, as follows:

- –1 default, go to the maximum depth possible, see the description of `act_depth`. (If the specified template is a binary template, hierarchical searches are not allowed and this parameter is ignored.)
- 0 indicates no hierarchical search.
- 1 - 4 depth of hierarchical search

`accept` This parameter specifies the minimum acceptable score for the final match. Default value is 0.8. Searching stops when this score is returned. If a match exceeding this score is not found, the best match is returned.

`give_up` If a hierarchical search is performed, this parameter specifies the minimum score that must be realized at each level of the search in order to continue searching at the next higher level. If a search does

	not find a match that exceeds this score, the search returns to the next candidate at the lowest level. Default value is 0.2.
`dmode`	Optional real-valued expression specifying the display mode for this operator. The choices are: –1 = no draw, 0 = erase, 1 = draw solid, 2 = complement, 3 = draw dashed, and 4 = complement dashed. The default is 1 (draw solid). When drawn, both the search window and the highest-scoring template position are shown.
`data[]`	Real array into which the results of the correlation are placed. Element "data[i]" receives the best correlation score, in the range 0 to 1 (where 1 is perfection). Elements "data[i+1]" and "data[i+2]" receive the X and Y coordinates (in millimeters) of the center of the template where it received the best correlation. Location accuracy is only to one pixel.
`i`	Optional array index that identifies the first element to be defined in the array "**`data[]`**". The default is 0. If a multiple-dimension array is specified, only the right-most index is incremented as the values are assigned.
`act_depth`	Optional variable that returns the actual depth of a hierarchical search. Searches will not go to the deepest level specified in `max_depth` if the template becomes too small, the search area size is too close to the template size, or the template degrades as a result of reducing.
`tplnum`	Real-valued expression specifying the correlation template number in the range 1 to 99, inclusive.
`ibr`	Optional integer value specifying the image buffer region to search for a match with a correlation template (**`tplnum`**). Image buffer regions specify both a size and a frame store (see the description of VDEF.AOI). The template size is reduced (if necessary) to a multiple of four pixels. If an image buffer region is not specified, the currently selected frame store is used. The template size is reduced (if necessary) to a multiple of four pixels. If an `ibr` is not specified, the currently selected frame store is used.
`shape`	Not currently used.
`cx, cy`	Real-valued expressions specifying the position of the center of the search window, in millimeters.
`dx, dy`	Real-valued expressions specifying the width (dx) and height (dy) of the search window, in millimeters.
`ang`	Not currently used.

Details

Normalized Grayscale Correlation (NGC) is an image-to-template comparison. More than simply an image subtraction, normalized correlation normalizes the differences, accounting for lighting or contrast changes. Additive or multiplicative changes to the image do not affect the correlation results. Binary correlation simply compares the black / white thresholded image with the binary template.

To define a template, you merely define an orthogonal rectangle in a desired image using the VTRAIN.MODEL instruction. The pixels within the image are stored in the vision CPU memory. More than one template may be defined, and they may be stored to disk for later recall (see the VSTORE and VLOAD keywords for details). The pull-down menus on the Vision display window let you list, show, delete, and rename templates.

Templates are numbered in the range 1 to 99. Some monitor commands and instructions that reference templates use a string name of the form “TMPL_n”, where “n” is the template number in the range 1 to 99.

One of the most important features is the ability to skip rows and / or columns when correlating at the most-reduced levels of hierarchy. Not performing correlation matches at every pixel location results in a much faster correlation tool. However, certain templates such as the ones with low contrast or with skinny features do not permit skipping. VTRAIN.MODEL analyzes each template during training to determine a default skip pattern and a default depth of hierarchy. The determination criteria places higher priority on robustness than speed, resulting in a more conservative default depth and skip pattern. Being conservative means a slower tool, but with less variation in execution time and less chance of not finding the object.

The search time limit for levels 0 - 4 is controlled by the V.MAX.TIME parameter. A value of 0 means that there is no time limit for the search.

The default values can be overridden by specifying the desired depth and skip pattern using input parameters “max_depth” and “mode”, respectively. However, not using the recommended depth and skip pattern may result in mismatches or increased variability of execution times.

If the depth parameter is greater than 0, a hierarchical search is performed. The image and template are first subsampled by averaging, and “mode” number of reduced images are produced. A search is then made of the most reduced image, and the location with the highest correlation match is selected. Next, the “mode – 1” reduced image is searched in close proximity to the location with the best match. This process is iterated until the final target region image is searched at full resolution. This process greatly increases the speed of correlation matching. The system will not allow a search that results in a reduced image smaller than 8x8

pixels. For example, take an AIO that is being searched with "mode" = 3. First the area of interest definition is reduced to 64x64, the template is reduced a corresponding amount, and a search is made for a match. Next a 128x128 area is searched in close proximity to the best match found in the 64x64 reduction. Finally, a search is made of the 256x256 area in close proximity to the best match found in the 128x128 reduction. (See Appendix F in the ***AdeptVision User's Guide*** for further details.)

The speed of the correlation operation is independent of the complexity of the image. The speed depends on the size of the template, the size of the search window, and the effectiveness[1] of the hierarchical search. Specifically, if the template size is "w" by "h" pixels and the window size is "W" by "H", the time spent to correlate is proportional to $f_h(w*h*(W-w+1)*(H-h+1))$. When the search window is the same size as the template, the above expression is $f_h(w*h)$, and the operation runs fairly fast.

> **NOTE:** Correlation templates provide a way to save images to CPU memory or disk. Copying a complete image to memory takes about 1 second. Storing it to disk takes about 14 seconds. Loading it from disk takes 13.5 seconds. Copying it to a frame buffer takes about 0.1 seconds.

If your controller is equipped with the AdeptVision Enhanced VXL Interface option, binary correlation may be performed using the hardware binary correlator. This executes a very fast correlation operation. The area-of-interest for binary correlation must be at least 16 pixels wider and 4 pixels higher than the template.

With 640x480 virtual frame stores, unused portions of the frame store are used to store the correlation template. This unused space is 384x480 pixels, so this is the largest template allowed. For a hierarchical search the template is limited to 384x341 pixels.

For frame stores smaller than 640x480, the template may be as large as the virtual frame store (although hierarchical correlation is limited to 512x341 pixels).

Hierarchical correlation and binary correlation require an extra "scratch" frame store. If both physical frame stores contain image data, VSELECT(mode = 1) must be used to specify which frame store to use as the scratch frame store.

[1] An "effective" hierarchical search has a "depth" that quickly identifies candidate areas. Too shallow a search will be slow to identify candidate areas. The give_up parameter can also help the effectiveness of a search by halting unproductive searches before they reach the top level.

Example

Performs a normalized correlation using template #3, and display the search window and results. The search window is centered at the point (140,260) and is 20mm x 30mm in size.

```
VDEF.AOI 3000 = 1, 140, 260, 20, 30
VCORRELATE ( ) results[] = 3, 3011
```

Related Keywords

VDEF.AOI (program instruction)
VLOAD (monitor command)
VLOAD (program instruction)
VSHOW.MODEL (program instruction)
VSELECT (program instruction)
VSTORE (monitor command)
VSTORE (program instruction)
VTRAIN.MODEL (program instruction)

Syntax

```
VDEF.AOI  aoi = shape, dim1, dim2, dim3, dim4, ang1, ang2
```

Function

Define an area-of-interest (AOI). Areas-of-interest are used by most vision tools to specify the tool placement within an image.

Usage Considerations

A special AOI (accessed as numbers 1000, 1001, or 1002) is predefined for compatibility with programs written for 10.x versions of AdeptVision AGS. This AOI refers to the entire frame store and cannot be redefined with VDEF.AOI.

Parameters

aoi	Real value, variable, or expression interpreted as an integer in the range 2000 to 200000 maximum (the actual range is specified with the DEVICE instruction) that specifies the area-of-interest being defined.
shape	Real value, variable, or expression specifying the shape of the area-of-interest. See Details below.
dim1 - dim4	Dimensions of the area-of-interest. The interpretation of these values, as well as which values are required, is determined by the shape parameter. See Details below.
ang1, ang2	Orientation or angular range of the area-of-interest. The interpretation of these values, as well as which values are required, is determined by the shape parameter. See Details below.

Details

This instruction defines the shape, size, and orientation for most vision tools. Figure 2-2 below shows the shape parameters that define rectangular or linear shaped tools. Note that an area-of-interest can be defined that includes all elements but still can be used for shapes that do not require all elements. For example, the instruction:

```
VDEF.AOI 2000 = 1, 25, 45, 20, 10, 45
```

can be used by both point finders and linear rulers. In the case of a point finder, the tool will be centered at x = 25, y = 45, with a width of 20mm, a height of 10mm, and a rotation of 45°. If the area-of-interest is used for a linear ruler, the width specification will be ignored, resulting in a ruler centered at x = 25, y = 45, with a

length of 20mm and a rotation of 45°. If you want to define an area-of-interest just for a ruler, you have to leave a place marker for the ignored dimensions. For example, the following definition could be used for linear tools but not for tools requiring a full rectangle:

```
VDEF.AOI 2000 = 1, 25, 45, 20, 0, 45
```

A shape 3 definition accepts positive or negative values for dim3 and dim4. If both these dimensions are negative, dim1 and dim2 specify the upper right corner rather than the lower left corner.

> **NOTE:** The area-of-interest widths are automatically reduced to a multiple of four pixels.

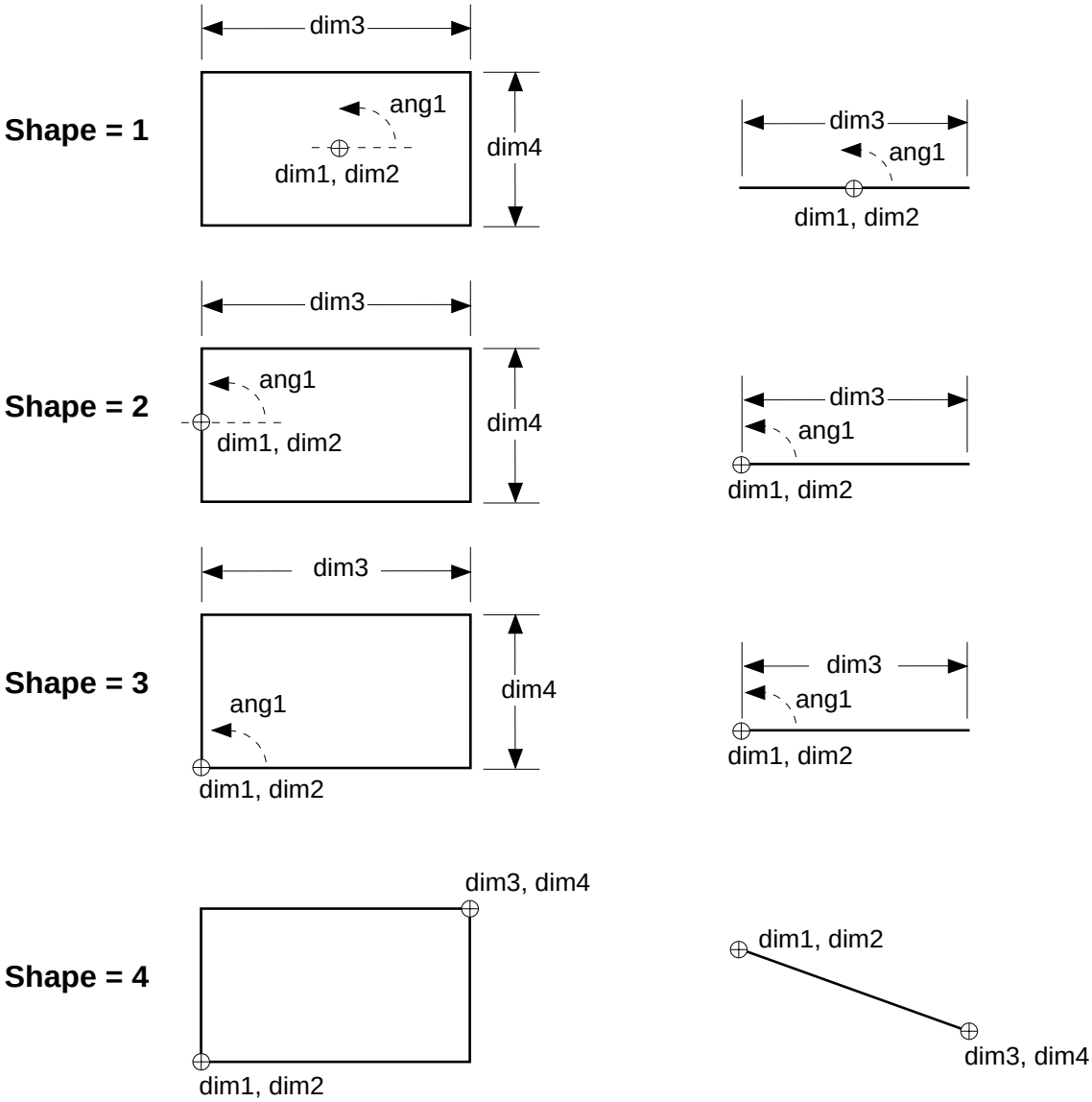


Figure 2-2. Shape Parameters for Rectangular Tools

Figure 2-3 shows the shape parameters that define arc-shaped tools. As in rectangle-shaped tools, a fully defined area-of-interest can be used by simpler shaped tools, and the appropriate dimensions are ignored. In the illustration, dim1 and dim2 always refer to the center of the arc/circle. For shape 7, a positive value for ang2 indicates a counterclockwise range from ang1 and a negative value indicates a clockwise range. The sign of ang2 also determines whether ruler tools search for edges in a clockwise or counterclockwise direction.

Shape 3 is inappropriate but allowed for finder tools. Shape 4 is not allowed for line or point finders. Shapes 9 and 10 are inappropriate but allowed with the VFIND.ARC instruction.

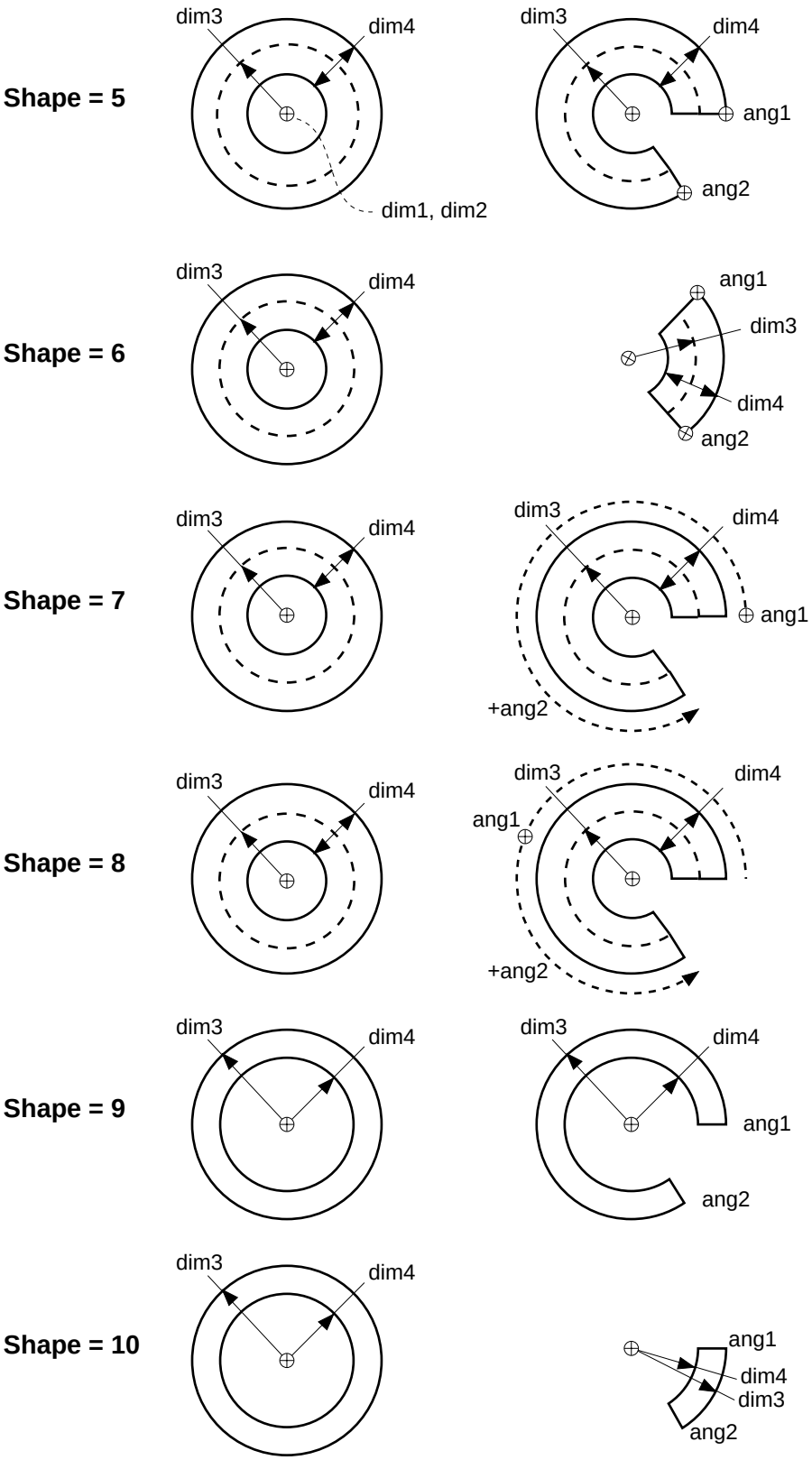


Figure 2-3. Shape Parameters for Arc-Shaped Tools

Before an AOI can be used by a vision tool, it must be combined with a virtual frame buffer number. The VDEF.AOI instruction specifies only the area-of-interest information. You must add a frame buffer specification to the area-of-interest number to create a complete "image buffer region". An image buffer region is a 4- to 6-digit value. VDEF.AOI specifies the left 1 to 3 digits and ignores the right three digits. The frame store information must be added to the right three digits to complete the image buffer region specification. This allows the same area-of-interest to be used in different virtual frame buffers. See the example below.

There are three physical frame stores in an AdeptVision VXL system. Two are for vision operations and one is for displaying vision data.[1] The two physical frame buffers used for vision operations are further divided into "virtual" frame stores. Virtual frame buffers are allocated with the V+ DEVICE command. See the *V+ Language Reference Guide* for details on DEVICE. See the *AdeptVision User's Guide* for a summary of using DEVICE to configure virtual frame stores. AOIs that extend beyond the virtual frame buffer are clipped. The form of a complete image buffer region is:

NNN	VV	P
area-of-interest number	virtual frame buffer	physical frame store

When "VVP" = 0, the currently selected virtual frame buffer is used. When "NNN" = 0, 1 is assumed (full-frame).

> **NOTE:** Instructions requiring two areas-of-interest that are the same size will reduce the larger area-of-interest about its center.

Example

```
   fs2 = 2                          ;Physical frame store 2
   fs1 = 1                          ;Physical frame store 1
   virt_fr = 2*10                   ;Virtual frame store 2

; Define two areas-of-interest

   aoi_up = 8*1000                  ;Area-of-interest 8
   aoi_dwn = 12*1000                ;Area-of-interest 12
   VDEF.AOI aoi_up = 1, 10, 15, 20, 10, 90
   VDEF.AOI aoi_dwn = 1, 10, 5, 20, 10, 90
```

[1] The special value VVP = 009 refers to the display frame store. This frame store contains the image displayed on the monitor and can be referenced only by a VCOPY instruction.

```
; Define three image buffer regions

   src_ibr1 = aoi_up+virt_fr+fs1  ;AOI 8, virtual frame 21
   src_ibr2 = aoi_dwn+virt_fr+fs2 ;AOI 12, virtual frame 22
   dest_ibr = aoi_up+virt_fr+fs2  ;AOI 8, virtual frame 22

; Add AOI 8 in virtual frame 21 (src_ibr1 = 8021) to
;  AOI 12 in virtual frame 22 (src_ibr2 = 12022). Place
;  the result in AOI 8 in virtual frame 22 (dest_ibr = 8022).

   VADD(1) dest_ibr = src_ibr1, src_ibr2
```

Related Keywords

VADD (program instruction)
VAUTOTHR (program instruction)
VCONVOLVE (program instruction)
VCOPY (program instruction)
VDEF.TRANS (program instruction)
VEDGE (program instruction)
VGET.AOI (program instruction)
VGET.TRANS (program instruction)
VHISTOGRAM (program instruction)
VMORPH (program instruction)
VPICTURE (program instruction)
VSUBTRACT (program instruction)
VCORRELATE (program instruction)
VFIND.ARC (program instruction)
VFIND.LINE (program instruction)
VFIND.POINT (program instruction)
VOCR (program instruction)
VRULERI (program instruction)
VTRAIN (program instruction)
VTRAIN.MODEL (program instruction)
VWAIT (program instruction)
VWINDOW (program instruction)
VWINDOWB (program instruction)
VWINDOWI (program instruction)

Syntax

```
VDEF.CONVOLVE    type = array[i,j]
```

Function

Define an image convolution.

Usage Considerations

VDEF.CONVOLVE is intended only for experienced users. It is not critical to the operation of the vision system.

Convolutions larger than 3x3 require the AdeptVision Enhanced VXL Interface option. When a convolution with a definition larger than 3x3 is performed by a standard vision system, only the center 3x3 elements are used in the convolution. Thus, predefined convolutions 1 to 6 will perform correctly on all systems. Convolutions 7 to 14 will run on all systems but may not return the expected results.

Parameters

type	Integer value in the range 17 to 32, inclusive. This is the number of the user-definable convolution that is being defined.
array[,]	Array of dimensions 7x7 defining a convolution. Each element in the array must be in the range –128 to +127.
i, j	Optional integers specifying the starting array index values. The default value of i and j is 0.

Details

A convolution is a neighborhood pixel operator that may be used to smooth an image or extract edges. The effect of a convolution is visible in VDISPLAY mode #1. The VCONVOLVE program instruction performs convolutions.

The convolution is applied to each pixel in the grayscale frame store.

a. The convolution is applied to a single pixel by centering the convolution on the pixel so that the 7x7 array of convolution terms overlays the 7x7-pixel neighborhood of the pixel.

b. Each term of the convolution is multiplied by the pixel value that it overlays.

c. The 49 products are summed and divided by 128.

d. The absolute value of the result is stored in the second image memory for the center pixel. The exception is in convolution #14, where it has a signed output with zero at graylevel 64.

All convolutions are 7x7 in size. However, smaller convolution sizes (such as 3x3) may be defined inside a 7x7 matrix (as shown below).

The speed of VCONVOLVE depends on the system and the kernel: On the standard system, the time is roughly 9ms for each nonzero element in the kernel. With the AdeptVision Enhanced VXL Interface option, 4x4 or smaller takes 9ms; larger kernels take 17ms. Also, the width of the AOI is limited to 508 pixels for kernels larger than 4x4.

Convolutions numbered 1 to 16 are predefined and may not be redefined by the user. The predefined convolutions are as follows:

1: 3x3 flat average

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	14	14	14	0	0
0	0	14	14	14	0	0
0	0	14	14	14	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

2: 3x3 gaussian average

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	5	15	5	0	0
0	0	15	47	15	0	0
0	0	5	15	5	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

3: 3x3 vertical edge

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	–25	0	25	0	0
0	0	–75	0	75	0	0
0	0	–25	0	25	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

4: 3x3 horizontal edge

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	25	75	25	0	0
0	0	0	0	0	0	0
0	0	–25	–75	–25	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

5: 3x3 diagonal edge

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	30	48	0	0	0
0	0	48	0	–48	0	0
0	0	0	–48	–30	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

6: 3x3 diagonal edge

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	48	30	0	0
0	0	–48	0	48	0	0
0	0	–30	–48	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

7: 5x5 flat average

0	0	0	0	0	0	0
0	5	5	5	5	5	0
0	5	5	5	5	5	0
0	5	5	5	5	5	0
0	5	5	5	5	5	0
0	5	5	5	5	5	0
0	0	0	0	0	0	0

8: 5x5 gaussian average

0	0	0	0	0	0	0
0	0	2	3	2	0	0
0	2	7	13	7	2	0
0	3	13	20	13	3	0
0	2	7	13	7	2	0
0	0	2	3	2	0	0
0	0	0	0	0	0	0

9: 7x7 vertical edge

0	–1	–1	0	1	1	0
–1	–4	–5	0	5	4	1
–4	–11	–13	0	13	11	4
–5	–14	–17	0	17	14	5
–4	–11	–13	0	13	11	4
–1	–4	–5	0	5	4	1
0	–1	–1	0	1	1	0

10: 7x7 horizontal edge

0	1	4	5	4	1	0
1	4	11	14	11	4	1
1	5	13	17	13	5	1
0	0	0	0	0	0	0
–1	–5	–13	–17	–13	–5	–1
–1	–4	–11	–14	–11	–4	–1
0	–1	–4	–5	–4	–1	0

11: 7x7 diagonal edge

0	2	3	3	1	0	0
2	6	11	10	3	0	0
3	11	17	11	0	–3	–1
3	10	11	0	–11	–10	–3
1	3	0	–11	–17	–11	–3
0	0	–3	–10	–11	–6	–2
0	0	–1	–3	–3	–2	0

12: 7x7 diagonal edge

0	0	1	3	3	2	0
0	0	3	10	11	6	2
–1	–3	0	11	17	11	3
–3	–10	–11	0	11	10	3
–3	–11	–17	–11	0	3	1
–2	–6	–11	–10	–3	0	0
0	–2	–3	–3	–1	0	0

13: 7x7 gaussian average

0	0	1	1	1	0	0
0	1	3	4	3	1	0
1	3	6	9	6	3	1
1	4	9	11	9	4	1
1	3	6	9	6	3	1
0	1	3	4	3	1	0
0	0	1	1	1	0	0

14: 7x7 Laplacian

0	1	2	3	2	1	0
1	3	6	7	6	3	1
2	6	0	–17	0	6	2
3	7	–17	–59	–17	7	3
2	6	0	–17	0	6	2
1	3	6	7	6	3	1
0	1	2	3	2	1	0

15, 16: Unity (reserved for future)

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	127	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Convolutions numbered 17 to 32 can be defined by the user. When you define a convolution, the instruction parameter "**type**" must be in the range 17 to 32 and the parameter "**array[,]**" must be defined. The arrangement of the array elements can be visualized as shown in **Figure 2-4** (the figure assumes the starting array indices are "array[1,1]").

When you are defining a convolution, the terms should add up to about 128. If both positive and negative terms are used, the sum of the positive ones and the sum of the negative ones should be close to 128 and –128, respectively. These guidelines will prevent overflow when computing the new center pixel value. If the contrast in the image is low, however, larger terms may be used.

If a convolution is defined that is not centered in the 7x7 array, it will shift the resulting image. To compensate for the shift, you should recalibrate the vision system with that convolution enabled.

If an edge convolution is used, the V.BINARY system switch should be enabled. Otherwise, the vision system will be finding edges of the edges.

When you use a convolution, the image boundaries (as defined by VDEF.AOI) should be moved into the image a pixel or two to mask out the noise generated when the convolution straddles the physical image boundary.

[1,1]	[1,2]	[1,3]	[1,4]	[1,5]	[1,6]	[1,7]
[2,1]	[2,2]	[2,3]	[2,4]	[2,5]	[2,6]	[2,7]
[3,1]	[3,2]	[3,3]	[3,4]	[3,5]	[3,6]	[3,7]
[4,1]	[4,2]	[4,3]	[4,4]	[4,5]	[4,6]	[4,7]
[5,1]	[5,2]	[5,3]	[5,4]	[5,5]	[5,6]	[5,7]
[6,1]	[6,2]	[6,3]	[6,4]	[6,5]	[6,6]	[6,7]
[7,1]	[7,2]	[7,3]	[7,4]	[7,5]	[7,6]	[7,7]

Figure 2-4. Arrangement of Elements of Convolution Matrix

NOTE: Only the innermost 3x3 elements are considered with standard vision systems.

Example

The following sequence of program instructions creates a new 3x3 low-pass filter convolution that is identical to predefined convolution number 1 ("3x3 average—flat"):

```
FOR i = 1 TO 7              ;Zero the entire 7x7 matrix
   FOR j = 1 TO 7
     vc[i,j] = 0
   END
END
FOR i = 3 TO 5              ;Define the nonzero elements
   FOR j = 3 TO 5
     vc[i,j] = 14
   END
END
VDEF.CONVOLVE 17 = vc[1,1] ;Define the convolution as #17
```

Related Keyword

VCONVOLVE (program instruction)

Syntax

VDEF.FONT (op) **font_num, $chars**, height, black_chars

Function

Define, replace, or modify an Optical Character Recognition (OCR) font.

Usage Considerations

The VISION switch must be enabled, the vision processor must be idle, and vision model training must not be active for this instruction to be executed.

Parameters

op	Optional real-valued expression defining the desired operation: 0 = define or replace a font, 1 = modify an existing font. The default is 0.
font_num	Real-valued expression defining the number of the font being defined, replaced, or modified. The number must in the range 1 to 99, inclusive.
$chars	String expression containing all of the characters in the font (in any order). See below for more information.
height	Optional real-valued expression defining the typical height (in pixels) of the tallest character in the font. The value must be in the range 6 to 63 pixels. (Note: if characters in the font are wider than they are tall, the maximum **width** should be specified instead of the height.) This parameter is required if the operation ("op") is 0; otherwise, "height" is ignored.
black_chars	Optional real-valued expression that is interpreted as a boolean (TRUE / FALSE) value. A TRUE value indicates that the characters in the font are dark against a light background. A FALSE value indicates the reverse situation. This parameter is required if the operation ("op") is 0. Otherwise, "black_chars" is ignored.

Details

This instruction defines an OCR font to the vision system. It tells the system what characters are in the font, the approximate size of the characters in the font, and the (relative) color of the characters. A font must be defined with this instruction before the vision system is trained on the font (with the VTRAIN.MODEL instruction).

If a font is already defined and VDEF.FONT is performed with "op" equal to 0, the existing font is deleted and replaced with the new definition. If "op" is 1, however, an existing font is modified, not deleted. The only modification to a font that is allowed is to add or delete characters in the font. The height and color of a font cannot be changed. When a font is modified, the given string ("$chars") defines the new set of characters. Any training experience on characters that remain in the modified font is retained. Up to 99 fonts may be defined, memory permitting. Each font is completely independent. A font may contain up to 94 characters, including symbols such as "{" and "*". All the characters must be in the ASCII range 33 ("!") to 126 ("~"). In particular, the space character (ASCII 32) is **not** permitted.

The characters defined to be in a font are actually just labels. The vision system has no built-in strategies that are character-specific. Normally, you would use "A" to label the character "A". However, OCR performance will not be degraded if you use "+" to label the character "A", and train the system as such. Then, when the vision system subsequently sees an "A" character, it will report that it saw a "+".

See the ***AdeptVision User's Guide*** for an overview of the AdeptVision OCR capability.

Example

The following instruction defines font #1 as a black set of numerals, approximately 24 pixels tall.

```
VDEF.FONT (0) 1, "0123456789", 24, TRUE
```

Related Keywords

VDELETE (monitor command and program instruction)
VOCR (program instruction)
VSHOW.MODEL (program instruction)
VTRAIN.MODEL (program instruction)

Syntax

```
VDEFGRIP  $proto, grip, mode, num_fngrs, trans[i], w[j], h[k]

VDEFGRIP  $proto, 0
```

Function

Define the shape and position of a robot gripper for clear-grip tests.

Usage Considerations

The VISION switch must be enabled **and** the vision processor must be idle for this instruction to be executed.

Parameters

$proto	String expression specifying the name of the prototype for which the grip is being defined or deleted.
grip	Real-valued expression with a value in the range 0 to 4. A gripper number of 0 deletes all defined grippers for the specified prototype. If the gripper number is in the range 1 to 4, it indicates which gripper is being newly defined for the prototype.
mode	Real-valued expression, interpreted as TRUE or FALSE, indicating how invisible fingers are to be treated. Fingers are invisible if they are outside the field of view. If the mode value is FALSE (zero), invisible fingers are not considered clear. Invisible fingers are assumed to be clear if the mode value is TRUE (nonzero).
num_fngrs	Real-valued expression with a value in the range 1 to 5, specifying the number of rectangles that are used to model the gripper fingerprints.
trans[]	Array of transformations that define the 2-D location and orientation of the rectangles. The X, Y, and RZ components of each transformation define the center position and orientation of one rectangle relative to the position of the prototype. "**num_fngrs**" number of transformations must have been stored in sequential array elements.
i, j, k	These are optional array indexes that specify the first elements to be accessed in the respective arrays. Zero is assumed for any array index that is omitted. If a multiple-dimension array is specified, only the right-most index is incremented as the values are referenced.

w[]	Array of real values defining the widths of the rectangles, in millimeters. "**num_fngrs**" number of values must have been stored in sequential array elements.
h[]	Array of real values defining the heights of the rectangles, in millimeters. "**num_fngrs**" number of values must have been stored in sequential array elements.

Details

One to four grip models may be defined per prototype. Each grip model consists of one to five rectangles in any orientation, representing the gripper fingerprints around the prototype. For simple two-finger grippers, two rectangles are adequate. Odd-shaped grippers may be approximated by defining multiple, overlapping rectangles. Grip models are automatically stored and restored with the prototype model when the VSTORE and VLOAD commands are issued. The grips defined for a prototype may be seen in the Vision display window via the VSHOW command.

The clear-grip test is automatically performed whenever a prototype is recognized and grippers are defined for it. The results of the test are available via the VFEATURE function and are shown in the Vision display window with a graphics display mode when the V.SHOW.GRIPS switch is enabled.

Grips are tested in the order of their numbering, 1 through 4. Once a grip is found to be clear, further testing is halted, because only one clear grip is required. The clear-grip algorithm checks each of the rectangles defining the grip until one is found to be not clear or until all are found to be clear. A rectangle is clear if it covers only background-colored pixels, where the background is defined by the system V.BACKLIGHT switch. As a consequence, a rectangle completely enclosed in a hole that is the color of the background is considered clear. Regions smaller than V.MIN.AREA, that merge into the background, assume the color of the background. Thus, these "noise regions" do not cause the clear-grip test to fail.

Example

In the following example, gripper fingerprints are defined for the prototype named CASTING. Two rectangles are used to model the gripper, both 30 millimeters tall and 7.5 millimeters wide. The rectangles are located plus and minus 25 millimeters from the prototype center along the X axis and 10 millimeters up from the center along the Y axis. The rectangles are oriented orthogonally with respect to the X and Y axes.

```
SET rects[0] = TRANS(25, 10, , , , 0)
SET rects[1] = TRANS(-25, 10, , , , 0)
wids[0] = 7.5
wids[1] = 7.5
hts[0] = 30
hts[1] = 30
VDEFGRIP "CASTING", 1, TRUE, 2, rects[], wids[], hts[]
```

Related Keywords

VFEATURE (real-valued function)
VSHOW (monitor command)
VSHOW (program instruction)
V.SHOW.GRIP (system switch)

Syntax

VDEF.LUT lut.num = lut[]

Function

Define a grayscale and binary "look-up table" for mapping graylevel and binary values during a VCOPY operation.

Usage Considerations

Available only with the AdeptVision Enhanced VXL Interface option.

Parameters

lut.num Integer from 1 to 133 maximum that identifies the **lut** to define.

lut[] Array of integer values in the range 0 - 255 that define the pixel mapping.

Details

The maximum "**lut_num**" is configurable using the DEVICE command. The default value is 3. The maximum is 133 LUTs, taking 33 Kb.

The "**lut[]**" is an array of 256 values (index in range 0 to 255), where each value must range from 0 to 255. The indices and the values in the array are both "pixel values". A pixel value is an 8-bit number. Bits 1 thru 7 are the graylevel value (0-127), and the binary bit is bit 8. Therefore, indices 0-127 to the array will transform the graylevels for which the binary bit is 0, and indices 128-255 will transform the graylevels for which the binary bit is 1. In other words, this is an "8 bits in, 8 bits out" LUT. The one exception is when the VCOPY is used to subsample or zoom. Then, the LUT is only "7 bits in". That is, only the indices 0-127 are used, and they are applied AFTER the subsample or zoom operation. In this case, the LUT is still "8 bits out".

Example

Define a look-up table that turns all graylevel values 50 or lower to black and sets the binary bit to black:

```
lut3 = 3
FOR x = 0 TO 50  ;Zero out indices where the binary
  lut[x] = 0     ; bit is 0 and the graylevel value
END              ; is <= 50
FOR x = 128 TO 238;Zero out indices where the binary
  lut[x] = 0     ; bit is 1 and the graylevel value
END              ; is <= 50
```

```
; Leave remaining values unchanged
   FOR x = 51 to 127
     lut[x] = x
   END
   FOR x = 239 to 255
     lut[x] = x
   END
   VDEF.LUT lut3 = lut[]
```

Related Keywords

VCOPY (program instruction)
DEVICE (program instruction)

Syntax

VDEF.MORPH (mode) **type = array[]**, dx, dy

Function

Define a binary morphological operation.

Usage Considerations

VDEF.MORPH is intended only for experienced users. It is not critical to the operation of the vision system.

The VISION switch must be enabled, the vision processor must be idle, and vision model training must not be active for this instruction to be executed.

Parameters

mode	Not currently used.
type	Integer value in the range 9 to 16, inclusive. This is the number of the user-definable morphological operation that is being defined.
array[]	Array of length 512 that defines the morphological operation. Each element in the array must have a zero or nonzero value.
dx, dy	Not currently used.

Details

Binary morphological operations are nonlinear transformations of binary (or edge) images. The VDEF.MORPH instruction defines the operation to perform. The VMORPH instruction actually performs the operation.

Morphological operations may be used to eliminate small holes and gaps from the image (by dilating and then eroding or vice versa), to thin edges, isolate certain features such as straight lines, etc. Multiple operations are often performed in sequence. Morphological operation types 1 and 2 are predefined as erosion and dilation. Types 3 through 8 are undefined but reserved to be additional built-in operations in the future. Types 9 through 16 can be defined by the user. (Type 9 is actually predefined to be the game of Life, popularized years ago in *Scientific American,* but it can be redefined by the user.) Binary morphological operations are applied to every 3x3 pixel neighborhood in the binary image. Based on the binary pixel values in a neighborhood and the operation to be performed, the center binary pixel value may be changed. The parameter “array[ ]”, which

defines the operation, actually defines a look-up table. To index into the table, the vision system constructs the nine bits of the index from a 3x3 pixel neighborhood. The binary values of the nine pixels in a neighborhood about the center pixel "5" are as follows:

9	8	7
6	5	4
3	2	1

Each of the pixel values 1 through 9 has the value 0 or 1. The 9 bits are combined into a single number, ordered 987654321 ("1" is the least-significant bit). For instance, consider a 3x3 binary neighborhood in the image as follows:

1	1	1
1	1	0
1	0	0

When the above is combined into a single number, the value in binary format would be 111110100, which is 500 decimal.

Now, the VMORPH instruction operates as follows: Let the 9-bit value for a neighborhood be called "neigh". If "array[neigh]" is not 0, the center pixel of the neighborhood is complemented. Otherwise, the center pixel is not changed.

Example

The following sequence of program instructions defines a simple morphological operation to extract the boundary from a binary image. The operation simply zeros binary pixels when the neighborhood is uniformly white (all 1s) and leaves all other pixels unchanged. After executing this program, the operation can be performed on an image by executing "VMORPH (10)".

```
FOR i = 0 TO 510          ;Zero almost the entire array
  morph[i] = 0
END
morph[511] = 1            ;This neighborhood causes a
                          ; change
VDEF.MORPH 10 = morph[]   ;Define type 10
```

Related Keyword

VMORPH (program instruction)

Syntax

```
VDEF.SUBPROTO   proto:subname, first_edge, last_edge
```

Function

Define a subprototype.

Parameters

proto
subname — "**proto**" is the name of the prototype of which the new subprototype is to be a portion. "**subname**" is the name of the subprototype being defined. For the monitor command, these must be specified by an unquoted string (see the example below). For the program instruction, they must be specified by a string constant, variable, or expression containing the ":" separator (see the example below).

first_edge — Real-valued expression that specifies the edge number of the prototype that is to be the first edge of the subprototype.

last_edge — Real-valued expression that specifies the edge number of the prototype that is to be the last edge of the subprototype. Note that "**last_edge**" may be less than "**first_edge**".

Details

This instruction defines a subprototype of the prototype "**proto**" to be the series of edges from "**first_edge**" to "**last_edge**". The subprototype must consist of edges from only one region of the subprototype. Thus, for example, two holes cannot be grouped together to form a single subprototype. If "**first_edge**" equals "**last_edge**", the subprototype consists of a single edge.

Once the subprototype is defined, it has its own edge numbering. Thus, edge "**first_edge**" of "**proto**" is edge #1 of the subprototype.

A subprototype remains associated with a prototype until the subprototype or the prototype is VDELETEd. A subprototype is automatically saved and restored when its associated prototype is VSTOREd and VLOADed. Subprototypes may also be referenced by VEDGE.INFO, VGAPS, VSHOW, and VSUBPROTO. The maximum number of prototypes and subprototypes (counted together) that may be loaded in the system at one time is 25. Of course, an unlimited number may be stored on disk.

Examples

The following examples compare the VDEF.SUBPROTO monitor command with the program instruction. If VDEF.SUBPROTO is performed as a monitor command, “proto:subname” must be a nonquoted string (like the VSHOW parameter). If VDEF.SUBPROTO is used as a program instruction, the prototype must be specified with a string expression.

```
VDEF.SUBPROTO CASTING:TOP, 3, 9        Monitor command

VDEF.SUBPROTO "CASTING:TOP", 3, 9      Program instruction
```

Related Keywords

V.EDGE.INFO (program instruction)
VGAPS (program instruction)
VSHOW (monitor command)
VSHOW (program instruction)
VSUBPROTO (program instruction)

Syntax

VDEF.TRANS (mode) dx, dy, angle, scale

Function

Define a transformation to apply to the location of all vision tools placed until the next VDEF.TRANS instruction.

Usage Considerations

The transformation applies only to the task in which it is defined. Thus, each task that requires a transformation must execute a VDEF.TRANS instruction for that task.

This transformation applies to all tools regardless of whether they use the area of interest definition syntax or explicitly include position information.

Parameters

mode	Optional argument that must currently be set to 1. The default value is 1.
dx, dy	Optional real values, variables, or expressions interpreted as millimeter offsets in the X,Y Cartesian plane. The default value is 0.
angle	Optional real value, variable, or expression interpreted as an angular rotation. The default value is 0.
scale	Optional real value, variable, or expression interpreted as an integer that scales the dimensions of the tool. The default is 1.

Details

The primary use of VDEF.TRANS is to place inspection tools relative to a frame or object previously located by the vision system.

All vision tools (regardless of syntax used) placed after a VDEF.TRANS instruction has been issued will have the indicated transformation applied before the tool is placed. Issuing a VDEF.TRANS instruction with no arguments (defaults of dx, dy, and angle = 0) will cancel the vision transformation.

Examples

```
; Rotate a ruler tool perpendicular to the angle returned by a
; line finder. "ibr_rect" has been defined for rectangular tools.
    VFIND.LINE (1) data[] = ibr_rect      ;Find the line
    ang = data[4]+90                      ;Angle for perpen. rotation
; Define the transformation
```

```
     VDEF.TRANS ,, ang
; Place the ruler
     VRULERI (1) rdata[] = ibr_rect
     VDEF.TRANS                              ;Cancel transformation

; Place an inspection window relative to a found blob
     ENABLE V.CENTROID                       ;Get centroid information
     VDEF.AOI win.aoi = 1, 10, -40           ;Postion of window relative
                                             ; to blob.
     VPICTURE (1)
     VLOCATE (1,2)"?"                        ;Locate a "blob"
     IF VFEATURE(1) THEN                     ;Make sure a blob was found
          cx = VFEATURE(42)
          cy = VFEATURE(43)
          ang = VFEATURE(48)
          VDEF.TRANS cx, cy, ang
          VWINDOWI wdata[] = win.aoi
     END
     VDEF.TRANS                              ;Cancel transformation
```

Related Keywords

VDEF.AOI (program instruction)
VGET.AOI (program instruction)
VGET.TRANS (program instruction)

Syntax

VDELETE model_name

Function

Delete a specified prototype, subprototype, Optical Character Recognition (OCR) font, or correlation template in the vision system.

Usage Considerations

The VISION switch must be enabled *and* the vision processor must be idle for this command or instruction to be executed.

Parameters

model_name Name of the prototype, subprototype, OCR font, or correlation template to be deleted.

If a subprototype is being deleted, the model name has the form "name1:name2", where "name1" and "name2" are the names of the prototype and its subprototype, respectively.

If a font is being deleted, the model name has the form "FONT_n", where "n" is the number of the font in the range 1 to 99.

If a template is being deleted, the model name has the form "TMPL_n", where "n" is the number of the template in the range 1 to 99.

For the monitor command, this parameter must be a string constant (**not** surrounded by quotes). For the program instruction, however, the name can be specified with a string constant (including quotes) or an expression.

Details

The type of model being supported is determined by the name, which has been trained as either a prototype or an ObjectFinder model. The name first is checked against the lists of prototypes for backwards compatibility. Therefore, you should not use the same name for a prototype model and an ObjectFinder model.

VDELETE deletes a prototype that has been trained or loaded into the vision system. It does not affect a prototype that has been stored in a disk file.

For the VDELETE monitor command, the prototype or subprototype is displayed in the Vision display window, and the operator is asked for confirmation before it is deleted.

VDELETE deletes OCR fonts and correlation templates in a similar manner.

Examples

Note that when VDELETE is used as a monitor command (as in the first example below), the model name is **not** enclosed in quotes. When VDELETE is used as a program instruction, however, quotes must be included if a string value is specified.

Monitor command to delete the subprototype “fillet” associated with the prototype “casting”:

```
VDELETE casting:fillet
```

Program instruction to delete the model named by the string variable “$model”:

```
VDELETE $model
```

Program instruction to delete font #2:

```
VDELETE "FONT_2"
```

Program instruction to delete template #1:

```
VDELETE "TMPL_1"
```

Related Keywords

VDEF.FONT (program instruction)
VDEF.SUBPROTO (monitor command and program instruction)
VLOAD (monitor command)
VLOAD (program instruction)
VTRAIN (monitor command and program instruction)
VTRAIN.MODEL (program instruction)

Syntax

```
... V.DISJOINT [camera]
```

Function

Determine whether or not prototypes may be matched to multiple disjoint regions.

Usage Considerations

A change to this switch takes effect when the next VPICTURE command or instruction, or VWINDOW instruction, is executed.

When this switch is enabled, the V.HOLES switch has its effect (but not its setting) disabled.

This is an array of switches—one for each virtual camera. (See the general description of switches in the ***AdeptVision User's Guide*** for syntax details.)

Details

If V.DISJOINT is enabled, the system finishes processing the boundaries of all regions in the image before attempting object recognition. Disjoint regions in the image may then contribute to the recognition of a single prototype.

When V.DISJOINT is disabled, each region in the image is processed to completion (including all recognition attempts) before processing of the next region is begun.

When scenes consist of multiple objects and V.DISJOINT is enabled, the vision system uses more memory and may take more time to process the image.

The V.DISJOINT and V.TOUCHING switches affect the interpretation of the "how_many" parameter to the VPICTURE and VWINDOW instructions. That parameter specifies the maximum number of objects the vision system should try to recognize. V.DISJOINT affects how the "how_many" parameter applies to the image as a whole, whereas V.TOUCHING affects how it applies to each region in the image. If V.DISJOINT is enabled, the number of objects specified by "how_many" is the most that will be recognized in the entire image. If V.DISJOINT is disabled, there is no limit on how many objects will be recognized in the entire image.

The following table summarizes the relationship between V.DISJOINT, V.TOUCHING, and the “how_many” parameter to VPICTURE and VWINDOW.

V.TOUCHING	V.DISJOINT	Number of Objects per Region	Number of Objects per Scene
Off	Off	1	No limit
Off	On	1	how_many
On	Off	how_many	No limit
On	On	how_many	how_many

Example

The two images in **Figure 2-5** illustrate an object (a fork) that appears to the vision system to be split into two parts. Thus, the V.DISJOINT switch must be enabled to recognize the object.

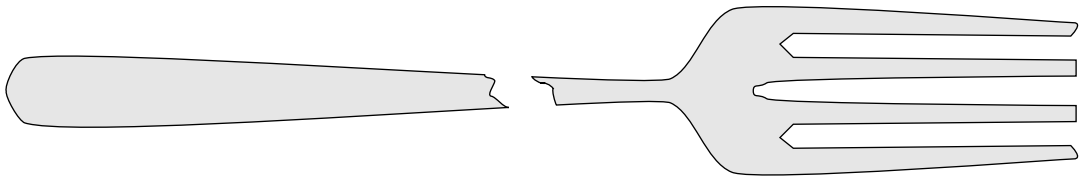
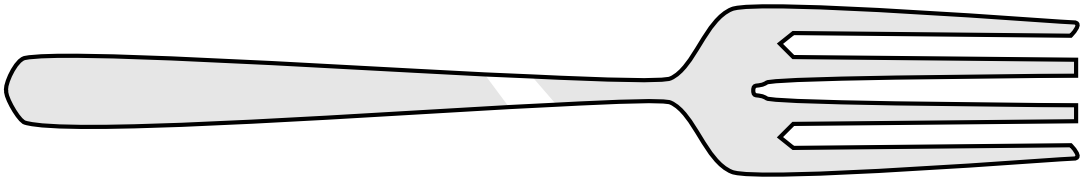


Image of fork split into two regions (V.SHOW.BOUNDS enabled)



Recognition of fork with V.DISJOINT and V.SHOW.RECOG enabled

Figure 2-5. Effect of V.DISJOINT Switch

Related Keyword

V.BOUNDARIES (system switch)
V.HOLES (system switch)
V.TOUCHING (system switch)

Syntax

VDISPLAY (camera) **mode**, overlay, xpan, ypan, zoom

Function

Select the current vision display mode or the display mode to be used when the vision system performs its normal image processing functions.

Usage Considerations

The VDISPLAY monitor command may be used when a program is executing.

Vision model training must not be active for this instruction to execute.

Parameter

`camera` Optional real-valued expression that specifies the virtual camera number. The special camera number 0 specifies all virtual cameras.

This parameter defaults to 0 (all virtual cameras) if no value is specified. In that case, the parentheses can be omitted.

`mode` Real-valued expression that specifies the mode of operation to be used. The available modes are listed here but described below.

–1	Grayscale camera output (live grayscale)
0	Thresholded camera output (live binary)
1	Frame-grabbed grayscale
2	Frame-grabbed edge or binary
3	Special display (graphics only)
4	Static graphics (graphics only)
5	No display (no alter)

`overlay` Optional real-valued expression that specifies whether or not to overlay the graphics memory on the image to be displayed. Overlay values are 0 for no overlay, 1 for overlay, and 2 for static overlay. See Details below.

`xpan` Optional real value, variable, or expression in the range 0 to 1023 specifying a pan in the x direction. The default is 0.

`ypan` Optional real value, variable, or expression in the range 0 to 511 specifying a pan in the y direction. The default is 0.

`zoom` Optional real value, variable, or expression indicating the magnification of the displayed image. Acceptable values are 0.5, 1, 2, and 4. The default is 1.

The zoom argument is encoded with a resolution, as follows:

zoom = zoom factor + resolution * 1000

Resolution	Description
0	No change to the resolution
1	Full resolution
2	Half resolution
3	Quarter resolution
4	Focus mode

Details

Modes are set and remembered individually for each virtual camera (or for all virtual cameras if "camera" is 0). In all modes, VDISPLAY has immediate effect if the vision system is not actively processing an image. Otherwise, the VDISPLAY mode will go into effect the next time a VPICTURE command or instruction is executed. The following describes the vision display modes.

Mode –1 **Grayscale camera output.** The live camera image is displayed in the Vision display window. This is useful for positioning the camera, focusing, and setting the aperture. Also note that the system parameter settings V.GAIN and V.OFFSET affect the live camera image displayed. This is the initial display mode.

NOTE: If no virtual camera number is specified with modes # –1 and #0, the image from the camera most recently referenced is displayed.

Mode 0 **Thresholded camera output.** The Vision display window shows the live thresholded output of the camera. This mode is primarily useful for setting the threshold for binary image processing. The effects of the parameters V.THRESHOLD and V.2ND.THRESH are both visible. (See the Note above.)

Mode 1 **Frame-grabbed grayscale.** With this display mode, one of the two grayscale frame stores is displayed. After a VPICTURE operation, the frame acquired is displayed. Otherwise, VSELECT may be used to select the frame to be displayed.

Mode 2 **Frame-grabbed edge or binary.** With this mode, one of the two binary-edge frame stores is displayed. Depending on whether the

V.BINARY system switch was enabled or not when the VPICTURE was last executed, this display mode shows a frozen binary or edge image. The VSELECT instruction may be used to select the frame to be displayed.

Mode 3 **Special display.** This is the graphics display mode. All user graphics, histograms, VSHOW of prototypes, training images, and the special display graphics are visible in this mode.

In response to VPICTURE (or VWINDOW), the objects in the camera view are graphically depicted in the Vision display window. The image looks like display mode #2 except that only processed regions are shown. Unprocessed regions, those not shown, are smaller than V.MIN.AREA, larger than V.MAX.AREA, outside the image bounds V.FIRST.COL, V.LAST.COL, etc.

In addition to the regions shown, image boundaries, prototype boundaries, or clear-grip positions are displayed depending on what switches are enabled. The relevant system switches are V.SHOW.BOUNDS, V.SHOW.EDGES, V.SHOW.VERIFY, V.SHOW.RECOG and V.SHOW.GRIP. Note that if too many of these display switches are enabled, the resulting screen image may become muddled.

The VPICTURE instruction and monitor command automatically erases the vision graphics window when the display mode is #3 or overlay is in effect. The erasure occurs simultaneously with the acquire and any subsequent processing.

Mode 4 **Static graphics.** This is a graphics display mode like #3, except that the vision display window is not automatically erased with each VPICTURE. Further, the vision system does not automatically draw graphics when operations such as VPICTURE, VRULER, and VWINDOW are performed. Only the graphics instructions (such as GCLEAR and GLINE) affect the vision display window. Thus, with this display mode, the user has complete control over the graphics overlay image. For example, statistical process control information can be displayed continuously.

Mode 5 **No display or no alter.** This mode is for high-speed, multiple-camera applications. A camera number should be specified along with this display mode. Then, when a VPICTURE is performed using that camera, the current display mode is not changed. For example, if image from virtual camera #2 is being displayed in binary mode (#2), the image from virtual camera #1 is being displayed in no alter mode (#5), and a VPICTURE with virtual

camera #1 is issued, the image from virtual camera #2 will continue to be displayed in binary mode.

If camera = 0 (all virtual cameras), multiple VCOPY operations can be performed to the display frame store allowing you to create your own image (high-end systems only).

The "overlay" parameter to VDISPLAY may have the value 0, 1, or 2. The "overlay" value 0 (which is assumed when the system is initialized) disables graphics overlay. A value of 1 or 2 enables the overlay of the graphics memory over the image to be displayed, as selected by the VDISPLAY mode (# –1, #0, #1, or #2).

The source of the color overlay is the graphics frame store that is visible in VDISPLAY mode #3. Colors 1 through 16 are displayed as overlay colors.

An "overlay" value of 1 is like VDISPLAY mode #3. The vision display window is automatically erased when a VPICTURE operation is performed and is automatically drawn into when instructions such as VRULER and VWINDOW are executed. See the description of mode #3 above, particularly that regarding the erasure time.

An "overlay" value of 2 is like VDISPLAY mode #4. That is, the vision display window is static—modified only with user graphics instructions.

When acquiring images in single field mode, a "zoom" value of 4 is not allowed. (The image is already horizontally zoomed by 2x so it will display correctly.)

The x and y pan values allow you to scan the entire frame store, regardless of which areas have valid image data. A single frame-grabbed image uses 640 x 480 pixels leaving the remaining 1024 x 512 frame buffer untouched. However, operations such as VCOPY and VCONVOLVE may write to any portion of the frame store. Correlation templates are also stored in unused portions of the frame store.

Examples

Turn on special display mode for virtual camera 4:

```
VDISPLAY (4) 3
```

Enable live video input with static overlay:

```
VDISPLAY –1, 2
```

Turn on live thresholded mode for all virtual cameras:

```
VDISPLAY
```

Related Keywords

VDEF.AOI (program instruction)
VPICTURE (monitor command and program instruction)
VSELECT (program instruction)
V.SHOW.BOUNDS (system switch)
V.SHOW.EDGES (system switch)
V.SHOW.GRIP (system switch)
V.SHOW.RECOG (system switch)
V.SHOW.VERIFY (system switch)

Syntax

```
... V.DRY.RUN
```

Function

Enable graphics-only mode for various vision operators.

Usage Considerations

A change to this switch takes effect when the next vision operator is executed.

Details

This switch is for use by V+ programs that wish to display a vision operation, such as VFIND.LINE in the Vision display window, and move it around without it actually fitting lines and showing results. For example, VisionWare uses this option extensively. The setting of V.DRY.RUN affects all instructions that use an image buffer region.

When V.DRY.RUN is enabled, tool graphics are displayed (if the tool's display mode is set appropriately), but the operation is not actually performed.

Example

Enter graphics-only mode:

```
ENABLE V.DRY.RUN
```

Syntax

VEDGE (cam, type, dmode) dest_ibr = **src_ibr**

Function

Compute edges in the grayscale image and threshold the edges, replacing the binary image, using either a cross-gradient or Sobel algorithm.

Parameters

cam	Optional integer value specifying a virtual camera. The value of V.EDGE.STRENGTH associated with the virtual camera will be used to compute edges. The default is 1.
type	Optional integer value indicating the type of edge algorithm to use: 1 for cross-gradient and 2 for Sobel. The default is 1 (cross-gradient).
dmode	Optional real-valued expression specifying the display mode for this operator. The choices are: –1 = no draw, 0 = erase, 1 = draw solid, 2 = complement, 3 = draw dashed, and 4 = complement dashed. The default is 1 (draw solid).
dest_ibr	Optional integer value specifying the image buffer region to receive the edge image. Image buffer regions specify both a size and a frame store (see the description of VDEF.AOI). If this parameter is not specified, the edge image will overwrite the source image (**src_ibr**).
src_ibr	Integer value specifying the image buffer region from which to extract edges. The image buffer region's AOI must have been defined with a VDEF.AOI instruction.

Details

This instruction computes edges from the grayscale values in an image and thresholds the edges. Only the binary image associated with "dest_ibr" is affected. If "dest_ibr" is not specified, "**src_ibr**" is used as the destination. That is, the edge operation is performed "in place".

The effect of a VEDGE operation is visible in VDISPLAY mode #2. It does not affect the associated grayscale image. The "type" parameter to VEDGE determines the method of edge extraction. The cross-gradient operator is faster, but the Sobel is a bit more robust. See the entry for the system parameter V.EDGE.TYPE on page 94 for a detailed description of the cross-gradient and Sobel methods.

The smaller the area of the image to be processed, the faster the operation executes. The cross-gradient operator is faster than the Sobel operator.

Example

Compute cross-gradient edges for all of frame store #1 using 22 as the edge strength:

```
PARAMETER V.EDGE.STRENGTH[1] = 22
VEDGE (1, 1) 1001
```

Related Keywords

VTHRESHOLD (program instruction)
V.EDGE.TYPE (system parameter)

Syntax

VEDGE.INFO data[i] = proto_nam**, edge_num**

Function

Retrieve information about the edges and corners of a prototype or of a region in the image.

Usage Considerations

The V.EDGE.INFO switch must be enabled before a VPICTURE (in modes 0 or –1) or VWINDOW instruction is executed in order to use VEDGE.INFO to retrieve information about regions in the image. Also, if V.DISJOINT is true, edge information is not available for regions that helped verify the recognition of a prototype.

Vision model training must be inactive for this instruction to execute.

Parameters

data[] Real array containing the requested edge information:

data[i+0] = 0 (line), +1 (convex arc), or –1 (concave arc)
data[i+1] = Weight: 0 to 100 (for prototype edges only)
data[i+2] = X coordinate of edge starting corner
data[i+3] = Y " " "
data[i+4] = X coordinate of edge ending corner
data[i+5] = Y " " "
data[i+6] = X coordinate of arc circle center
data[i+7] = Y " " "
data[i+8] = Radius of arc circle

Coordinates and radii are in millimeters. For line edges, elements "data[i+6]" through "data[i+8]" are not defined.

i Optional integer value that identifies the first array element to be defined in "**data[]**". Zero is assumed if the index is omitted. If a multiple-dimension array is specified, only the right-most index is incremented as the values are assigned.

proto_nam String expression that specifies the name of a prototype, a subproto-type, or "?". If a subprototype is specified, the string must have the form "name1:name2", where "name1" is the name of a prototype and "name2" is the name of the prototype's subprototype. If edge information about a region in the image (that is, not a prototype) is being requested, this parameter should not be specified (defaulted) or it should be "?".

edge_num Real-valued expression that specifies the number of the edge of the region, prototype, or subprototype for which information is requested.

Details

Knowing the positions and lengths of prototype edges could be useful in a V+ program that automatically places VFIND.LINE and VFIND.ARC operators on selected edges of prototypes to determine edge positions with high accuracy. The operators would be applied after prototype recognition.

The instruction VEDGE.INFO retrieves information about the edges and (indirectly) the corners of a prototype. The edges of a prototype are numbered starting at 1. VSHOW can be used to display a prototype with its edges numbered. These are the edge numbers used with VEDGE.INFO to reference the edges.

Each subprototype has its own edge numbering, starting with 1. Again, VSHOW can be used to display a subprototype with its edges numbered.

To use VEDGE.INFO to get information about a region, the V.EDGE.INFO switch must be enabled before the VPICTURE or VWINDOW instruction is executed. Then the region must be VLOCATEd. A second VPICTURE or VWINDOW operation deletes all edge information from the previous VPICTURE (or VWINDOW) operation, even if a different virtual camera number is specified. (When a different virtual camera is specified, objects from the previous VPICTURE may still be VLOCATEd, but no edge information is available.)

After a region has been VLOCATEd, VFEATURE item 27 provides the number of edges bounding the region, not counting holes. To get information about holes, follow the same procedure for getting VFEATURE information about holes.

If V.DISJOINT is true, VEDGE.INFO does not return edge information for regions that helped verify the recognition of a prototype.

Example

```
.PROGRAM display_edges($proto)

;ABSTRACT:    This program displays, in the Monitor display window,
;    information about the edges of a prototype or subprototype.
;    The name of the prototype or subprototype is passed to the
;    program when it is called.

;INPUT PARAM: $proto - name of the prototype of interest
         LOCAL data[], edge_num, num_edges, ocorn[]
         LOCAL $type
```

```
; Display the prototype or subprototype and use VFEATURE
; to determine the number of edges it has

          VSHOW 2, $proto, , , -1
          IF NOT VFEATURE(1) THEN
               TYPE $proto, " is not a known (sub)prototype"
               GOTO 100
          END
          num_edges = VFEATURE(vf.num.bounds)
          ocorn[1] = 0            ;Need some initial values
          ocorn[2] = 0

; For each edge of the prototype, use VEDGE.INFO to
; retrieve information about the edge

          FOR edge_num = 1 TO num_edges
               VEDGE.INFO data[] = $proto, edge_num
               IF (ocorn[1] <> data[2]) OR (ocorn[2] <> data[3]) THEN
                    IF edge_num <> 1 THEN
                         TYPE " *** New region or edge gap ***"
                    END
                    TYPE " Corner: ", data[2], ",", /I0, data[3]
               END
               IF data[0] == 0 THEN         ;It's a LINE
                    TYPE edge_num, ") Line. Wt: ", data[1]
               ELSE                         ;It's an ARC
                    IF data[0] > 0 THEN
                         $type "Convex"
                    ELSE
                         $type "Concave"
                    END
                    TYPE edge_num, ") ", $type, " arc.", /S
                    TYPE " Center: ", data[6], ",", /I0, data[7], /S
                    TYPE ", Radius: ", data[8], /S
                    TYPE ", Wt: ", data[1]
               END
               TYPE " Corner: ", data[4], ",", /I0, data[5]
               ocorn[1] = data[4]      ;Record endpoint of this edge
               ocorn[2] = data[5]
          END
100 RETURN
.END
```

Related Keywords

VSHOW (monitor command)
VSHOW (program instruction)
V.EDGE.INFO (system switch)

Syntax

```
... V.EDGE.INFO [camera]
```

Function

Enable saving of information about edges in the image for recall via the VEDGE.INFO instruction.

Usage Considerations

The V.BOUNDARIES system switch must be enabled in order for edge information to be computed. Also, the V.FIT.ARCS switch must be enabled if circular arcs are to be fit in addition to lines.

A change to this switch takes effect when the next VPICTURE command or instruction, or VWINDOW instruction, is executed.

This is an array of switches—one for each virtual camera. (See the general description of switches in the *AdeptVision User's Guide* for syntax details.)

Details

When regions are analyzed and arcs and lines are fit to the boundaries of the regions, the information is stored away if V.EDGE.INFO is enabled. The instruction VEDGE.INFO may then be used to retrieve descriptions of the lines and arcs that were fit.

If the V.FIT.ARCS system switch is disabled, no arcs will be fit. Furthermore, if the V.MAX.PIXEL.VAR system parameter is 0, neither lines nor arcs will be fit to region boundaries. In that case, the VEDGE.INFO instruction will return the primitive edges that bound the regions. The time spent by the vision system fitting lines and arcs will be saved, but usually many more primitive edges will then be present. VEDGE.INFO must then be executed more times to retrieve the entire boundary description.

Additional memory and some extra processing time is spent storing the edge information, so if the information is not needed, this switch should be disabled. This switch is disabled when the V$^+$ and AdeptVision systems are loaded into memory from disk.

Example

Save edge info for virtual camera #1:

```
ENABLE V.EDGE.INFO[1]
```

Related Keyword

VEDGE.INFO (program instruction)

Syntax

```
... V.EDGE.STRENGTH  [camera]
```

Function

Set the edge threshold for grayscale image processing and fine-edge rulers.

Usage Considerations

A change to this parameter takes effect when the next VPICTURE, VEDGE, VRULER, VWINDOWI, VFIND.LINE, VFIND.POINT, or VFIND.ARC instruction is executed.

This is an array of parameters—one for each virtual camera. (See the general description of parameters in the *AdeptVision User's Guide* for syntax details.)

Details

This parameter is used by various vision functions, the first of which is to threshold the edges during grayscale image processing. Grayscale image processing occurs in response to a VEDGE instruction or a VPICTURE request when the V.BINARY system switch is disabled. With grayscale edge processing, edges are extracted from the image based on local gradients (local changes in image intensity). The V.EDGE.TYPE parameter determines the edge operator used: cross gradient or Sobel. The V.EDGE.STRENGTH parameter is the edge threshold. If the edge magnitude in the neighborhood of a pixel is greater than V.EDGE.STRENGTH, the pixel is considered white. Otherwise, the pixel is considered black. The resulting image consists of regions of intensity change.

V.EDGE.STRENGTH is also used as an edge-sensitivity threshold for the inspection operators: fine-edge ruler, type #5 VWINDOWI, VFIND.LINE, VFIND.POINT, and VFIND.ARC. For example, consider fine-edge rulers. As the grayscale pixels along the ruler are scanned, the gradients in the neighborhood of each pixel are computed. Then, if the maximum gradient in the neighborhood is greater than V.EDGE.STRENGTH, a search is started for the center of the edge along the ruler. The position of the edge center, computed with subpixel accuracy via interpolation, is then reported by VRULER as the edge point.

As the value of V.EDGE.STRENGTH is lowered, more edge points will appear in the image and will be reported by VRULER. On the other hand, increasing V.EDGE.STRENGTH will reduce the number of edge points. To determine the best value of V.EDGE.STRENGTH, disable the V.BINARY system switch and enter display mode #2 so that the edges are visible in the Vision display window. Then, take a series of pictures (each with "VPICTURE (cam) 2"), with a different value of V.EDGE.STRENGTH for each picture.

This parameter must be assigned an integer value in the range 0 to 127, inclusive. The parameter is set to 20 when the V+ and AdeptVision systems are loaded into memory from disk.

Example

Use 15 as the edge threshold for all cameras:

```
PARAMETER V.EDGE.STRENGTH = 15
```

Related Keywords

V.BINARY (system switch)
VEDGE (program instruction)
VFIND.ARC (program instruction)
VFIND.LINE (program instruction)
VFIND.POINT (program instruction)
VPICTURE (program instruction)
VRULERI (program instruction)
VWINDOWI (program instruction)
V.BINARY (system parameter)
V.EDGE.TYPE (system parameter)

Syntax

```
... V.EDGE.TYPE [camera]
```

Function

Determine the type of edge operator to use, cross-gradient or Sobel, when a VPICTURE instruction is performed.

Usage Considerations

A change to this parameter takes effect when the next VPICTURE instruction is executed.

This is an array of parameters—one for each virtual camera. (See the general description of parameters in this manual for syntax details.)

Details

This parameter determines the method of edge extraction for VPICTURE when the V.BINARY system switch is disabled. If the V.EDGE.TYPE parameter value is 1, a cross-gradient operator is applied to the grayscale image. Otherwise, if the parameter value is 2, the Sobel operator is used. The parameter is set to 1, for cross-gradient, when the V^{+} and AdeptVision systems are loaded into memory from disk.

The cross-gradient operator is faster, but the Sobel operator is a bit more robust. The V.EDGE.STRENGTH system parameter is the threshold for both operators.

The following details are only for the curious—they are not required for use. Both operators are applied to every 3x3 neighborhood in the image window defined by the system parameters V.FIRST.COL, V.LAST.COL, V.FIRST.LINE, and V.LAST.LINE. Let the 9 grayscale pixels in a 3x3 neighborhood be labeled “a” through “i”:

a b c

d e f

g h i

Then the cross-gradient computed for the neighborhood is simply:

```
Edge = MAX(ABS(a - i), ABS(c - g))
```

The Sobel computed for the neighborhood is more complex. The vertical and horizontal edges are computed separately, squared, added together, and the square root of the sum is computed. (The system performs this operation more efficiently than the following definition suggests.)

```
Dx = a + 2*d + g - (c + 2*f + i)

Dy = a + 2*b + c - (g + 2*h + i)

Edge = SQRT(Dx*Dx + Dy*Dy) / 4
```

"Dx" and "Dy" are the edge magnitudes in the horizontal and vertical directions, respectively. "Dx" and "Dy" are more clearly depicted as convolutions with positive and negative weights:

```
     1   0   -1              1   2   1
Dx = 2   0   -2         Dy =  0   0   0
     1   0   -1             -1  -2  -1
```

For every neighborhood, the edge magnitude has been computed ("Edge" above) and compared with the V.EDGE.STRENGTH system parameter. If the edge magnitude is greater than the threshold, the binary edge bit is set to 1. Otherwise, it is set to 0.

To keep the threshold values comparable for the two edge operators, a normalization factor is applied to the result of the Sobel operator before the threshold comparison. Therefore, the V.EDGE.STRENGTH threshold used with one edge operator is generally useful for the other. However, you should always select the threshold specifically for the operator being used.

Example

Perform the cross-gradient for all cameras:

```
PARAMETER V.EDGE.TYPE = 1
```

Related Keywords

VEDGE (program instruction)
VPICTURE (program instruction)
V.BINARY (system parameter)
V.EDGE.STRENGTH (system parameter)

Syntax

```
VFEATURE (index)
```

Function

Return specified information about the object most recently VLOCATEd or the prototype most recently displayed by the VSHOW program instruction.

Usage Considerations

The VFEATURE function refers to the object most recently VLOCATEd or VSHOWed, regardless of which program task executed the VLOCATE or VSHOW instruction. Consequently, for predictable operation, only one program should execute VLOCATE and VSHOW instructions.

The VSHOW monitor command has no effect on the data available with this function.

Parameter

index Real value specifying the desired information item. The value must be in the range 1 to 50, inclusive (see below).

Details

The VFEATURE indexes, corresponding values, and units of measure are listed in Table 2-1, Table 2-2, and Table 2-3. All are explained in detail below.

Table 2-1. VFEATURE Function Data

Index	Value	Units (comment)
1	Valid	(TRUE/FALSE)
2	X	millimeters
3	Y	millimeters
4	Z	millimeters
5	RX	degrees
6	RY	degrees
7	RZ	degrees
8	Encoder offset	encoder counts
9	Verify percentage	percentage
10	Area	raw pixels
11	Region ID	
12	Instance ID	

Table 2-1. VFEATURE Function Data (Continued)

Index	Value	Units (comment)
13	Min_X	millimeters
14	Max_X	millimeters
15	Min_Y	millimeters
16	Max_Y	millimeters
17	# holes in region	
18	Time	seconds
19	Flags	(bit field: TRUE/FALSE)
20	Valid gripper	(number of the gripper)
21	Hole number	(number of the hole)
22	Parent number	(number of hole's parent)
23	Virtual camera number	(1 through 32)
24	Effort level	(0 through 4)
25	Color of prototype	(0 or 1)
26	# samples taught	
27	# bounds	
28	Min area of prototype	raw pixels
29	Max area of prototype	raw pixels
30	Cameras of prototype	(bit field: 1 to 16)
31	Cameras of prototype	(bit field: 17 to 32)
32	First edge number	
33	Last edge number	
34	X constraint of prototype	mm (defined during training)
35	Y constraint of prototype	
36	Angular constraint of proto	
37 - 39	(Reserved for future use)	
40	Area of all holes	raw pixels
41	Perimeter	millimeters
42	Centroid X	millimeters
43	Centroid Y	millimeters
44	Minimum radius angle	degrees
45	Maximum radius angle	degrees
46	Minimum radius	millimeters
47	Maximum radius	millimeters

Table 2-1. VFEATURE Function Data (Continued)

Index	Value	Units (comment)
48	2nd moments major axis	degrees
49	Major ellipse radius	millimeters
50	Minor ellipse radius	millimeters

ITEM 1. VFEATURE(1) has the value TRUE when the object or prototype information is valid; otherwise it is FALSE.

The information is valid when a VLOCATE succeeds in locating an object in an image or VSHOW succeeds in getting a prototype from the loaded prototype list.

The information is invalid when a no-wait VLOCATE fails to locate an object or a VSHOW instruction fails to find a loaded prototype. An error occurs if the object information is invalid and a VFEATURE is executed with an index other than 1.

ITEMS 2 to 7. X, Y, Z, RX, RY, and RZ are the components of the object or prototype transformation. (Note that the complete transformation is optionally returned by the VLOCATE and VSHOW instructions.) The component RX is always 0, and RY is always 180. RZ is the orientation of the object in the horizontal plane, the image plane.

The setting of the V.CENTROID switch may affect the location components. If the region is unknown (that is, the object name is "?") and the V.CENTROID switch is enabled, the X,Y location is for the region centroid and RZ is 0. If the V.CENTROID switch is disabled, the location is the center of the bounding box for the unknown region. The setting of the V.CENTROID switch has the same effect on the location components after a VLOCATE in find-hole mode. However, after a VSHOW instruction in get-hole mode, the X,Y location is always the centroid of the prototype.

After VLOCATE, the coordinates are in the reference frame of the camera. After a VSHOW instruction, the coordinates are in the reference frame of the prototype (origin of the coordinate system is the prototype centroid). Note that VSHOW draws the prototype in the Vision display window so that the center of its bounding box is at the center of the window.

ITEM 8. This VFEATURE item is the belt encoder value at the time the strobe light fired (see the V.STROBE system switch). (This value is also available from the V^+ DEVICE real-valued function.)

ITEM 9. Verify percentage is a recognition certainty value. The verify percentage is 0 if the region VLOCATEd is unknown. Otherwise, it is the percentage of the weighted boundary of the prototype that was verified. If following a VSHOW instruction, this is the verify percentage required for recognition of an instance of the prototype.

ITEM 10. Following a VLOCATE, this is the area of the region in camera pixels. If the V.SUBTRACT.HOLE system switch has been enabled, the region area has the areas of holes subtracted from it. Otherwise, they are included. In either case, the total area of the holes is available as VFEATURE item #40.

Note that this is the area of the region, not the area of the prototype instance. Following a VSHOW, this feature is the area of the prototype.

ITEMS 11 and 12. (After VLOCATE only) Region ID is a count of the number of regions encountered. Instance ID is a count of the number of objects recognized and regions not recognized. These ID numbers start at 1 and count up.

The distinction between region and instance is for touching and overlapping parts. If two objects touch and both are recognized, they account for 1 region ID increment and 2 instance ID increments. If a region consists of multiple objects and the objects are not recognized, the vision system has no way of knowing how many objects there are, so the region ID and instance ID counts are both incremented by 1.

If the vision V.DISJOINT system switch has been enabled, there may be more regions than instances, because a single prototype may span two or more regions.

ITEMS 13 to 16. Min_X, Max_X, Min_Y, and Max_Y define a bounding box for the region VLOCATEd or the prototype last VSHOWed. The bounding box is an orthogonal rectangle that encloses the region, with the four corners: (Min_X, Min_Y), (Min_X, Max_Y), (Max_X, Max_Y), (Max_X, Min_Y). One application of a bounding box definition is for finding clear space for placing things on a conveyor belt or a pallet. For a region, the bounding box coordinates are in the reference frame of the camera. For a prototype (following a VSHOW instruction), the coordinates are with respect to the reference frame of the prototype.

ITEM 17. Following VLOCATE, this is the number of holes detected in the region. Following VSHOW, this is the number of holes in the prototype.

ITEM 18. This is the elapsed time (in seconds) spent to acquire and analyze the image and to perform recognition. Since it is elapsed time, it includes the time spent to update the display and to respond to any commands such as VSTATUS and VQUEUE. The detailed procedure for keeping time is as follows.

Timekeeping starts when the VPICTURE or VWINDOW command is received. A time record is kept for each object recognized (or region analyzed and not recognized) in the image. As each object is entered in the queue, making it ready to be VLOCATEd, the elapsed time is noted. Timekeeping is reset as the search begins for the next object in the image. Consequently, the elapsed time associated with the first object includes the time spent to acquire the image.

Other consequences of this timekeeping procedure are as follows. If multiple objects are recognized in one region, the time spent performing boundary analysis is attributed only to the first object located. Furthermore, if the V.DISJOINT switch has been enabled, the time spent analyzing the boundaries of all regions is attributed to the first object recognized. The elapsed time reported for subsequent objects is only the time spent in the recognition algorithm.

ITEM 19. “Flags” is a bit field. Bit #1 is reserved for future use.

Bit #2 is defined when the V.HOLES switch has been enabled. If set, the bit indicates that the region (or hole) VLOCATEd helped to verify the recognition of the object. The boundary of a region helps to verify the recognition if the region boundary coincides with the prototype boundary. Only part of the hole boundary must coincide in order to help.

The other bits in the bit field are reserved for future use.

ITEM 20. Following a VLOCATE, this is the number of the grip position that is clear. This VFEATURE entry is zero if no grippers have been defined via VDEFGRIP, or if none of the grip positions are clear. Otherwise, it is the number of the first grip that is clear. Grips are tested in the order of their numbering, 1 to 4.

ITEMS 21 and 22. “Hole number” and “parent number” provide a numbering of holes for determining parent, child, and sibling relationships. Following a VLOCATE in the normal find-object mode or a VSHOW instruction in get-prototype mode, the “hole number” is always 0 and the “parent number” is always –1. The parent in this case is the background. Following a VLOCATE in the special find-hole mode or a VSHOW in get-hole mode, the “hole number” is an integer value, 1 or greater. All of the holes in the object are numbered, including holes in holes. The “parent number” of a hole is the parent’s “hole number”. Thus, if an object has one hole and the hole has one hole in it, the first hole’s “hole number” is 1 and its “parent number” is 0. The innermost hole has a “hole number” of 2, with a “parent number” of 1.

ITEM 23. After a VLOCATE, this is the number of the virtual camera with which the object was located. After a VSHOW, this is the number of a virtual camera associated with the prototype. (VFEATURE items #30 and #31 are bit masks representing all cameras associated with the prototype.)

ITEM 24. (After VSHOW only) The effort level assigned to a prototype during training.

ITEM 25. (After VSHOW only) The color (black or white) that the prototype had during training.

ITEM 26. (After VSHOW only) The number of prototype samples taught during training. This is a measure of the reliability of the prototype.

ITEM 27. The number of bounds in the prototype VSHOWed or region VLOCATEd. If a prototype, the count includes holes. If a region, the count does not include holes. The bounds counted are lines and arcs. A solid disk has one bound, a rectangular plate has four bounds, etc. In get-hole mode, it is the number of bounds in the region for the hole.

ITEMS 28 and 29. (After VSHOW only) These are the minimum and maximum areas associated with the prototype. The values are assigned to the prototype during training.

ITEMS 30 and 31. (After VSHOW only) These two items are bit masks representing the virtual cameras associated with the prototype. Both items are 16-bit fields, with each bit representing a different virtual camera. Item #30 represents virtual cameras 1 through 16. The least-significant bit of item #30 represents camera number 1, and the highest bit represents camera number 16. Item #31 represents cameras 17 through 32, with the ordering similar to that of item #30.

The prototype is associated with those cameras whose corresponding bits are set to 1. For example, the value 6 in item #30 means that cameras 2 and 3 are associated with the prototype (^B10+^B100 = 6).

ITEMS 32 and 33. (After VSHOW only) After VSHOW of a prototype, item #32 is 1 and item #33 is the number of bounds in the prototype's main region (not counting holes). These items define the range of edge numbers for the main region. After VSHOW in get-subproto or get-hole mode, items #32 and #33 contain the range of edge numbers for the subprototype or hole's region. The subprototype numbers are relative to the prototype, not the subprototype (otherwise, item #32 would always be 1 in get-subproto mode).

ITEMS 34 to 36. (After VSHOW only) During prototype training, constraints can be defined that limit recognition to a specific area and rotation relative to the defined prototype. These items return the x, y, and angular constraints defined during training.

ITEM 40. (After VLOCATE only) This is the total area of all holes in the region last located with a VLOCATE instruction.

ITEM 41. (After VLOCATE only) If the V.PERIMETER system switch has been enabled, this is the outer perimeter of the region in millimeters.

ITEMS 42 and 43. (After VLOCATE only) If the V.CENTROID system switch has been enabled, these are the X,Y coordinates of the region centroid. These items are undefined if V.CENTROID has not been enabled. The positions and areas of holes are not considered in the centroid computation. (The V.SUBTRACT.HOLE system switch does not affect this.)

ITEMS 44 to 47. (After VLOCATE only) If the switches V.MIN.MAX.RADII and V.CENTROID have been enabled, these VFEATURE items indicate the points on the boundary of the region that are closest to and farthest from the region centroid. Items #44 and #45 indicate the direction of the closest and farthest points relative to the region centroid. Items #46 and #47 are the distances from the centroid to the points, measured in millimeters.

ITEMS 48 to 50. (After VLOCATE only) These are the dimensions of the "best-fit ellipse" to the region. System switches V.2ND.MOMENTS and V.CENTROID must have been enabled for this computation. Item #48 is the direction of the region major axis (axis of least inertia), in the range –90 to +90 degrees. Items #49 and #50 are the radii of the best-fit ellipse for the region. The ellipse is centered at the region centroid, and its major axis coincides with the region major axis. (See the description of the V.2ND.MOMENTS switch for an explanation of the derivation.)

Table 2-2. VFEATURE Function Data for ObjectFinder (following VLOCATE)

Index	Value	Units (comment)
1	Valid	(TRUE / FALSE)
2	X	millimeters
3	Y	millimeters
4	Z	millimeters
5	RX	degrees
6	RY	degrees
7	RZ	degrees
8	Encoder offset	encoder counts
9	Verify percentage	percentage
11	Model number	in planning list, 1 - 10
12	Instance ID	integer
13	Min_X	millimeters
14	Max_X	millimeters
15	Min_Y	millimeters

Table 2-2. VFEATURE Function Data for ObjectFinder (following VLOCATE)

Index	Value	Units (comment)
16	Max_Y	millimeters
18	Time	seconds
23	Virtual camera number	1 - 32
27	# features in model	integer

See the descriptions following **Table 2-1** for each item in **Table 2-2**. Exceptions are shown below.

ITEMS 11. (After VLOCATE only) The model number is the number of the ObjectFinder model used in the planning list (1 - 10).

ITEMS 12. (After VLOCATE only) Instance ID is a count of the number of objects recognized and regions not recognized. These ID numbers start at 1 and count up.

ITEM 27. The number of features in the model VLOCATEd. All of the line segments and circular arcs in the model are counted, even if the weight (after multiple-instance training) is zero.

Table 2-3. VFEATURE Function Data for ObjectFinder (following VSHOW)

Index	Value	Units (comment)
1	Valid	(TRUE/FALSE)
2	X	millimeters
3	Y	millimeters
4	Z	millimeters
5	RX	degrees
6	RY	degrees
7	RZ	degrees
9	Verify percentage	percentage
12*	Min verify percent	percentage
13	Min_X	millimeters
14	Max_X	millimeters
15	Min_Y	millimeters
16	Max_Y	millimeters
17	# of pairs	integer
19*	Avg verify percent	percent

* Indexes 12, 19, and 20 are a set that is computed following multi-instance training.

Table 2-3. VFEATURE Function Data for ObjectFinder (following VSHOW)

Index	Value	Units (comment)
20*	Max verify percent	percent
21	Hierarchical level	
22	Parent number	always –1
23	Virtual camera number	1 - 32
24	Effort level	0
25	Convergence measure	
26	# samples taught	integer
27	# features in the model	integer
28	Max pixel variance	
29	Max loc distance	

* Indexes 12, 19, and 20 are a set that is computed following multi-instance training.

See the descriptions following **Table 2-1** for each item in **Table 2-3**. Exceptions are shown below.

ITEM 12. Minimum verify percentage is a recognition certainty value. The verify percentage following a VSHOW instruction is the minimum verify percentage required for recognition of an instance of the finder model.

ITEM 17. Number of pairs. Following VSHOW, this is the number of pairs in the finder model.

ITEM 19. Average verify percentage is a recognition certainty value. When following a VSHOW instruction, this is the *average* verify percentage required for recognition of an instance of the finder model.

ITEM 20. Max verify percentage is a recognition certainty value. When following a VSHOW instruction, this is the *maximum* verify percentage required for recognition of an instance of the finder model.

ITEM 21. Hierarchical level. This is the image processing level (0 - 2). See **VTRAIN.FINDER** for details.

ITEM 25. Convergence measure. This is a measure of the convergence of the feature weights during multi-instance training. Higher means more convergence (i.e., higher confidence that the weights are really converging to some set of constant values). This would mean that the model is stabilizing. In most cases, the value should be at least 3. However, a value of 2 may be sufficient for some applications.

ITEM 27. The number of features in the finder model VSHOWed. All of the line segments and circular arcs in the model are counted, even if the weight (after multiple-instance training) is zero.

ITEM 28. Maximum pixel variation (measured in units of pixels). This parameter controls the fitting of features to edges.

ITEM 29. Maximum location distance (measured in units of pixels). This parameter controls the distance between proposals. It is used in a strategy that attempts to prevent too many proposals from being made in the same location.

Example

```
IF VFEATURE(1) THEN
   TYPE "Recognition time = ", VFEATURE(18)
END
```

Related Keywords

VLOCATE (program instruction)
VSHOW (program instruction)

Syntax

```
VFIND.ARC (cam, mode, dmode, effort, type) data[i] = ibr

VFIND.ARC (cam, mode, dmode, effort, type) data[i] = 1,
                                              xc, yc, r, rr, ang0, angn
```

Function

Fit a circular arc to an image edge bounded by a window that is shaped like a ring or a ring segment.

Usage Considerations

If the calibration being used includes correction for perspective distortion, the correction is applied to the given center of the arc (xc, yc) and to the returned values describing the fit arc.

Adept recommends that you use the first syntax.

Parameters

`cam` Optional real-valued expression that specifies a virtual camera number. The default is 1. The camera number is used to determine which V.EDGE.STRENGTH parameter to use.

`mode` Optional bit-field expression. The default is 0. In summary, the bit-mask values and their meanings are listed here (see below for details):

Circle color: 0 = dark, 1 = light
Find: 0 = center only, 2 = radius only, 4 = both
Search start position: 0 = center, 8 = inner, 16 = outer

`dmode` Optional real-valued expression specifying the display mode for this operator. The choices are: –1 = no draw, 0 = erase, 1 = draw solid, 2 = complement, 3 = draw dashed, and 4 = complement dashed. The default is 1 (draw solid).

`effort` Optional real-valued expression indicating the effort level to use when finding the arc. The effort level ranges from 1 (least effort) to 100 (maximum effort). The default is 20.

`type` Optional real-valued expression specifying the type of arc finder: –2 = dynamic binary, –1 = raw binary, 0 = run-length binary, 2 = fine-edge. Default is 2 (fine-edge).

`data[]` Real array describing the outcome of the arc fit:

data[i+0] = TRUE if an arc was fit; otherwise, FALSE

data[i+1] = TRUE if any part of the search window falls off the Vision display window

data[i+2], data[i+3] = X,Y coordinates of the arc center, in millimeters

data[i+4] = Radius of the arc, in millimeters

data[i+5] = Percentage of the estimated arc's extent for which edge points were found: 0.0 to 100.0

data[i+6] = Maximum error (distance from the fit arc to the most distant edge point found), in pixels

data[i+7] = Maximum error toward the inside of the circle or arc, in pixels

data[i+8] = Maximum error toward the outside of the circle or arc, in pixels

data[i+9] = Percent of edge points filtered out

`i` Optional array index that identifies the first element to be defined in "**`data[]`**". The default is 0. If a multiple-dimension array is specified, only the right-most index is incremented as the values are assigned.

`ibr` Integer value specifying the image buffer region within which to search for an arc. Image buffer regions specify both a size and a frame store (see the description of VDEF.AOI).

`xc, yc` Real-valued expressions specifying the center coordinate of the estimated arc circle, in millimeters. This coordinate may be outside the image, but it must be within a 1000-pixel radius of the image origin (bottom left corner) after being transformed from millimeters to image pixels.

`r` Real-valued expression specifying the radius of the estimated arc circle, in millimeters. The value for "r" must be in the range 1 to 512 image pixels after it is converted from millimeters. Furthermore, there is a minimum arc circumference (see "ang0, angn" below).

`rr` Real-valued expression specifying the radius range for the search, in millimeters. This is the distance across the estimated radius within which the vision system searches for the arc's edge points. In other words, the search extends from (r–rr / 2) to (r+rr / 2). The range of acceptable values for "rr" is between 0 and (2*r).

`ang0, angn` Optional real-valued expressions specifying the angular range for the search in degrees. If "ang0" and "angn" are 0 and 360 (or 0 and 0), the search space is a complete ring and a circle is fit. The acceptable range of values for these parameters is –1000 to +1000

degrees. If (angn–ang0) exceeds 360 degrees, a complete ring is assumed. The circumference of the estimated arc must be at least three pixels. The arc circumference, with "r" in pixels, is (2∗π∗r∗(angn–ang0) / 360). The defaults are 0.

Details

VFIND.ARC, operating in grayscale mode, finds a circular arc edge with subpixel accuracy. The instruction has three basic modes of operation: find the center and radius of the arc's circle; find only the center, knowing the radius ("dim3" of the area of interest definition); or find only the radius, knowing the center ("dim1" and "dim2" of the area of interest definition). The operator takes about the same amount of processing time for each of the three modes, but the accuracy of the results is different, especially with small angular ranges. This is clarified below.

VFIND.ARC types # –2, # –1, and #0 are binary arc finders. That is, arcs are fit to binary edge points, which are black-to-white or white-to-black transitions in the image.

Type # –2 arc finders use the grayscale frame store. (This is the data visible in VDISPLAY mode #1.) The values of the threshold parameters V.THRESHOLD and V.2ND.THRESH **at the time the finder is executed** are used to determine which pixels are black and white. Therefore, a different binary threshold can be used for each finder.

Type # –1 arc finders use the raw-binary frame store. (This is the data visible in VDISPLAY mode #2.) This is binary or edge data, depending on the setting of the V.BINARY system switch.

Type #0 arc finders use the processed binary data produced by VWINDOW or VPICTURE in modes # –1 or #0. (This data is shown in VDISPLAY mode #3.) Note that system parameters such as V.MIN.AREA and V.FIRST.COL are used during picture processing, whereas they are not taken into account with type # –1 or # –2 finders. Again, the image data is binary or edge data, depending on the setting of the V.BINARY system switch.

Type #2 arc finders are the highest-precision arc finders. The edge points used in the least-squares type of fit algorithm are found in the grayscale frame store with subpixel accuracy. (The image data used is visible in the Vision display window in VDISPLAY mode #1.)

The portion of the image through which VFIND.ARC searches for edge points is shaped like a ring or a ring segment. This area is defined by the area of interest definition or the instruction parameters for the estimated arc center (xc,yc), estimated arc radius (r), the search range (rr), and the angular range (ang0,angn).

The mode bits for the VFIND.ARC "mode" parameter are defined as follows (they all default to zero):

Bit 1 (LSB) Dark inside (0) versus light inside (1) (mask value = 1)

If the inner part of the arc (or circle) is light, this bit should be set. Otherwise, this bit should be clear. Based on this bit, VFIND.ARC searches for edges with a particular sign: pixel transitions that are light-to-dark or dark-to-light.

Bits 2,3 Center only (0), radius only (2), both (4) (mask value = 6)

Center only (0): In this mode, the given radius ("dim3" of the area-of-interest definition) is assumed to be precise, and only the arc's circle center is computed.

Radius only (0): In this mode, the given center (first and second elements of the area of interest definition) is assumed to be precise, and only the arc's radius is computed.

Both (4): In this mode, both the arc's circle center and radius are fit. The given arc center ("dim1" and "dim2" elements of the area-of-interest definition) and radius ("dim3" of the area-of-interest definition) are used only to define the area to be searched for edge points.

Bits 4,5 Center (0), inner (8), outer (16) (mask value = 24)

This defines the initial search point for edges. The search for edges is along radial lines from the center of the arc. The length of each edge search is "rr". If more than one edge is found along one radial line, these bits (#4 and #5) of the mode parameter determine which one to use. By default, the edge closest to the center of the search area (the radius ["dim3" of the area of interest definition] from the center ["dim1" and "dim2" of the area of interest definition]) is chosen. This is fine for most situations. However, if nearby edges are known to be present that do not belong to the arc edge, one of these mode bits may be set to help avoid use of the non-arc edges. If inner (bit #4) is set, the edge closest to the center of the arc's circle is used. If outer (bit #5) is set, the edge farthest from the center is used. Only one bit may be set.

The VFIND.ARC operator is displayed in the Vision display window as a ring or ring segment. The outer half of the ring is either dark blue or light blue, depending on the given color of the arc's circle ("mode" bit # 1). The inner half is drawn in the other color. Bisecting the width of the ring is a guide arc drawn in green—this is the estimated arc (defined by elements 1, 2, 3, 5, and 6 of the area of interest definition). The inner and outer arcs show the search range (third element of the area of interest definition) for the search. The initial search point is indicated by a knot in yellow on either the inner, center, or outer arc (depending on "mode" bit 4 and 5). The resulting fit arc is drawn in red (if one is found).

When using the first syntax, the shape defined by VDEF.AOI should be 5, 6, 7, or 8. **Figure 2-6** shows the mechanics of an arc finder. These mechanics are determined by the "mode" bits in the VFIND.ARC instruction and the shape specification in the area-of-interest definition. The appearance of the tool shown in **Figure 2-6** was created with a VDEF.AOI instruction specifying shape 5. The tool dimensions from the VDEF.AOI instruction are as follows: dim1 and dim2 are the center of the finder tool within the image; dim3 is the guide radius (shown as a dotted line in **Figure 2-3**); dim4 is the search range; and ang1 and ang2 indicate the angular range of the search. The "mode" bits from the VFIND.ARC instruction are: bit 1 is 0, indicating a search for an arc with a dark inside (shown by the dark blue inner radius of the tool); bit 2 = 0, bit 3 = 1 indicating that the arc's radius and center will be calculated; and bit 4 = 0, bit 5 = 1, indicating that the search will start from the outer radius (shown by a white dot on the outer radius). The following code will create the arc finder in **Figure 2-6**:

```
arc_ibr = 2011          ;area-of-interest 2, frame buffer 11
arc = 5                 ;shape #5
dim1 = 35
dim2 = 40.5
dim3 = 12
dim4 = 8
ang1 = 0
ang2 = 290
mode = 16+4+0           ;start from outside (mask = 16), find
                        ;  both (mask = 4), dark inside (mask = 0)
VDEF.AOI arc_ibr = arc, dim1, dim2, dim3, dim4, ang1, ang2
VFIND.ARC (1, mode, 1) data[] = arc_ibr
```

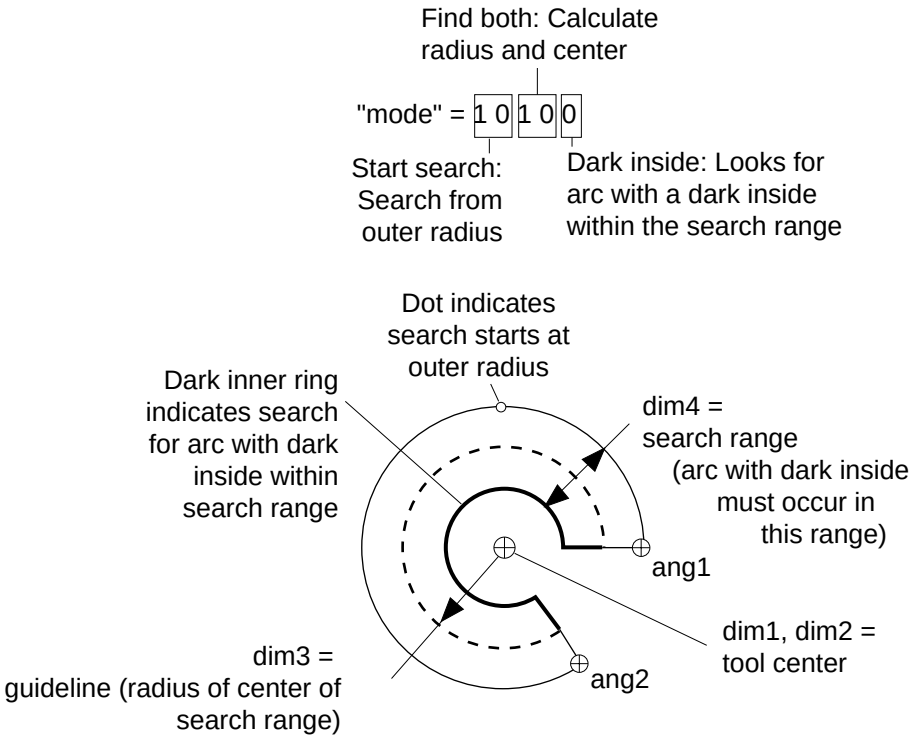


Figure 2-6. Arc Finder Shape

If part of the operator falls off the Vision display window (that is, if part of the search area must be clipped by the virtual frame buffer defined in the image buffer region), that situation is noted by the value of “data[i+1]”. The vision system still attempts to fit an arc.

No arc is fit if fewer than three edge points are found, or if the resulting arc radius would be too large. Then “data[i+0]” returns FALSE.

Edge points are found within the search area, and then an arc is fit to the edge points using a special triangulation algorithm. The effort level specifies how much time should be spent searching for edge points. If the effort level is 1, only a few edge points (about five, evenly spaced across the angular range) are sought. If the effort level is 100, as many edge points as possible are sought. High effort levels with big arcs use a lot of execution time. Unless the arc image is small or noisy, a low effort level should be used. This should provide adequate accuracy.

VFIND.ARC optionally filters out edge points that are far from the fit arc and then refits the arc. This is an iterative process that increases the execution time but provides more accurate and consistent results, particularly when noise is present in the image. Two system parameters, V.MAX.SD and V.MAX.PIXEL.VAR, control the filtering process. VFIND.ARC filters edge points if V.MAX.SD is nonzero.

Otherwise, filtering is disabled. V.MAX.SD specifies the number of distance standard deviations from the fit arc beyond which edge points should be discarded. V.MAX.PIXEL.VAR specifies the maximum pixel distance below which no edge points should be discarded. (See the description of the V.MAX.SD system parameter for more information on the iterative, filtering process.)

The edge points found by VFIND.ARC are displayed in the Vision display window when the V.SHOW.EDGES system switch is enabled. Edge points used in the final fit of the arc are displayed in white. Edge points discarded during the filtering process are displayed in gray. (Note that displaying the edge points is computationally expensive, so it should be enabled only for investigative purposes.)

Items “data[i+7]”, “data[i+8]”, and “data[i+9]” are returned only when edge filtering is enabled (that is, when V.MAX.SD is nonzero). Item “data[i+9]” is the percentage of edge points filtered out. This percentage ranges from 0 up toward 100. It can never be 100, however, because the filtering process always stops before filtering out too many points.

The execution time for VFIND.ARC is proportional to the effort level multiplied by the circumference of the arc. Also, the execution time increases as the search range increases. An approximate formula for computing the execution time in microseconds is shown below—where C is the arc circumference in pixels, E is the effort level, and RR is the radius range in pixels. This formula does not account for edge filtering, which will add additional time that depends on the number of filtering iterations performed.

```
Time = C * E * (11 + RR/6)
```

After an arc has been fit (that is, after the final fit if edge filtering is enabled), the edge points used to fit the arc are compared with the fit arc, and the distance from the fit arc to the most distant edge point is returned (data[i+6]). This distance is in pixels. If this value is not close to one, the arc edge in the image is either very rough or some extraneous edges were detected and used in the arc fitting. If edge filtering is enabled, “data[i+7]” and “data[i+8]” more specifically indicate the most distant edge points inside and outside the arc, respectively. (Data item “data[i+6]” is the maximum of “data[i+7]” and “data[i+8]”.)

Accuracy is more dependent on the angular range of the arc than on the effort level. Angular ranges of 180 degrees or more provide very accurate (subpixel) results. Fitting an arc with an angular range of less than 45 degrees can have very large errors. This assumes that both the center and radius are being computed. The results are much more accurate if one or the other is known in advance.

Example

The following instruction will use a fine-edge (default type) arc finder to look for a light circle on a dark background (mode bit #1 set), computing both the center and radius (mode bit #3 set). The effort level is 50. The estimated center location and radius are (30,46) and 10, respectively. The search range is 5; the angular range is 360 degrees (0 to 0).

```
VDEF.IAO 2001 = 5, 30, 46, 10, 5
VFIND.ARC (, 1 BOR 4, , 50) data[] = 2001
```

Related Keywords

VDEF.AOI (program instruction)
VFIND.LINE (program instruction)
VFIND.POINT (program instruction)
V.MAX.PIXEL.VAR (system parameter)
V.MAX.SD (system parameter)
V.SHOW.EDGES (system switch)

Syntax

```
VFINDER (cam, type, dmode, how_many_total {, times[]}) ibr
```

Function

This instruction performs ObjectFinder recognition using the planning associated with the given virtual camera.

Parameter

cam Real-valued expression indicating the virtual camera's "planning" to use and parameters to read.

type Type of recognition operation to perform. Default value is 1.
1 = ObjectFinder recognition
2 = ObjectFinder recognition for multi-instance training

dmode Real-valued expression that specifies the display mode to use when displaying the border of the window:

–1	No draw
0	Erase
1	Draw solid (default)
2	Complement
3	Draw dashed
4	Complement dashed

how_many_total
Specifies the total number of objects to find. In the case of multiple models planned together, this is regardless of which models are found. DEFAULT = –1 (find as many objects as possible). If 0 is given, none will be found, but the image is still processed through feature extraction.

times[ ] Array of "max_time" information for the recognition process. The whole array is optional, defaults are as if "times[0]" = 0.
[0] How many "max_time" parameters to follow.
[1] Max time to first part. (defaults to V.MAX.TIME[cam])
[2] Max time to subsequent parts. (defaults to times[1])

ibr Integer value specifying the image buffer region within which to search. The AOI must be a nonrotated rectangle. Image buffer regions specify both a size and a virtual frame buffer (see the description of VDEF.AOI).

NOTE: There is no optional shape specification; it must be an image buffer region (ibr).

Details

This instruction performs ObjectFinder recognition using the planning associated with the given virtual camera. As shown below, several of the same switches and parameters apply as for prototype recognition.

Input parameters using virtual cameras

> **NOTE:** All switches and parameters should use the default settings, except as noted here.

The following settings are required:

PARAMETER V.MIN.AREA[vc] = 4

PARAMETER V.MIN.HOLE.AREA[vc] = 4

The following settings are suggested:

V.MAX.PIXEL.VAR[vc]	Same as for models trained.
V.EDGE.STRENGTH[vc]	Same as for models trained, assuming that the F-stop or illumination has not changed significantly from when the model was trained.
V.MAX.TIME[vc] = 2.0	Used if the "`times[]`" array is omitted.
V.MAX.VER.DIST[vc] = 5.0	

The following switches are available for enabling visual feedback:

V.SHOW.FBNDS[vc]	Similar to V.SHOW.BOUNDS, but for ObjectFinder.
V.SHOW.FEATS[vc]	Shows final features used for ObjectFinder.
V.SHOW.VERIFY[vc]	Same as for prototype recognition. The lines and arcs of the prototype are compared to the edges in the image, and they are drawn in the Vision display window.
V.SHOW.RECOG[vc]	Overlays the models on the found instances.

Syntax

```
VFIND.LINE  (cam, pos, dmode, effort, type) data[i] = ibr

VFIND.LINE  (cam, pos, dmode, effort, type) data[i] = 1,
                                    xc, yc, length, width, angle
```

Function

Fit a straight line to an image edge within a window.

Usage Considerations

If the calibration being used includes correction for perspective distortion, the correction is applied to the returned values describing the fit line.

Adept recommends that you use the first syntax.

Parameters

`cam` Optional real-valued expression that specifies a virtual camera number. The default is 1. The camera number is used to determine which V.EDGE.STRENGTH parameter to use.

`pos` Optional real-valued expression indicating the starting point in the search window: –1 = dark side, 0 = middle guideline, +1 = light side. The default value of "pos" is 0.

`dmode` Optional real-valued expression specifying the display mode for this operator. The choices are: –1 = no draw, 0 = erase, 1 = draw solid, 2 = complement, 3 = draw dashed, and 4 = complement dashed. The default is 1 (draw solid).

`effort` Optional real-valued expression indicating the effort level to use when fitting the line. The effort level can range from 1 (least effort) to 100 (maximum effort). The default is 50.

`type` Optional real-valued expression specifying the type of line finder: –2 = dynamic binary, –1 = raw binary, 0 = run-length binary, 2 = fine-edge. The default is 2 (fine-edge).

`data[]` Real array describing the outcome of the line fit:

data[i+0] = TRUE if a line was fit; otherwise, FALSE

data[i+1] = TRUE if any part of the search window falls off the vision display window

data[i+2], data[i+3] = X,Y coordinate on the line nearest to the initial search point, in millimeters

data[i+4] = Angle of the fit line, in degrees

data[i+5] = Percentage of the guideline's extent for which edge points were found: 0.0 to 100.0

data[i+6] = Maximum error (distance from the fit line to the most distant edge point found), in pixels

data[i+7] = Maximum error toward the dark side of the line, in pixels

data[i+8] = Maximum error toward the bright side of the line, in pixels

data[i+9] = Percent of edge points filtered out

ibr Integer value specifying the image buffer region within which to search for a line. Image buffer regions specify both a size and a virtual frame buffer (see the description of VDEF.AOI).

i Optional array index that identifies the first element to be defined in "**data[]**". The default is 0. If a multiple-dimension array is specified, only the right-most index is incremented as the values are assigned.

xc, yc Real-valued expressions specifying the center of the guideline (center line in the search window), in millimeters. The values for both "xc" and "yc" must be in the range –1000 to 1000 image pixels after being converted from millimeters.

length Real-valued expression specifying the length of the guideline (length of the search window), in millimeters. The value for "length" must be in the range 1 to 1000 image pixels after it is converted from millimeters.

width Real-valued expression specifying the width of the search window, in millimeters. The value for "width" must be in the range 1 to 1000 image pixels after it is converted from millimeters.

angle Optional real-valued expressions specifying the angle of the guideline, in degrees. The default is 0. However, angles of 90 and 270 degrees are most efficient.

Details

VFIND.LINE finds a linear edge in a window with subpixel accuracy (if a grayscale tool is used). First, edge points are found, and then a line is fit to the edge points using a least-squares fit algorithm.

VFIND.LINE types #–2, #–1 and #0 are binary line finders. That is, lines are fit to binary edge points, which are black-to-white or white-to-black transitions in the image.

Dynamic binary line finders (type #–2) use the grayscale frame store. (This is the data visible in VDISPLAY mode #1.) The values of the threshold parameters V.THRESHOLD and V.2ND.THRESH **at the time the finder is executed** are used to determine which pixels are black and white. Therefore, a different binary threshold can be used for each finder.

Raw binary line finders (type #–1) use the raw-binary frame store. (This is the data shown in VDISPLAY mode #2.) This is binary or edge data, depending on the setting of the V.BINARY system switch.

Run-length line finders (type #0) use the processed binary data produced by VWINDOW or VPICTURE in modes #–1 or #0. (This data is shown in VDISPLAY mode #3.) Note that system parameters such as V.MIN.AREA and V.FIRST.COL are used during picture processing and, therefore, will affect this type of finder, whereas they will not be taken into account for the other binary finders (type #–1 and #–2). Again, the image data is binary or edge data, depending on the setting of the V.BINARY system switch.

Fine-edge line finders (type #2) are the highest-precision finders. The edge points used in the least-squares-fit algorithm are found in the grayscale frame store with subpixel accuracy. (The image data used is visible in the Vision display window in VDISPLAY mode #1.)

At the center of the VFIND.LINE search window is the “guideline”, the user’s estimate of the edge location. The guideline is defined by a point (the center point), and an angle. The search area is further specified by the length of the guideline (breadth of scan), the width of the search window (range about the guideline within which the vision system is to search), and an initial search location.

In the examples shown below, the guidelines are horizontal (angle = 0), so the search windows are rectangles and the “width” of the search window is the height of the rectangles. The examples illustrate the three possible initial search locations.

The different possible positions for the initial search point allow control over the processing of multiple edges in the search window. If the initial search point is in the center position, edge points nearest the guideline are used (when there are multiple edge points perpendicular to the guideline to choose from). Otherwise, the edge points nearest the side containing the initial search point are used. See **Figure 2-7**.

The VFIND.LINE operator has a polarity that determines whether a light-to-dark or dark-to-light edge is being fit. The vision system assumes that the dark side of the edge is on the top of the line being searched for and the light side is on the bottom (assuming “angle” is 0). For visual reference, one half of the search

window is drawn in dark blue, and the other half is drawn in light blue; the guideline is drawn in green, and the initial search point is drawn in yellow; the fit line is drawn in red, and the point on the line closest to the initial search point is drawn in white. See **Figure 2-7**.

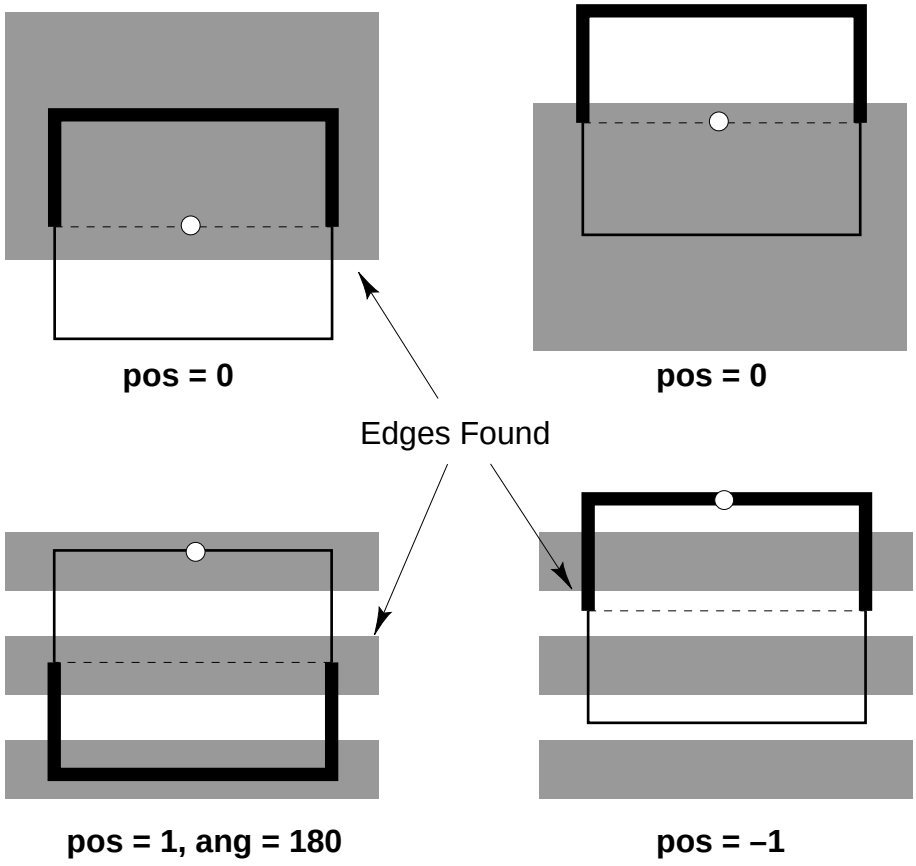


Figure 2-7. Line Finder Tool Start Position and Polarity

Figure 2-8 shows a finder tool and the instructions that created the tool.

```
shape = 1
dim1 = 20.4
dim2 = 25.6
dim3 = 15
dim4 = 12
ang1 = 0
rect_ibr = 3000+22    ;aoi 3, frame buffer 22
VDEF.AOI rect_ibr = shape, dim1, dim2, dim3, dim4, ang1
VFIND.LINE (1, 1, 1) data[] = rect_ibr
```

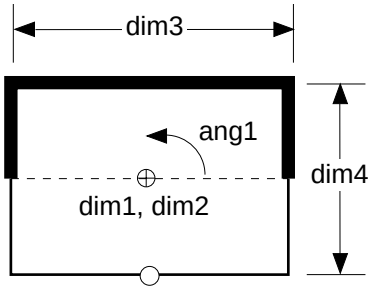
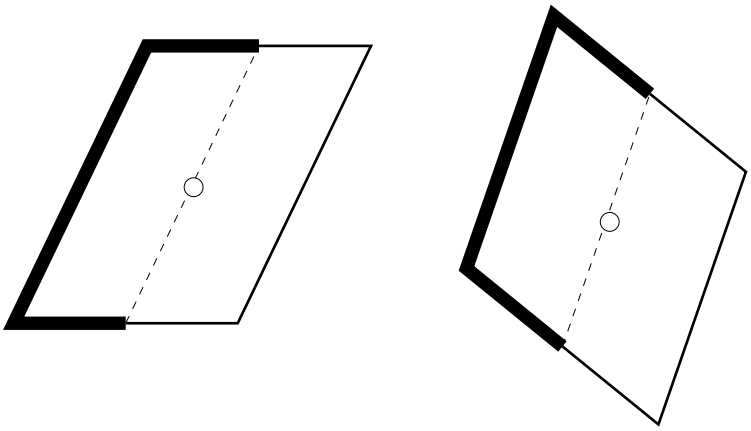


Figure 2-8. Sample Line Finder Tool

The area searched for edges always has its sides aligned along the horizontal, vertical, or diagonal directions. Consequently, the VFIND.LINE window is a parallelogram, not necessarily a rectangle. The guideline can have any orientation, but the sides of the search window at the ends of the guideline are always horizontal, vertical, or diagonal.

In the two search windows shown below, imagine that the guideline in the left-hand window is 20 degrees off vertical and the guideline in the right-hand window is 30 degrees off vertical. The windows have different shapes because the window automatically changes orientation from “vertical” to “diagonal” when the guideline is at 22.5 degrees, since that is halfway between 0 and 45 degrees. This is the worst-case shape distortion, since it is near the transition point.



The effort level specifies how dense the scan is along the guideline. Only a few edge points are sought if the effort level is 1. If the effort level is 100, as many edge points as possible are sought. The effort level affects both execution time and accuracy.

If part of the operator falls off the Vision display window (that is, if part of the search window must be clipped to the virtual frame store), "data[i+1]" notes it, and the vision system still attempts to fit a line. However, no line is fit if fewer than two edge points are found.

After a line has been fit, the edge points found are compared with the fit line, and the distance from the fit line to the most distant edge point is returned (data[i+6]). This distance is in pixels. If this value is not close to one, the linear edge in the image is very rough or some extraneous edges were detected and used in the line fitting.

> **NOTE:** For highest accuracy when fitting a line to a linear edge, the corners at the ends of the linear edge should not fall within the VFIND.LINE search window. Parts of the other edges that are connected to the corners may be detected by VFIND.LINE and used in the least-squares fit. This shifts the fit line away from the linear edge of interest toward one or both of the neighboring edges.

VFIND.LINE optionally filters out edge points that are far from the fit edge and then refits the edge. This is an iterative process that increases the execution time but provides more accurate and consistent results, particularly when noise is present in the image. Two system parameters, V.MAX.SD and V.MAX.PIXEL.VAR, control the filtering process. VFIND.LINE filters edge points if V.MAX.SD is nonzero. Otherwise, filtering is disabled. V.MAX.SD specifies the number of distance standard deviations from the fit line beyond which edge

points should be discarded. V.MAX.PIXEL.VAR specifies the maximum pixel distance below which no edge points should be discarded. (See the description of the V.MAX.SD system parameter for more information on the iterative filtering process.)

The edge points found by VFIND.LINE are displayed in the Vision display window when the system switch V.SHOW.EDGES is enabled. Edge points used in the final fit of the line are displayed in white. Edge points discarded during the filtering process are displayed in gray. (Note that displaying the edge points is computationally expensive, so it should be enabled only for investigative purposes.)

Items “data[i+7]”, “data[i+8]”, and “data[i+9]” are returned only when edge filtering is enabled (that is, when V.MAX.SD is nonzero). Item “data[i+9]” is the percent of edge points filtered out. This percentage ranges from 0 up toward 100. It can never be 100, however, because the filtering process always stops before filtering out too many points.

The execution time for VFIND.LINE is proportional to the effort level multiplied by the length of the guideline. Also, execution time increases as the width of the operator increases. And the operator is faster when rotated near 90 or 270 degrees than it is when rotated near 0 or 180 degrees. The cost of filtering depends on the number of iterations performed.

After a line has been fit (that is, after the final fit if edge filtering is enabled), the edge points used to fit the line are compared with the fit line, and the distance from the fit line to the most distant edge point is returned (data[i+6]). This distance is in pixels. If this value is more than one or so, the line edge in the image is very rough or some extraneous edges were detected and used in the line fitting. If edge filtering is enabled, “data[i+7]” and “data[i+8]” more specifically indicate the most distant edge points from the dark and bright sides of the line, respectively. (Data item “data[i+6]” is the maximum of “data[i+7]” and “data[i+8]”.)

Example

The following program segment finds a corner point with high precision, where the corner is formed by two linear edges. VFIND.LINE is used twice, one for each line. Then the program “line_line” (also shown below) is called to compute the point where the two lines intersect. (A similar program is shown in the programming example in ***AdeptVision User's Guide.***)

```
cam = 1                                    ;Define virtual camera

VPICTURE (cam) 2                           ;Quick frame grab

PARAMETER V.MAX.SD[cam] = 1.5              ;Filter beyond 1.5 Std Devs
PARAMETER V.MAX.PIXEL.VAR[cam] = 1         ;Keep points within 1 pixel
```

```
rect_ibr = 2011
VDEF.AOI rect_ibr = 1, 300, 270, 80, 90
VFIND.LINE (cam, , , 100) a[] = rect_ibr
VDEF.TRANS 80, -70, 0, 0, 90                ;Move the aoi
VFIND.LINE (cam, , , 100) b[] = rect_ibr

IF NOT (a[0] AND b[0]) THEN
     TYPE "Both edges not found"
     STOP
END

CALL line_line(a[2], a[3], a[4], b[2], b[3], b[4], x, y)

; X,Y is the precision corner point found
.
.
.

.PROGRAM line_line(x1, y1, ang1, x2, y2, ang2, x, y)

;ABSTRACT     Find the intersection point of two lines

     LOCAL dx1, dy1, dx2, dy2, f, fract, numerator

     dx1 = COS(ang1)
     dy1 = SIN(ang1)
     dx2 = COS(ang2)
     dy2 = SIN(ang2)
     numerator = (y2-y1)*dx2-(x2-x1)*dy2

     IF ABS(dx1) > ABS(dy1) THEN            ;Divide by larger number
          fract = dy1/dx1
          f = numerator/(fract*dx2-dy2)
          x = x1+f
          y = y1+fract*f
     ELSE
          fract = dx1/dy1
          f = numerator/(dx2-fract*dy2)
          y = y1+f
          x = x1+fract*f
     END
.END
```

Related Keywords

VDEF.AOI (program instruction)
VFIND.ARC (program instruction)
VFIND.POINT (program instruction)
V.MAX.PIXEL.VAR (system parameter)
V.MAX.SD (system parameter)
V.SHOW.EDGES (system switch)

Syntax

```
VFIND.POINT  (cam, pos, dmode, effort, type) data[i] = ibr

VFIND.POINT  (cam, pos, dmode, effort, type) data[i] = 1,
                                    xc, yc, length, width, angle
```

Function

In a search window, find the edge point that is nearest to one side of the window.

Usage Considerations

The frame store currently selected must contain a valid picture. Otherwise, an error results.

If the calibration being used includes correction for perspective distortion, the correction is applied to the found point returned.

Adept recommends that you use the first syntax.

Parameters

`cam`	Optional real-valued expression that specifies a virtual camera number. The default is 1. The camera number is used to pick the set of switches and parameters to use.
`pos`	Optional real-valued expression indicating the starting point in the search window: –1 = dark side; +1 = light side. The default value of "pos" is –1.
`dmode`	Optional real-valued expression specifying the display mode for this operator. The choices are: –1 = no draw, 0 = erase, 1 = draw solid, 2 = complement, 3 = draw dashed, and 4 = complement dashed. The default is 1 (draw solid).
`effort`	Optional real-valued expression indicating the effort level to use when searching for edge points. The effort level can range from 1 (least effort) to 100 (maximum effort). The default is 100.
`type`	Optional real-valued expression specifying the type of point finder: –2 = dynamic binary, –1 = raw binary, 0 = run-length binary, 2 = fine-edge. The default is 2 (fine-edge).
`data[]`	Real array describing the outcome of the search, as follows: data[i+0] = TRUE if an edge point was found, otherwise FALSE

data[i+1] = TRUE if any part of the search window falls off the Vision display window

data[i+2], data[i+3] = X,Y coordinates of the edge point found, in millimeters

ibr Integer value specifying the image buffer region within which to search for a point. Image buffer regions specify both a size and a frame store (see the description of VDEF.AOI).

i Optional array index that identifies the first element to be defined in "**data[]**". The default is 0. If a multiple-dimension array is specified, only the right-most index is incremented as the values are assigned.

xc, yc Real-valued expressions specifying the center of the search window, in millimeters. The values for both "xc" and "yc" must be in the range –1000 to 1000 image pixels after they are converted from millimeters.

length Real-valued expression specifying the length of the search window, in millimeters. The value for "length" must be in the range 1 to 1000 image pixels after it is converted from millimeters.

width Real-valued expression specifying the width of the search window, in millimeters. The value for "length" must be in the range 1 to 1000 image pixels after it is converted from millimeters.

angle Optional real-valued expressions specifying the orientation of the search window in degrees. The default is 0. However, angles of 90 and 270 degrees are most efficient.

Details

The VFIND.POINT instruction looks for edges in a search window. It operates like a "fat ruler", looking along a wide path in search of an edge. An example application would be to find the right-most point on a circle when you know the circle's approximate position. VFIND.POINT could also be used to find the end of a pin.

VFIND.POINT types # –2, # –1 and #0 are binary point finders. Binary edge points are black-to-white or white-to-black transitions. The maximum accuracy is one pixel.

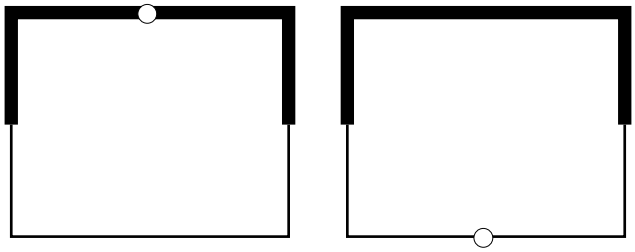
Dynamic binary point finders (type # –2) use the grayscale frame store. (This is the data visible in VDISPLAY mode #1.) The values of the threshold parameters V.THRESHOLD and V.2ND.THRESH **at the time the finder is executed** are used to determine which pixels are black and white. Therefore, a different binary threshold can be used for each finder.

Raw binary point finders (type # –1) use the raw-binary frame store. (This is the data shown in VDISPLAY mode #2.) This is binary or edge data, depending on the setting of the V.BINARY system switch.

Run-length binary point finders (type #0) use the processed binary data produced by VWINDOW or VPICTURE in modes # –1 or #0. (This data is shown in VDISPLAY mode #3.) Note that system parameters such as V.MIN.AREA and V.FIRST.COL are used during picture processing and, therefore, will affect this type of finder, whereas they will not be taken into account for the other binary finders (type # –1 and # –2). Again, the image data is binary or edge data, depending on the setting of the V.BINARY system switch.

Fine-edge point finders (type #2) find edge points with subpixel accuracy. The image data used is in the grayscale frame store (which is visible in the Vision display window in VDISPLAY mode #1).

VFIND.POINT is similar to the VFIND.LINE instruction. The search window is a parallelogram. Graphically, the operator looks the same except the center guideline is missing. As shown below, the knot (o) specifying the initial search position is on either one side of the window or the other.



Given the search window on the left above, the edge finder would return the (X,Y) coordinate of the edge point in the window that is closest to the top of the window. In the search window on the right, the edge point closest to the bottom of the window would be returned.

The VFIND.POINT operator has a polarity that determines whether a light-to-dark or dark-to-light edge is being fit. The vision system assumes that the dark side of the edge is on the top and the light side is on the bottom when the tool rotation is 0. For visual reference, one half of the search window is drawn in dark blue, and the other half is drawn in light blue. The initial search point is drawn in yellow. The edge point found is drawn in red. If the V.SHOW.EDGES system switch is enabled, all the candidate edge points within the search window (that is, those that have the correct polarity) are displayed in white. (Note that displaying all of the edge points is computationally expensive, so V.SHOW.EDGES should be enabled only for investigative purposes.)

Like VFIND.LINE, the VFIND.POINT operator always searches for edges along horizontal, vertical, or diagonal directions. Consequently, the search window is a parallelogram, not necessarily a rectangle. The sides of the window that contain the initial search point can have any orientation, but the other two sides of the search window are always horizontal, vertical, or diagonal.

VFIND.POINT returns a coordinate whose distance from the side of the search window containing the initial search point ("o") is accurate. Note that the **distance**—but not necessarily the coordinate—is accurate. In particular, the coordinate has only about one pixel of accuracy along the axis parallel to the "o" side of the window. The accuracy of the coordinate along the axis that is perpendicular to the "o" side of the window is much better, particularly for fine-edge VFIND.POINT operations, similar to that of a fine-edge ruler.

The effort level specifies the "fineness of the comb" used in the search for edge points. VFIND.POINT searches inside the window along search lines that run perpendicular to the side of the window containing the initial search point. The search lines are spaced one pixel apart. If the effort level is 100, all search lines are searched for edge points. If the effort level is 50, only every other search line is searched for edges. The effort level is basically the percentage of all potential search lines that will be searched for edge points.

For very high accuracy in locating a corner formed by two linear edges, the VFIND.LINE instruction should be used. (See the program example given with VFIND.LINE.) If one or both of the edges are circular arcs, the VFIND.ARC instruction may be used in a similar manner.

If part of the operator falls off the Vision display window (that is, if part of the search window must be clipped to the virtual frame store), "data[i+1]" notes it, and the vision system still searches for edge points.

The execution time for VFIND.POINT is proportional to the length of the search window. Also, the execution time increases as the width of the window increases, and as the effort level increases. The operator is faster when rotated near 90 or 270 degrees than it is when rotated near 0 or 180 degrees.

Example

The following instruction sequence uses the V.EDGE.STRENGTH system parameter for camera #3 to find the right-most edge point in a search rectangle. The search is from the "light side" (pos = 1) of a rectangle located at (100, 67.33). The search window is 40 millimeters high and 25 millimeters wide. The rotation angle is 90 degrees.

```
cam = 3
pos = 1
shape = 1
dim1 = 100
```

```
dim2 = 67.33
dim3 = 40
dim4 = 25
ang1 = 90
rect_ibr = 4012          ;area-of-interest 4, virt.frame buffer 12
VDEF.AOI rect_ibr = shape, dim1, dim2, dim3, dim4, ang1
VFIND.POINT (cam, pos) data[] = rect_ibr
IF data[0] THEN
     TYPE "Point found at (", /F0.3, data[2], ",", data[3], ")"
ELSE
     TYPE "No point found"
END
```

Related Keywords

VDEF.AOI (program instruction)
VFIND.ARC (program instruction)
VFIND.LINE (program instruction)
VRULERI (program instruction)
V.SHOW.EDGES (system switch)

Syntax

```
... V.FIRST.COL  [camera]
```

Function

Set the number of the first column of pixels to be processed.

Usage Considerations

A change to this parameter takes effect when the next VPICTURE or VWINDOW is executed.

This is an array of parameters—one for each virtual camera. (See the general description of parameters in the ***AdeptVision User's Guide*** for syntax details.)

Details

This parameter, together with V.LAST.COL, is used to set the range of camera pixel columns that are processed during VPICTURE and VWINDOW operations. Columns outside the range specified are ignored by the vision system. (VDEF.AOI is the preferred method for defining a processing border for VPICTURE and VWINDOW.)

This parameter must be assigned an integer value in the range 1 (at the left edge of the virtual frame buffer) to the current value of the parameter V.LAST.COL. The parameter V.FIRST.COL is set to 1 when the V$^+$ and AdeptVision systems are loaded into memory from disk.

Example

Process all camera images starting at the left-most border:

```
PARAMETER V.FIRST.COL = 1
```

Related Keywords

VDEF.AOI (program instruction)
V.FIRST.LINE (system parameter)
V.LAST.COL (system parameter)
V.LAST.LINE (system parameter)

Syntax

```
... V.FIRST.LINE  [camera]
```

Function

Set the number of the first line of pixels to be processed.

Usage Considerations

A change to this parameter takes effect when the next VPICTURE or VWINDOW instruction is executed.

This is an array of parameters—one for each virtual camera. (See the general description of parameters in the *AdeptVision User's Guide* for syntax details.)

Details

This parameter, together with V.LAST.LINE, is used to set the range of camera pixel lines that are processed during VPICTURE or VWINDOW operations. Lines outside the range specified are ignored by the vision system. (VDEF.AOI is the preferred method for defining a processing border for VPICTURE and VWINDOW.)

This parameter must be assigned an integer value in the range 1 (for the line at the bottom of the virtual frame buffer) to the current value of the parameter V.LAST.LINE. The parameter V.FIRST.LINE is set to 1 when the V$^+$ and AdeptVision systems are loaded into memory from disk.

Example

Start processing camera images beginning with the first line of data:

```
PARAMETER V.FIRST.LINE = 1
```

Related Keywords

VDEF.AOI (program instruction)
V.FIRST.COL (system parameter)
V.LAST.COL (system parameter)
V.LAST.LINE (system parameter)

Syntax

```
... V.FIT.ARCS   [camera]
```

Function

Enable or disable the fitting of circular arcs when performing boundary analysis.

Usage Considerations

A change to this switch takes effect when the next VPICTURE, VWINDOW, or VTRAIN operation is executed.

This is an array of switches—one for each virtual camera. (See the general description of switches in the *AdeptVision User's Guide* for syntax details.)

Details

This switch allows the user to disable arc fitting during boundary analysis. When arc fitting is disabled, the vision system attempts to fit only lines instead of attempting to fit both lines and arcs. This improves performance for scenes that contain no (or very few) boundary segments that appear as arcs.

If the vision system is doing prototype recognition and none of the prototypes associated with the virtual camera being VPICTUREd have any arcs, V.FIT.ARCS is automatically disabled for the VPICTURE (or VWINDOW) operation. Even if the prototypes have arcs, the vision system will fit only arcs with similar radii, filtering out arcs with other radii. (The definition of "similar radii" is liberal, depending on the standard deviations of the radii of the prototype arcs and the effort levels of the prototypes.)

The V.BOUNDARIES system switch overrides the V.FIT.ARCS switch. That is, if V.BOUNDARIES is disabled, arcs will not be fit regardless of the setting of V.FIT.ARCS.

Similarly, if the V.MAX.PIXEL.VAR system parameter is 0, both line fitting and arc fitting are automatically disabled.

Related Keywords

V.BOUNDARIES (system switch)
VFEATURE (real-valued function)
VPICTURE (monitor command and program instruction)
V.RECOGNITION (system switch)

Syntax

```
... V.GAIN [camera]
```

Function

Set the gain for the incoming video (camera) signal.

Usage Considerations

Changing this parameter immediately affects the video output of the camera interface board.

This is an array of parameters—one for each virtual camera. (See the general description of parameters in the *AdeptVision User's Guide* for syntax details.)

Details

The V.GAIN parameter works with the V.OFFSET parameter to select the incoming analog video gain and offset, respectively. V.GAIN multiplies (scales) the video signal, whereas V.OFFSET shifts (translates) the video signal.

Before adjusting the values of V.OFFSET and V.GAIN, you should take a picture, compute the histogram, and study the video data displayed in the Vision display window. To take the picture and compute the histogram, enter the V+ monitor commands "VPICT (cam) 2" and "VHIST" (where "cam" is the number of the virtual camera being used) or use the mouse to make the menu selections to perform these same operations.

The goal is to have the video data fill most of the intensity range (0 to 127) without spilling over either end. If the video data spills over the left end, the histogram curve shows a spike over the "0" intensity label. Similarly, if the video data spills over the right end, the curve shows a spike at or near the "127" intensity label.

You should increase V.GAIN to expand the intensity range of the video data or decrease V.GAIN to reduce the intensity range. Similarly, you should increase V.OFFSET to shift the video data toward the left and decrease V.OFFSET to shift it toward the right. For good results, you may have to repeat the procedure of taking a picture, computing the new histogram, and adjusting V.GAIN and V.OFFSET a few times.

V.GAIN must be assigned an integer value in the range 1 to 256, inclusive. The parameter is set to 128 when the V+ and AdeptVision systems are loaded into memory from disk.

Example

Make the video gain 120 for all virtual cameras:

```
PARAMETER V.GAIN = 120
```

Related Keywords

VHISTOGRAM (monitor command and program instruction)
V.OFFSET (system parameter)

Syntax

```
VGAPS  data[i] = proto_name, edge_num
```

Function

Find the unverified gaps in a match with a prototype or subprototype.

Usage Considerations

The V.LAST.VER.DIST system parameter must be nonzero when the last VPICTURE was performed to make the necessary information available to the VGAPS instruction.

The VGAPS instruction refers to the object most recently VLOCATEd regardless of which program task executed the VLOCATE instruction. Consequently, for predictable operation, only one program task should execute VLOCATE instructions.

Parameter

`data[]` Real array containing the requested gap information:

data[i+0] = Number of gaps in the edge or edge numbers in the list

data[i+1] = Verify percentage: 0 to 100. If the parameter "`edge_num`" is 0 (see below), this is the verify percentage for the entire prototype. Otherwise, it is the verify percentage for the one edge.

data[i+2] = If "`edge_num`" is 0, this is the start of a list of the edges of the prototype or subprototype that have gaps in them. If "`edge_num`" is not zero, this is the start of a list of the millimeter ranges for the gaps in the specified edge, where "data[i+2n]" and "data[i+2n+1]" indicate the start and end of gap #n, respectively.

`i` Optional integer value that identifies the first array element to be defined in "**`data[]`**". Zero is assumed if the index is omitted. If a multiple-dimension array is specified, only the right-most index is incremented as the values are assigned.

`proto_nam` Optional string expression that specifies the name of the prototype or subprototype for which gap information is requested. If a subprototype is specified, the string must have the form "name1:name2", where "name1" is the name of the prototype and "name2" is the name of the prototype's subprototype. The default, if "proto_name" is not specified, is the prototype last VLOCATEd.

`edge_num` Optional real-valued expression that specifies the number of the prototype or subprototype edge for which information is requested. The default is 0, which requests a list of all the edge numbers that have gaps in them. If the value is nonzero, gap information about a specific edge is requested. The edge number is relative to the prototype's edge numbering, unless "`proto_name`" specifies a subprototype. Then, the "`edge_num`" is relative to the **sub**prototype's edge numbering.

Details

When a prototype object is recognized in the image, each edge of the prototype is compared with the edges in the image to verify the prototype match. When the V.LAST.VER.DIST system parameter is nonzero, the portions of prototype edges that are not verified are remembered. These unverified portions are called gaps. Gaps could be caused by cutouts, flashing, occluding debris, or by dimensions that are out of tolerance.

To identify all the gaps in a prototype or subprototype, the prototype must first be VLOCATEd. Then use VGAPS with an "edge_num" value of 0. This returns into "data[]" a list of the edges that have gaps in them. Then use VGAPS again with these edge numbers to find where the gaps are within each particular edge. A gap is defined by a pair of distances (in millimeters) from the start of the edge (line or arc). The portion of the edge between the pair of distances is a nonverified gap. As their numbering indicates, the edges of each region in a prototype are ordered in a clockwise direction. The distances in the "**`data[]`**" array returned by VGAPS use the same direction. If a pair of distances in "**`data[]`**", indicating a gap, are 0 and 5.2, then the gap starts at the first corner of the edge (moving clockwise around the region), extending into the edge 5.2 millimeters. Circular edges begin at 0 degrees, the point on the circle that is farthest right in the Vision display window.

If only some of the edges of the prototype (or subprototype) are of interest, use the VSHOW monitor command to display all of the edge numbers for the prototype. Then use VGAPS with each edge number of interest.

VGAPS provides the requested information only if it refers to the most recent picture taken (via VPICTURE) and the most recent object located (via VLOCATE). Also, the V.LAST.VER.DIST system parameter must be nonzero when the VPICTURE operation is performed. Otherwise, the gap information is not retained and VGAPS results in the error message "*Information not available*".

Example

Before this program is called, the parameter V.LAST.VER.DIST must be nonzero, a VPICTURE must be performed, and a VLOCATE of the prototype must succeed. The name of the prototype must be passed to the "$proto" parameter of this program.

The second argument to VLOCATE (”vloc”) must have been specified and then passed to this program. This is a transformation variable that is assigned the location of the object.

```
.PROGRAM label_gaps($proto, vloc)

;ABSTRACT:    This program displays, in the Monitor display window,
;     information about all the gaps in a recognized prototype and
;     puts a "G" at the center of each gap in the Vision display
;     window.
;
;INPUT PARAMS: $proto - name of the prototype to examine
;                   vloc - transformation variable that is assigned the
;                         location of the object
;
     LOCAL cang, cnt, d360_circum, dx, dy, einfo[], enum
     LOCAL gaps[], gcx, gcy, glist[], gnum, mid, xta[]
     LOCAL $type

; Get a list of all the prototype edges that have gaps.

     VGAPS glist[] = , 0
     TYPE "Total verify percentage: ", glist[1]

; Display a description of each prototype edge that has a gap.
; Then display the gap ranges and label the gaps in the vision
; display window.

     FOR cnt = 1 TO glist[0]
          enum = glist[cnt+1]

; Get information about the prototype edge from VEDGE.INFO.
; Then get information about the edge gaps from VGAPS.

          VEDGE.INFO einfo[] = $proto, enum
          VGAPS gaps[] = , enum
          TYPE /C1, " Corner: " einfo[2], ",", einfo[3]

          IF einfo[0] == 0 THEN                  ;If line
               TYPE enum, ") Line. ", gaps[1], "% verified."
          ELSE                                   ;Else arc

               IF einfo[0] > 0 THEN
                    $type "Convex"
               ELSE
                    $type "Concave"
               END

               TYPE enum, ") ", $type, " arc.", /S
               TYPE " Radius: ", einfo[8], /S
               TYPE ", center: ", einfo[6], ",", einfo[7], /S
               TYPE ", ", gaps[1], "% verified."
               d360_circum = 360/(2*PI*einfo[8])
```

```
                cang = ATAN2(einfo[3]-einfo[7],einfo[2]-einfo[6])
           END

; For each gap in the edge, type out its range
; and label the gap with a "G".

           TYPE "         Gaps: ", /S
           FOR gnum = 1 TO gaps[0]
                IF gnum > 1 THEN
                     TYPE ", ", /S
                END
                TYPE gaps[gnum*2], " -->", gaps[gnum*2+1], /S
                mid = (gaps[gnum*2]+gaps[gnum*2+1])/2
                IF einfo[0] == 0 THEN
                     dx = einfo[4]-einfo[2]            ;Line
                     dy = einfo[5]-einfo[3]
                     mid = mid/SQRT(SQR(dx)+SQR(dy))
                     gcx = einfo[2]+dx*mid
                     gcy = einfo[3]+dy*mid
                ELSE                                   ;Arc
                     IF einfo[0] > 0 THEN
                          mid = cang-mid*d360_circum
                     ELSE
                          mid = cang+mid*d360_circum
                     END
                     gcx = einfo[6]+einfo[8]*COS(mid)
                     gcy = einfo[7]+einfo[8]*SIN(mid)
                END
                DECOMPOSE xta[] = vloc:TRANS(gcx,gcy,0)
                GRANS (vwin,1)
                GTYPE (vwin) xta[0], xta[1], "G"
           END
           TYPE /C1, " Corner: ", einfo[4], ",", einfo[5]
      END
.END
```

Related Keywords

VDEF.SUBPROTO (program instruction)
VSHOW (monitor command)
VSHOW (program instruction)
VSUBPROTO (program instruction)
V.LAST.VER.DIST (system parameter)

Syntax

```
VGET.AOI array[i] = aoi
```

Function

Return the definition of an area-of-interest.

Usage Considerations

The area-of-interest must have been defined with a VDEF.AOI instruction.

Parameters

array[]	Variable name identifying the array to receive the AOI definition.
i	Optional integer value specifying the starting array index.
aoi	Real-valued expression identifying the area-of-interest.

Details

If the VGET.AOI instruction specifies a valid AOI number, the following information is returned in "array[ ]":

array[i]	Returns the shape number if definition is valid; returns –1 if definition is not valid
array[i + 1]	Dimension 1 of AOI
array[i + 2]	Dimension 2 of AOI
array[i + 3]	Dimension 3 of AOI
array[i + 4]	Dimension 4 of AOI
array[i + 5]	Angle 1 of AOI
array[i + 6]	Angle 2 of AOI

Example

```
Change dim3 of aoi 3 to 37.4:
VGET.AOI def[] = 3000
IF def[0] <> -1 THEN
   VDEF.AOI 3000 = def[0], def[1], def[2], 37.4, def[4], def[5]
END
```

Related Keyword

VDEF.AOI (program instruction)

Syntax

VGETCAL (cam) **scalers[i]**, pmm.to.pix[j,k], pix.to.pmm[l,m], to.cam

Function

Ask the system to fill in arrays with the previously defined vision calibration data for a given virtual camera.

Parameters

cam	Optional real-valued expression that specifies the virtual camera number.
scalers[]	Real array that receives the scaler calibration values.
pmm.to.pix[,]	Optional, two-dimensional real array that receives the millimeter-to-pixel transformation matrix. If this array is specified, the array "pix.to.pmm[,]" must also be specified.
pix.to.pmm[,]	Optional, two-dimensional real array that receives the pixel-to-millimeter transformation matrix. If this array is specified, the array "pmm.to.pix[,]" must also be specified.
i j,k l,m	Optional integer values that identify the first array element to be defined in the respective array. Zero is assumed for each index that is omitted. If an array is specified that has more dimensions than needed, only the right-most indexes are incremented as the values are assigned.
to.cam	Optional variable that recieves the vision transformation.

Details

This instruction is used by the calibration programs supplied with the vision system. The calibration programs perform a VGETCAL to determine what calibration data was previously defined via VPUTCAL.

On power-up, the vision system preassigns calibration data for all virtual cameras. Virtual cameras are associated with physical cameras in blocks of four; virtual cameras 1-4 are associated with physical cameras 1-4, virtual cameras 5-8 are associated with physical cameras 5-8, etc. For each virtual camera, one millimeter is made equal to one pixel, and perspective calibration is not used. This makes the system usable for setup or experimentation. For precision work, however, a calibration procedure should be performed.

Since the calibration arrays are normally filled by an Adept V+ utility program, the programmer should not have to be completely familiar with the individual array elements. However, the contents of the arrays are listed in this manual in the description of the VPUTCAL instruction.

Example

```
VGETCAL (3) cal[], mp[,], pm[,] ;Get cam calibration data
```

Related Keyword

VPUTCAL (program instruction)

Syntax

VGETPIC (cam, type, s_rate, s_mode) **$pic[**r,c**]** = shape, x0, y0, dx, dy

Function

Read all or part of an image into a string array.

Usage Considerations

The VISION switch must be enabled, the vision processor must be idle, and vision model training must not be active for this instruction to be executed.

Parameter

cam — Optional real-valued expression that specifies a virtual camera number. The default is 1. (This parameter is currently ignored.)

type — Optional real-valued expression indicating the type of data to store in the array "**$pic[,]**". The default is 1.

1 = Grayscale image and binary (or edge) image
2 = Binary (or edge) image only

s_rate — Optional real-valued expression specifying the sampling rate for reading pixels. If the rate is N, every Nth pixel is read out of every Nth row. The rate must be in the range 1 to 100. The default is 1.

s_mode — Optional real-valued expression specifying the sample mode, which is meaningful only if the sample rate (s_rate) is greater than 1. Pixels are simply sampled if "s_mode" is 0 (the default). Pixels are averaged if "s_mode" is 1, in which case the binary portion of the image will be stripped.

NOTE: The parentheses in the instruction syntax can be omitted if all four of the above parameters are omitted.

$pic[,] — Array into which to put a header string and picture data.

r,c — Row and column indexes into the array "**$pic[,]**", indicating where the picture and header information is to be stored. The defaults are 0 for both "r" and "c". The header string is stored in the element "$pic[r,c]". The picture data is stored starting at element "$pic[r+1,c+1]".

shape — Optional real-valued expression indicating the shape of the image to read. The default is 1, indicating a rectangular shape (which is the only shape now available).

`x0, y0` Optional real-valued expressions specifying the coordinate of the lower-left corner of the image to be read. The coordinate is in pixel units and must be within the image. The default is (1,1), the lower-left corner on the Vision display window.

`dx, dy` Optional real-valued expressions specifying the width and height of the image to read, in pixel units. The image dimensions are listed in the table above. If the width or height of the image window being read exceeds the image dimensions on the top or right, the dimensions are automatically reduced and the new dimensions are stored in the header string. If not specified, "`dx`" and "`dy`" default to the full image size. (For grayscale VGETPICs, "`dx`" is limited to 512 pixels.)

Details

This instruction reads a rectangular window of an image and stores it into a two-dimension string array. Unless the data is packed (see below), each string in the array holds 128 pixels. As many strings as needed are filled for each row (four is the maximum required). There is one row of strings for each row of the image window being read. A header string is stored in "**`$pic[r,c]`**" (normally the [0,0] location), which contains pertinent information about the stored image. See below.

Type #1 VGETPIC reads the image in the frame store currently selected. (See the VSELECT program instruction.) This includes both the grayscale data (lower 7 bits of each byte) and the binary or edge data (high bit of each byte). A type #2 VGETPIC reads only the binary or edge image—the image you see when in VDISPLAY mode 2.

The image data is binary for type #2 VGETPIC. Therefore, for efficiency, the data is packed eight pixels to a string character, with the left-most pixel in the least-significant bit. No partial characters are packed, so "`x0`" and "`dx`" are automatically adjusted to have the image window read start and stop on an eight-pixel boundary ("`dx`" ends up being a multiple of eight). The actual starting coordinates and dimensions used are stored in the header string.

> **NOTE:** The only required parameter is "**`$pic[,]`**". All the other parameters default so that the entire image is saved.

The sample rate (`s_rate`) and sample mode (`s_mode`) parameters are supported for the image frame stores. When the sample rate is greater than one, the image stored in "**`$pic[,]`**" is significantly smaller. The sample mode determines the method of image reduction. For example, if the sample rate is 2, the image is shrunk by a factor of 2 in both the X and Y directions. If the sample mode is 0, the image is reduced simply by using every other pixel on every other line. If the

sample mode is 1, each pixel in the reduced image is the average of the pixels in a 2x2 neighborhood. Likewise, if the sample rate is 6, mode 0 would use every 6th pixel on every 6th line, and mode 1 would determine each pixel of the reduced image to be the average over a 6x6 neighborhood.

Use of the sampling feature can greatly reduce the amount of memory needed to store an image. A sample rate of 2 results in an image that is 1/4 the size of the original. Sampling may be used to save several reduced images for simultaneous display (see the description of VPUTPIC later in this chapter).

Normally, you never have to examine or alter the header string. However, for completeness, its contents are described in **Table 2-4** below. The VPUTPIC instruction requires that this information be present and consistent with the picture data stored in the rest of the array. Therefore, you should not change the header string without a full understanding of the format. The items in the header string are all integer values. They may be extracted using the V+ INTB function.

Table 2-4. Contents of VGETPIC/VPUTPIC Header String

Start Char	Item	Description
1	version	Version number for maintaining compatibility
3	system	Type of system: 2 for area grayscale systems
5	camera	Virtual camera accessed by VGETPIC
7	type	Type of image data: 1, 2, [3, 4] (pre-11.0 images only)
9	packed	0 for 1 pixel per byte; 1 for 8 pixels per byte
11	sample_rate	Sample rate used
13	sample_mode	0 for sampling; 1 for averaging
15	shape	1 for rectangular
17	x0	Actual x0 of starting location
19	y0	Actual y0 of starting location
21	dx	Actual number of image columns read and stored
23	dy	Actual number of image rows read and stored
25	chars_per_row	Number of characters per row used in $pic[,]
27	clipped	Boolean: TRUE if clipped in either direction

Example

```
VGETPIC $savpic[,] ;Save the current full image
   .
   .
   .
VPUTPIC $savpic[,] ;Restore the image saved above
```

See the description of VPUTPIC for more examples.

Related Keywords

VDISPLAY (monitor command and program instruction)
VPUTPIC (program instruction)
VSELECT (program instruction)

Syntax

VGET.TRANS array[i]

Function

Return the value of the current vision transformation.

Usage Considerations

Each task has its own vision transformation, so VGET.TRANS must be issued in the correct task.

Parameters

array[]	Variable identifying the array to receive the vision transformation.
i	Optional integer expression specifying the starting array index. The default is 0.

Details

The current vision transformation definition is returned in the array elements:

array[i]	Always returns 1
array[i + 1]	X offset of the vision transformation
array[i + 2]	Y offset of the vision transformation
array[i + 3]	Rotation of the vision transformation
array[i + 4]	Scale of the vision transformation

Example

Rotate the current vision transformation by 90°:

```
VGET.TRANS t_def[]
VDEF.TRANS t_def[1], t_def[2], t_def[3]+90
```

Related Keyword

VDEF.TRANS (program instruction)

Syntax

```
VHISTOGRAM (dmode) array[index] = ibr
```

Function

Compute the histogram for a grayscale frame store.

Usage Considerations

The VISION switch must be enabled, the vision processor must be idle, and vision model training must not be active for this instruction to be executed.

Parameters

`dmode`	Optional real-valued expression specifying the display mode: 1 to display the histogram, or 0 not to display it. This parameter defaults to 1 (do display) if no value is specified. In that case, the parentheses can be omitted.
`array[]`	Optional array into which the values are placed. Elements [index] to [index+127] are filled with the pixel counts for each of the possible intensity values, 0 to 127.
`index`	Optional integer value that identifies the first array element to be defined in "`array[]`". Zero is assumed if the index is omitted. If a multiple-dimension array is specified, only the right-most index is incremented as the values are assigned.
`ibr`	Optional integer value that specifies the image buffer region (area-of-interest and frame store) to use. The area-of-interest must have been defined with a VDEF.AOI instruction. The default frame is the current frame—the one selected most recently via VPICTURE, VSELECT, or by a menu pick using the mouse.

Details

VHISTOGRAM computes a histogram for an image currently in memory. The image is visible in VDISPLAY mode #1. The histogram is computed by simply reading all the pixels in the image defined by "`ibr`" and counting the number of pixels at each intensity level. These counts are placed in the array if that optional parameter was specified.

The image memory read may be any valid image buffer region. The region must contain a valid picture. Otherwise, an error results. (See the VSELECT program instruction.) Image histograms are useful for determining the gain (V.GAIN) and offset (V.OFFSET) for the incoming camera data. Histograms also help determine the thresholds for binary image processing.

When the histogram is displayed in the Vision display window, the range of intensity is represented by the horizontal axis and the number of pixels per intensity is represented along the vertical axis. The vertical axis is scaled to accommodate the highest peak. That axis is not labeled because usually only the relative magnitudes are important.

The following text is displayed to the side of the histogram:

```
Min: aaa
Max: bbb
```

where "aaa" is the minimum intensity with a nonzero count of pixels and "bbb" is the maximum intensity with a nonzero count.

Note that the VWINDOWI program instruction can return the same information as VHISTOGRAM. VWINDOWI does not display the histogram, but it can be executed when the vision system is not in an idle state.

Example

Take a quick picture:

```
VPICTURE 2
```

Compute and display the histogram:

```
VHISTOGRAM
```

Related Keywords

VDEF.AOI (program instruction)
VAUTOTHR (monitor command and program instruction)
VWINDOWI (program instruction)
V.GAIN (system parameter)
V.OFFSET (system parameter)

Syntax

```
... V.HOLES [camera]
```

Function

Enable or disable the accounting of interior features in all objects.

Usage Considerations

A change to this switch takes effect when the next VPICTURE command or instruction, or VWINDOW instruction, is executed.

The setting of this switch is ignored by the vision system when the V.DISJOINT switch is enabled.

This is an array of switches—one for each virtual camera. (See the general description of switches in the ***AdeptVision User's Guide*** for syntax details.)

Details

If the V.HOLES switch is enabled, the system keeps track of the holes in all objects, whether or not the objects have been recognized. The description of a hole is available by executing the VLOCATE instruction, followed by use of the VFEATURE function. Holes are described by their area, bounding box, and their parent, child, and sibling relationships with other holes. Also, if an object has been recognized, the VFEATURE function reports which holes helped verify the presence of the object in the image.

Determining hole locations may be useful for inspection or for implementing unusual recognition strategies. As an example of inspection, components missing from printed circuit boards may be discovered by backlighting the boards and looking for unexpected holes.

The memory used for storage of the hole information is taken away from the memory normally allocated for the object queue. The object queue holds the objects seen until VLOCATE instructions are processed. Every satisfied VLOCATE instruction removes one object from the object queue, leaving room for a new object. (The object queue is displayed when the VQUEUE monitor command is issued.)

The vision system queue has a maximum capacity of 1200 objects and holes. Each hole in the queue takes the place of one object.

Related Keywords

VFEATURE (real-valued function)
VLOCATE (program instruction)
V.DISJOINT (system switch)

Syntax

```
... V.IO.WAIT [camera]
```

Function

Enable the synchronization of taking pictures (VPICTUREs) with an external event that triggers the fast digital-input interrupt line.

Usage Considerations

Operation of the external trigger can be configured with the Adept controller configuration program (in the file CONFIG_C.V2 on the Utility Disk). See the *V+ Language User's Guide* for details on digital I/O.

A change to this parameter takes effect when the next VPICTURE command or instruction is executed.

This is an array of parameters—one for each virtual camera. (See the general description of parameters in the *AdeptVision User's Guide* for syntax details.)

Details

If the V.IO.WAIT parameter is set to 1, VPICTURE commands or instructions wait for an interrupt from the fast digital-input interrupt line before acquiring an image. This is useful for taking pictures of fast-moving objects. For example, the object could trigger a simple sensor, such as a photoelectric cell, that is wired into the V+ controller.

> **NOTE:** When taking pictures of moving objects, a strobe light or shuttered camera should be used to minimize image blur. (For more information on the use of strobe lights, see the system switch V.STROBE and the system parameter V.SYNC.STROBE in this manual. For information on the various camera types, see the *AdeptVision User's Guide*.)

The fast digital-input interrupt line provides a significant improvement in response time to external events. When this line is used, the worst-case delay from the time the line is triggered until the strobe light output signal is fired is about 10 microseconds. If, instead, a V+ program uses the WAIT instruction to wait for a digital signal before performing a VPICTURE, the delay can be as long as 16 milliseconds.[1] The worst-case delay should be planned for if objects in the scene are moving fast with respect to the camera field of view.

[1] This worst-case delay assumes the Adept system controls a robot, has multiple execution tasks active, or has busy device drivers. If the system has no robot, only a single execution task running, and no busy device drivers, the delay can be as long as 2 milliseconds.

What is the effect of a worst-case delay of 16 milliseconds? Consider, for example, a field of view that is 5.12 centimeters wide. Each pixel, then, is 0.1 millimeter wide.[1] If the objects in the field of view are moving at the (high) speed of 1 meter per second, a worst-case delay of 16 milliseconds corresponds to a shift of 160 pixels.[2] Since the frame store is 512 pixels wide, a shift of 160 pixels could shift part of the object being analyzed out of the field of view.

When the V.IO.WAIT parameter is set to 1, a VPICTURE operation could wait indefinitely for the external trigger. However, the VPICTURE instruction has a special no-wait mode that may be used to allow the application program to continue without waiting for the VPICTURE operation to begin. (Also note that the VPICTURE monitor command or a program containing the VPICTURE instruction may be aborted in various ways.)

When V.IO.WAIT is set to 0 (the default), VPICTURE operations do not wait for the external trigger. This parameter may be assigned only the values 0 or 1.

Example

Wait for the external trigger when taking pictures with camera #1:

```
PARAMETER V.IO.WAIT[1] = 1
```

Related Keywords

VPICTURE (monitor command and program instruction)
VABORT (monitor command and program instruction)
VWAIT (program instruction)
V.STROBE (system switch)
V.SYNC.STROBE (system parameter)

[1] 5.12 cm per 512 pixels = 0.01 cm / pixel = 0.1 mm / pixel

[2] (0.016 second) * (1000 mm / second) / (0.1 mm / pixel) = 160 pixels

Syntax

```
... VISION
```

Function

Enable the entire vision system.

Usage Considerations

This switch must be enabled before any vision commands or instructions may be executed.

Details

The vision processor initializes itself when the VISION system switch is enabled. The vision processor automatically enables VISION on power-up.

If the switch is already enabled and a picture is being processed (a VPICTURE or VWINDOW has been issued earlier), typing “ENABLE VISION” makes the vision system stop processing images and reinitialize itself. When VISION is enabled, the vision system performs an equipment check. If there is a failure, the V^+ system displays an error message in the Monitor display window, such as “*Camera interface board absent*”. When the VISION switch is disabled, the vision system closes the vision display window, removing the menu selections. When enabled, the vision system reopens the window and restores the menu selections, assuming that a user task does not have the window opened for read / write.

Example

Restart the vision system:

```
ENABLE VISION
```

Syntax

```
... V.LAST.COL [camera]
```

Function

Set the number of the last column of pixels to be processed.

Usage Considerations

A change to this parameter takes effect when the next VPICTURE or VWINDOW is executed.

This is an array of parameters—one for each virtual camera. (See the general description of vision parameters in the *AdeptVision User's Guide* for syntax details.)

Details

This parameter, together with V.FIRST.COL, is used to set the range of pixel columns that are processed during VPICTURE and VWINDOW operations. Columns outside the range specified are ignored by the vision system. (VDEF.AOI is the preferred method for defining a processing border for VPICTURE and VWINDOW.)

The value of the parameter V.LAST.COL must be greater than the value of V.FIRST.COL and less than or equal to 640.[1] The parameter V.LAST.COL is set to its maximum value when the V+ and AdeptVision systems are loaded into memory from disk.

Example

Process all image data up to the 256th column of pixels:

```
PARAMETER V.LAST.COL = 256
```

Related Keywords

VDEF.AOI (program instruction)
V.FIRST.COL (system parameter)
V.FIRST.LINE (system parameter)
V.LAST.LINE (system parameter)

[1] This is the maximum setting. The effective value of V.LAST.COL is limited by the virtual frame allocation made with the DEVICE instruction or the CONFIG_C utility.

Syntax

```
... V.LAST.LINE [camera]
```

Function

Set the number of the last line of pixels to be processed.

Usage Considerations

A change to this parameter takes effect when the next VPICTURE or VWINDOW is executed.

This is an array of parameters—one for each virtual camera. (See the general description of parameters in the *AdeptVision User's Guide* for syntax details.)

Details

This parameter, together with V.FIRST.LINE, is used to set the range of pixel lines that are processed during VPICTURE and VWINDOW operations. Lines outside the range specified are ignored by the vision system. (VDEF.AOI is the preferred method for defining a processing border for VPICTURE and VWINDOW.)

The value of the parameter V.LAST.LINE must be greater than the value of V.FIRST.LINE and must be less than or equal to 480.[1] The parameter V.LAST.LINE is set to its maximum value when the V+ and AdeptVision systems are loaded into memory from disk.

Example

Process all camera images up to the last line of data:

```
PARAMETER V.LAST.LINE = 480
```

Related Keywords

VDEF.AOI (program instruction)
V.FIRST.COL (system parameter)
V.FIRST.LINE (system parameter)
V.LAST.COL (system parameter)

[1] This is the maximum setting. The effective value of V.LAST.LINE is limited by the virtual frame allocation made with the DEVICE instruction or the CONFIG_C utility. In field-acquire mode, lines are numbered from 1 to 240. Thus, in that mode the maximum effective value of V.LAST.LINE is 240.

Syntax

```
... V.LAST.VER.DIST [camera]
```

Function

Enable an extra verification of prototype-to-image matches and specify the pixel tolerance to use when determining boundary coincidence.

Usage Considerations

A change to this parameter takes effect when the next VPICTURE command or instruction, or VWINDOW instruction, is executed.

This is an array of parameters, one for each virtual camera. (See the general description of parameters in the *AdeptVision User's Guide* for syntax details.)

Details

When this parameter is nonzero, it enables an extra verification of each prototype-to-image match. Furthermore, the value of V.LAST.VER.DIST is used (in place of the value of the parameter V.MAX.VER.DIST) during the extra verification as the distance criterion in the coincidence test (that is, when the positions and orientations of prototype edges are compared with those of nearby edges in the image).

In addition, when V.LAST.VER.DIST is nonzero, information about the final verification is saved for use by the program instructions VGAPS and VSUBPROTO.

The VGAPS instruction provides information about unverified gaps in prototype edges. When this kind of information is needed, the parameter V.LAST.VER.DIST may be assigned a smaller value than V.MAX.VER.DIST, thereby specifying a tighter tolerance test. However, it should not be assigned a value smaller than V.MAX.PIXEL.VAR, which is the pixel variance allowed when fitting the edges (lines and arcs) in the image.

With the VSUBPROTO instruction, the verified percentages of edges of subprototypes or an individual prototype may be determined. VSUBPROTO may also be used to refine the position of the recognized prototype, based on a subprototype or a single prototype edge. In this latter case, the V.LAST.VER.DIST parameter may be assigned a value larger than V.MAX.VER.DIST, in order to "reach out" to edges that are suspected of deviating in position.

Like V.MAX.VER.DIST, V.LAST.VER.DIST is in pixel units. V.LAST.VER.DIST must be assigned a real value in the range 0 to 16, inclusive. The parameter is set to 0 when the V^+ and AdeptVision systems are loaded into memory from disk. That is, the extra verification is disabled.

Examples

```
PARAMETER V.LAST.VER.DIST[vcam] = 0
PARAMETER V.LAST.VER.DIST[2] = PARAMETER(V.MAX.VER.DIST[1])
```

Related Keywords

VGAPS (program instruction)
VSUBPROTO (program instruction)
V.MAX.VER.DIST (system parameter)

Syntax

```
VLOAD  file_spec
```

Function

Load vision models (ObjectFinder models, prototypes, Optical Character Recognition fonts, or correlation templates) from a disk file.

Usage Considerations

VLOAD will not load files created prior to version 11.0.

Parameter

`file_spec` Specification of the disk file from which the vision models are to be loaded. This consists of an optional physical device, an optional disk unit, an optional directory path, a file name, and an optional file extension. Uppercase or lowercase letters can be used.

The current default device, unit, and directory path are considered as appropriate (see the DEFAULT command in the *V+ Operating System Reference Guide*).

If no filename extension is specified, the extension ".VS" is appended to the name given.

Details

All the vision models previously VSTOREd in the given file are loaded and added to those already in memory. If the file contains a vision model with the same name, font number, or template number as one already in the vision system, loading is aborted. That is, none of the vision models in the file are loaded.

Any subprototypes associated with a prototype are also restored automatically.

As the VLOAD command is processed, V+ displays in the Monitor display window the names of the models, along with their virtual camera associations. If fonts or templates are loaded, their identifying numbers are displayed.

Example

The following monitor command loads the models from the disk file named "OBJECTS.VS" and displays their virtual camera associations as shown.

```
VLOAD objects
          0    5   10   15   20   25   30
          |....|....|....|....|....|....|..
CASTING    ***-----------------------------
FLANGE     *------------------------------
CRANK      -*--*-*-----------------------*
BRACKET    ------*---------------*---------
```

Related Keywords

VSHOW.MODEL (program instruction)
VTRAIN (program instruction)
VTRAIN.MODEL (program instruction)
VLOAD (program instruction)
VSTORE (monitor command)
VSTORE (program instruction)
VTRAIN.FINDER (program instruction)

Syntax

VLOAD (lun) `$file_spec`

Function

Load vision models (ObjectFinder models, prototypes, Optical Character Recognition fonts, or correlation templates) from a disk file.

Usage Considerations

VLOAD will not load files created prior to version 11.0.

Parameters

lun Real-valued expression that specifies the logical unit number to be associated with the operation. This must be one of the logical unit numbers for a disk device (see the ATTACH instruction in the *V+ Language Reference Guide*). The logical unit number used must not already be in use by the program for another disk access.

`$file_spec` String expression that specifies the disk file from which the vision models are to be loaded. This consists of an optional physical device, an optional disk unit, an optional directory path, a file name, and an optional file extension. Uppercase or lowercase letters can be used.

The current default device, unit, and directory path are considered as appropriate (see the DEFAULT command in the *V+ Operating System Reference Guide*).

If no filename extension is specified, the extension ".VS" is appended to the name given.

Details

All the vision models previously VSTOREd in the given file are loaded and added to those already in memory. If the file contains a vision model with the same name, font number, or template number as one already in the vision system, loading is aborted. That is, none of the vision models in the file are loaded.

Any subprototypes associated with a prototype are also restored automatically.

Vision models already in RAM cannot be overwritten by new models with the same name. They must be first be VDELETEd or VRENAMEd.

The IOSTAT real-valued function can be used after this instruction to determine if any error occurred during the load operation (see the *V+ Language User's Guide* for details).

> **NOTE:** The application program must not have attached the logical unit, since the VLOAD instruction automatically attaches and detaches the logical unit.

Example

The following program instruction loads the models stored in the disk file named "OBJECTS.VS" on the default system disk. Logical unit number 6 is associated with the operation and is used to check for successful completion:

```
VLOAD (6) "objects"
IF IOSTAT(6) < 0 THEN
     TYPE /C1, "VLOAD failure: ", $ERROR(IOSTAT(6)), /C1
     HALT
END
```

Related Keywords

VLOAD (monitor command)
VSTORE (monitor command
VSTORE (program instruction)

Syntax

VLOCATE (camera, mode, order) $name, trans_var

Function

Identify and locate an object in the scene.

Parameter

`camera` Optional real-valued expression that specifies the virtual camera number. All cameras are implied if the camera number is 0.

`mode` Optional bit-field expression indicating whether a particular object is being sought and whether the VLOCATE should wait if necessary. (See below for details.)

`order` Optional real-valued expression for selecting objects by size or position in the image. The default is 0, for no specific order desired. (See below for details.)

NOTE: This parameter has certain restrictions for the ObjectFinder tool. See the table on page 162 for details.

`$name` String variable to be assigned the name of the object found ("find-any" mode) or a string expression representing the name of the object to be located ("find-particular" mode). (This parameter must be omitted in find-hole mode [see below]).

`trans_var` Optional transformation variable to be assigned the location of the object. (See details below.)

Details

The vision system keeps a queue of the recognized objects and unrecognized regions found during image processing. The queue is filled by doing VPICTURE or VWINDOW instructions and is emptied using the VLOCATE instruction.[1] Each VPICTURE and VWINDOW operation clears the vision queue for the specified virtual camera and fills the queue with the results of the new image.

An object must be VLOCATEd in order to obtain subprototype information (see the VSUBPROTO instruction) or gap information (see the VGAPS instruction).

[1] VLOCATE retrieves one object from the queue. After each VLOCATE, the VFEATURE real-valued function may be used to obtain information about the found object.

NOTE: Subprototype information and gap information does not apply to the ObjectFinder tool.

The mode bits are defined as follows (they all default to zero).

Bit 1 (LSB) Wait (0) versus No-wait (1) (mask value = 1)

Wait (0): The VLOCATE instruction will stop program execution until any region is located, or a specific prototype is recognized, depending on the find-any / find-particular bit (see below). Once the VLOCATE is satisfied, control returns to the V+ program, even if image processing is not yet complete (that is, there are more regions to process). If the VLOCATE is not satisfied, program execution continues when the entire image has been processed.

No-wait (1): Program execution will continue without waiting.
Default: Wait mode (0).

Bit 2 Find-any (0) versus Find-particular (1) (mask value = 2)

Find-any (0): Any object located for the given virtual camera will satisfy the VLOCATE (the next one in the queue is returned in the desired "`order`"). The "`$name`" parameter must be a string variable, which is filled in with the name of the object found.

Find-particular (1): If this bit is set to one, a particular object is sought. The name of the object to be found must be given. The name "?" may be given if an unknown region is sought.

Default: Find-any mode (0).

Bit 3 Find-object (0) versus Find-hole (1) (mask value = 4)

NOTE: Find-hole does not apply to the ObjectFinder tool.

Find-object (0): This is the normal mode of operation where recognized objects or unrecognized regions are located.

Find-hole (1): In this mode, the VLOCATE is a request for hole information about the object last located. This mode provides a way to inspect for unexpected holes in objects. In order for this find-hole mode to be used, an object first has to be VLOCATEd in the normal find-object mode. The subsequent find-hole VLOCATEs refer to the object found. Each find-hole VLOCATE returns one hole of the object. The information provided by the VFEATURE function is sufficient to reconstruct the parent, child, and sibling relationships between the holes, in case there are holes within holes.

If two or more objects are touching or overlapping, they have the same set of holes because they are all part of the same outer region. Consequently, the same hole may be VLOCATEd in find-hole mode multiple times—once after each object is recognized. The duplicate hole descriptions are the same, except for the "flags" bit returned by VFEATURE that indicates if the hole boundary contributed to the recognition of the object.

VLOCATE in find-hole mode refers to the object most recently VLOCATEd, regardless of which program task executed the VLOCATE instruction. Consequently, there is possible confusion when more than one program task performs VLOCATEs. Thus, for predictable operation with find-hole requests, applications should be organized to have only one program task execute VLOCATE instructions.

The other VLOCATE mode bits are ignored in find-hole mode. That is, no-wait mode and find-any mode are implicitly in effect. Also, the "name" parameter to VLOCATE **must** be omitted when find-hole mode is specified.

VLOCATE in find-hole mode succeeds only after a VPICTURE or VWINDOW processed with the V.HOLES switch enabled and the V.DISJOINT switch disabled.

Default: Find-object mode (0).

The "`order`" parameter to VLOCATE allows you to locate objects in the image in a preferred order: biggest first, left-most first, etc. This applies to recognized objects, unrecognized regions, and holes, depending on the mode of the VLOCATE. The "`order`" parameter must have a value in the range 0 to 10. The values are interpreted as follows:

Value	Object Indicated
0	Any
1	Biggest first (not used for ObjectFinder)
2	Smallest first (not used for ObjectFinder)
3	Left-most, based on box center
4	Right-most, based on box center
5	Bottom-most, based on box center
6	Top-most, based on box center
7	Left-most, based on nearest box edge
8	Right-most, based on nearest box edge
9	Bottom-most, based on nearest box edge
10	Top-most, based on nearest box edge

If the "order" parameter is omitted, the value 0 is assumed, which means no particular order is desired. "order" values 3 through 6 select criteria based on the center of the object's bounding box. For example, an "order" value of 3 is a request for the object whose bounding box center is nearest the left side of the image. Values 6 through 10 select criteria based on the object's closest edge. For example, an "order" value of 8 is a request for the object whose right edge is nearest the right side of the image. When a nonzero value is specified for "order" in a VLOCATE instruction, the vision system searches through the queue of processed objects to find the one that satisfies the requested order. If the vision system has not completed processing of all the objects in the image, the unprocessed objects are not considered in the order search. If you want all the objects to be included, execute a VWAIT program instruction before the VLOCATE to ensure that image processing has completed.

The real-valued function VFEATURE(1) must be evaluated to determine if an object was located.

Memory is reserved for storing objects in the various vision queues (one queue for each virtual camera). This memory allocation can be changed with the DEVICE instruction. See the ***AdeptVision User's Guide*** for details.

The setting of the V.CENTROID switch may affect the location value returned by VLOCATE in find-any mode. If the region located is unknown (that is, the object name is "?") and the V.CENTROID system switch is enabled, the location returned is for the region centroid. If the V.CENTROID switch is disabled, the location is the center of the bounding box for the unknown region. The setting of the V.CENTROID switch has the same effect on the location returned by a VLOCATE in find-hole mode.

The V.CENTROID switch does not affect the location returned by VLOCATE in find-particular mode. In that mode, the location is always for the region centroid.

Example

Locate an instance of the prototype named OBJ found in virtual camera #3, waiting if necessary, and then assign the object location to "cx":

```
VLOCATE (3, 2) "OBJ", cx
```

Get the name and location of the next object or region in the queue in "wait" mode:

```
VLOCATE (0, ) $name, loc
```

Return the name and location of the next object in the queue for virtual camera #2, but do not wait if no object is found:

```
VLOCATE (2, 1) $name, loc
```

Related Keywords

V.BOUNDARIES (system switch)
VFEATURE (real-valued function)
VFINDER (program instruction)
VGAPS (program instruction)
V.HOLES (system switch)
VPICTURE (monitor command and program instruction)
V.RECOGNITION (system switch)
VSUBPROTO (program instruction)
VWINDOW (program instruction)

Syntax

```
... V.MAX.AREA [camera]
```

Function

Set the maximum area above which the vision system ignores regions.

Usage Considerations

A change to this parameter takes effect when the next VPICTURE command or instruction, or VWINDOW instruction, is executed.

The effects of changes to this parameter are visible in the Vision display window only in a special display mode. Regions larger than V.MAX.AREA will still appear in display modes #1 and #2.

This is an array of parameters—one for each virtual camera. See the *AdeptVision User's Guide* for details on setting parameters.

Details

Regions that contain more than V.MAX.AREA number of pixels are ignored by the vision system during object recognition, clear-grip testing, and standard ruler measurements. This parameter is less commonly useful than V.MIN.AREA. A situation in which V.MAX.AREA would be useful, however, occurs when a fixture is in the scene that is larger than the object to be recognized or measured. In that case, time can be saved by setting V.MAX.AREA smaller than the area of the fixture region but larger than the object region, because then the fixture boundary is not analyzed.

When a region area is larger than V.MAX.AREA, the region is merged into the background. Any holes of the region also become merged into the background because they are the same color as the background. However, any holes observed in the holes are kept and treated like separate regions.

> **CAUTION:** If multiple objects touch or overlap, their combined area in the image is compared with V.MAX.AREA. Consequently, if V.MAX.AREA is set to be larger than the area of one object but smaller than the combined areas of two objects, the image of two touching objects is ignored—they disappear.

This parameter must be assigned an integer value in the range 1 to 1,048,576 (1024∗1024), inclusive.[1] Furthermore, the value must be greater than or equal to the value of the parameter V.MIN.AREA. The parameter V.MAX.AREA is set to 307,200[1] when the V+ and AdeptVision systems are loaded into memory from disk.

Example

Ignore regions that are larger than 100,000 pixels in area:

```
PARAMETER V.MAX.AREA = 100000
```

Related Keywords

V.MIN.AREA (system parameter)
V.MIN.HOLE.AREA (system parameter)

[1] The effective maximum of V.MAX.AREA is the area of the virtual frame buffer. Virtual frame buffer sizes are specified with the DEVICE instruction.

Syntax

```
... V.MAX.PIXEL.VAR [camera]
```

Function

During a VFIND.LINE or VFIND.ARC operation, this parameter specifies the maximum pixel distance from the fit edge beyond which edge points may be filtered out.

During boundary analysis, this parameter sets the maximum pixel deviation allowed when fitting lines and arcs to region edges.

Usage Considerations

A change to this parameter takes effect when a new region is analyzed, or when a VFIND.LINE or VFIND.ARC instruction is executed.

The effects of changes to this parameter are visible in the Vision display window only during training or in a special display mode after a VPICTURE, VWINDOW, VTRAIN, VFIND.LINE, or VFIND.ARC operation.

This is an array of parameters—one for each virtual camera. See the *AdeptVision User's Guide* for details on setting parameters.

Details

This parameter has two different functions, depending on the image processing operation being performed. The line and arc finders, VFIND.LINE and VFIND.ARC, use it as a distance threshold for controlling the filtering of edge points. The finders optionally iterate, discarding edge points that are too far from the fit line or arc. The parameter V.MAX.PIXEL.VAR specifies the maximum distance from the fit edge within which edge points are always preserved. See the parameter V.MAX.SD on **page 170** for a more complete description of the filtering algorithm used by the finder.

As a general guideline, when the finders operate in grayscale mode, V.MAX.PIXEL.VAR should be set to 1 or larger. When operating in binary mode, it should be set to 1.5 or larger. In both cases, if it is set to less than 0.5, the finders automatically use the value 0.5.

The other function of V.MAX.PIXEL.VAR applies during boundary analysis, when a VWINDOW, VPICTURE (in mode #0 or #–1), or VTRAIN operation is executed. During boundary analysis, the vision system characterizes the regions in the image as a connected sequence of lines and arcs. The lines and arcs are an approximation of the boundary of the binary image. V.MAX.PIXEL.VAR determines the maximum number of pixels by which the lines and arcs may

deviate from the binary image boundary. When this tolerance is increased, edges are smoothed and fewer lines and arcs are generated. On the other hand, a more accurate model of the boundary is attained when this tolerance is decreased. For all practical purposes, V.MAX.PIXEL.VAR should be kept in the range 1.0 to 3.0.

If V.MAX.PIXEL.VAR is 0, both line and arc fitting are disabled. This may be useful for some applications that use the VEDGE.INFO instruction and want only the primitive edges that bound regions (that is, the edges displayed when the V.SHOW.EDGES system switch is enabled). The allowable range for this parameter is 0.0 to 8.0, inclusive. The parameter is set to 1.5 when the V$^+$ and AdeptVision systems are loaded into memory from disk.

Example

The following figure illustrates the effects on boundary analysis for three different values of V.MAX.PIXEL.VAR.

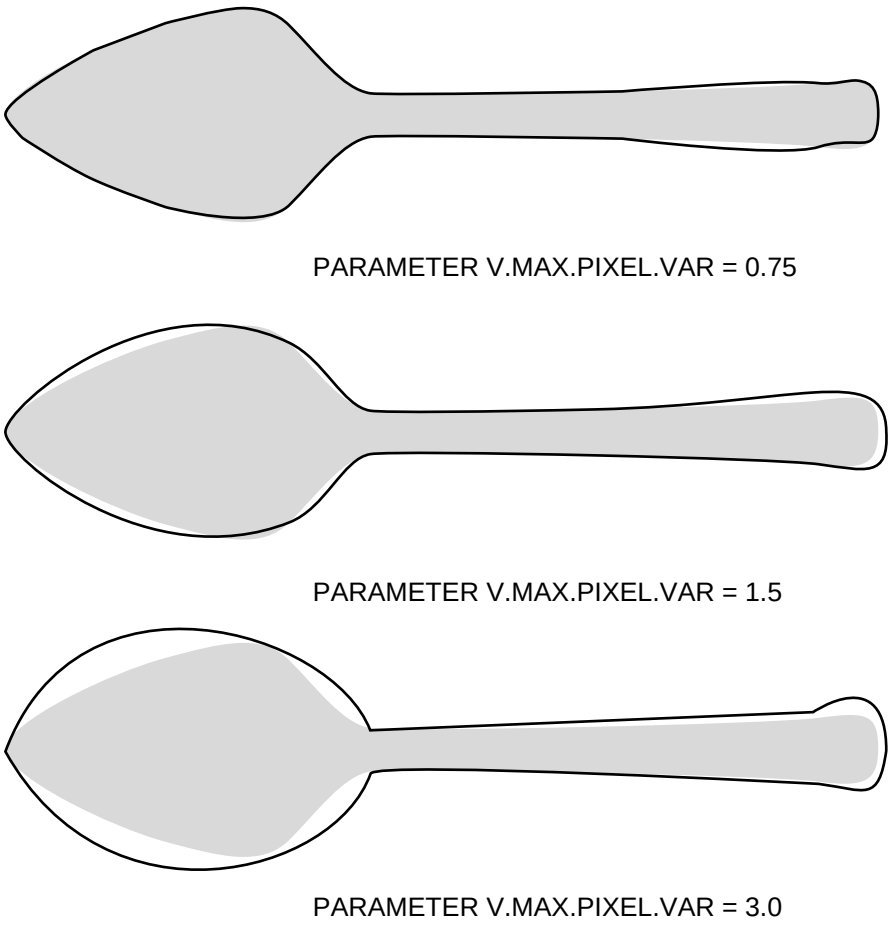


Figure 2-9. Effects of V.MAX.PIXEL.VAR Parameter

Related Keywords

VFIND.ARC (program instruction)
VFIND.LINE (program instruction)
V.MAX.VER.DIST (system parameter)
VPICTURE (monitor command and program instruction)
VTRAIN (monitor command)
VTRAIN (program instruction)
VWINDOW (program instruction)

Syntax

```
... V.MAX.SD [camera]
```

Function

Set the distance (in units of standard deviation) from the fit line or arc beyond which edge points should be filtered out.

Usage Considerations

A change to this parameter takes effect when a VFIND.LINE or VFIND.ARC program instruction is performed.

This is an array of parameters—one for each virtual camera. See the *AdeptVision User's Guide* for details on setting parameters.

Details

This parameter is used only during execution of the line and arc finders, VFIND.LINE and VFIND.ARC. If V.MAX.SD is nonzero, the finder filters out edge points that are more than V.MAX.SD standard deviations from the fit (line or arc) edge, and then the finder refits the edge. This filtering is disabled if V.MAX.SD is zero. In more detail, the filtering operation is as follows. First the finder (either VFIND.LINE or VFIND.ARC) locates edge points within its area of interest. Then the following steps are performed:

1. An edge, either a line or an arc, is fit to the edge points using a least-squares technique.
2. The distance from each edge point to the fit edge is computed, and the standard deviation of all the distances is computed.
3. Edge points farther than "d" from the fit edge are discarded, where "d" is defined as follows:

   ```
   d = MAX(V.MAX.PIXEL.VAR[cam], V.MAX.SD[cam] * sd)
   ```

   V.MAX.PIXEL.VAR is a system parameter, and "sd" is the standard deviation computed in step #2 above. V.MAX.PIXEL.VAR should be set to at least 1 when the finders operate in grayscale mode, and to at least 1.5 when the finders operate in binary mode. This distance threshold keeps the finders from discarding good edge points when the distance standard deviation ("sd") is small.
4. If no points were discarded, or too many points were discarded, processing stops. ("Too many points were discarded" means fewer than two edge points remain for fitting a line, or fewer than three edge points remain for fitting an arc.)

Otherwise, go back to step #1 to refit the edge to the remaining points.

This filtering capability is useful when “salt and pepper” noise is present in the image within the area of interest for the finder. When the noise is not close to the true edge, a single filtering iteration should remove all the noise. However, if there are many “noise edges” near the true edge, multiple iterations will be performed and some good edge points may be filtered out. An example situation where many iterations would be performed is when the line finder tries to fit a line to a nonlinear edge such as an arc.

The allowable range for V.MAX.SD is 0 to 5, inclusive. V.MAX.SD is set to 0 when V+ and AdeptVision systems are loaded into memory from disk.

Given a normal distribution of edge points, 5 standard deviations should encompass all of the edge points, so it will usually have the same effect as a V.MAX.SD value of 0—that is, causing no edge points to be filtered out. If edge filtering is desired, V.MAX.SD should be in the range 1 to 2 for the best results in most situations.

Related Keywords

VFIND.ARC (program instruction)
VFIND.LINE (program instruction)
V.MAX.PIXEL.VAR (system parameter)

Syntax

```
... V.MAX.TIME [camera]
```

Function

Set the maximum time allowed for the vision system to analyze a region during object finding, prototype recognition, or OCR. A value of 0 means that there is no time limit.

Usage Considerations

A change to this parameter takes effect when a new region is analyzed.

This is an array of parameters—one for each virtual camera. See the *AdeptVision User's Guide* for details on setting parameters.

Details

This parameter sets the (approximate) maximum number of seconds the vision system will spend either analyzing a region and recognizing an object, or performing character recognition (see the VOCR instruction). This limit keeps the vision system from taking too much time analyzing complex, unrecognizable images.

For prototype recognition, note that the maximum time is per object not per region. Thus, for example, if V.MAX.TIME is 2 and three parts touch in a region, the vision system could spend up to 6 seconds analyzing that region.

V.MAX.TIME is set to 5 when the V^+ and AdeptVision systems are loaded into memory from disk. The range is 0 to 999. A value of 0 means that there is no time limit.

Example

Allow 2.5 seconds for the vision system to recognize each object:

```
PARAMETER V.MAX.TIME = 2.5
```

Syntax

```
... V.MAX.VER.DIST [camera]
```

Function

Set the pixel tolerance for determining boundary coincidence during the verification of prototype-to-image matches.

Usage Considerations

A change to this parameter takes effect when the next VPICTURE command or instruction, or VWINDOW instruction, is executed.

This is an array of parameters—one for each virtual camera. (See the *AdeptVision User's Guide* for a general description of parameters.)

Details

During the processing of object recognition, prototype-to-image matches are proposed and verified. To verify a match (that is, verify that an instance of the prototype is visible in the image), the vision system searches for image boundaries that are "nearly coincident" with the proposed positions of the prototype boundaries. The system parameter V.MAX.VER.DIST defines, in terms of pixels, the tolerance of the "nearly coincident" test. The portions of the prototype boundaries that are located within V.MAX.VER.DIST pixels of image boundaries are considered verified.

The parameter V.LAST.VER.DIST is used in the same way as V.MAX.VER.DIST but only during the final verification of each match. Depending on the complexity of the prototype being recognized, one or more verifications are performed per match proposal. When a prototype-to-image match is verified, all the edges of the prototype are compared to the nearby image edges (that is, edges within V.MAX.VER.DIST pixels), and statistics are gathered describing the deviations in position and rotation. Following each verification, the vision system computes a prototype-to-image transformation that minimizes the deviations in edge positions and rotations.

These steps of verification and repositioning may be performed a number of times in order to refine the fit of the prototype. If the parameter V.LAST.VER.DIST has a nonzero value, an extra, final verification is performed using V.LAST.VER.DIST as the maximum verify distance. V.LAST.VER.DIST, if smaller than V.MAX.VER.DIST, becomes a tolerance test. If V.LAST.VER.DIST is larger than V.MAX.VER.DIST, V.LAST.VER.DIST reaches out to features of the object that may be out of tolerance. (The program instructions VSUBPROTO and VGAPS provide final verification information on selected individual edges or subprototypes of the prototype.)

This parameter must be assigned a real value in the range 1 to 16, inclusive. V.MAX.VER.DIST should always be greater than V.MAX.PIXEL.VAR. The parameter V.MAX.VER.DIST is set to 3 when the V+ and AdeptVision systems are loaded into memory from disk.

Example

```
PARAMETER V.MAX.VER.DIST = 3.5
```

Related Keywords

VGAPS (program instruction)
V.LAST.VER.DIST (system parameter)
V.MAX.PIXEL.VAR (system parameter)
V.RECOGNITION (system parameter)
VSUBPROTO (program instruction)

Syntax

```
... V.MIN.AREA [camera]
```

Function

Set the minimum area below which the vision system ignores regions.

Usage Considerations

Normally, changes to this parameter take effect immediately, filtering out any small regions that close off. However, when clear-grips have been defined, changes to this parameter do not go into effect until a new VPICTURE is executed.

The effects of changes to this parameter are visible in the Vision display window only in a special display mode. Regions smaller than V.MIN.AREA will still appear in display modes #1 and #2.

This is an array of parameters—one for each virtual camera. See the *AdeptVision User's Guide* for details on setting parameters.

Details

Regions that contain fewer than V.MIN.AREA number of pixels are ignored by the vision system during object recognition, clear-grip testing, and standard ruler measurements. This parameter should be set large enough so that the vision system does not waste time analyzing noise spots and small enough so that no region of interest is discarded.

This parameter must be assigned an integer value in the range 1 to 1,048,576 (1024∗1024), inclusive. However, the value must be less than or equal to the value of V.MAX.AREA,[1] and it must be greater than or equal to the value of the parameter V.MIN.HOLE.AREA. The parameter V.MIN.AREA is set to 16 when the V+ and AdeptVision systems are loaded into memory from disk.

Example

Ignore regions that are smaller than 25 pixels in area:

```
PARAMETER V.MIN.AREA = 25
```

Related Keywords

V.MAX.AREA (system parameter)
V.MIN.HOLE.AREA (system parameter)

[1] The effective maximum of V.MAX.AREA is the area of the virtual frame buffer. Virtual frame buffer sizes are specified with the DEVICE instruction.

Syntax

```
... V.MIN.HOLE.AREA [camera]
```

Function

Set the minimum area below which the vision system ignores holes.

Usage Considerations

A change to this parameter takes effect when a new region is analyzed.

The effects of changes to this parameter are visible in the Vision display window only in a special display mode. Holes with areas smaller than V.MIN.HOLE.AREA will still appear in display modes #1 and #2.

This is an array of parameters—one for each virtual camera. See the *AdeptVision User's Guide* for details on setting parameters.

Details

Holes that contain fewer than V.MIN.HOLE.AREA number of pixels are ignored by the vision system during object recognition, clear-grip testing, and standard ruler measurements. This parameter should be set large enough so that the vision system does not waste time analyzing noise spots and small enough so that no region of interest is discarded.

This parameter must be assigned an integer value in the range 1 to the value of the parameter V.MIN.AREA, inclusive. The system parameter V.MIN.HOLE.AREA is set to 8 when the V$^+$ and AdeptVision systems are loaded into memory from disk.

Example

Ignore holes that are smaller than 10 pixels in area:

```
PARAMETER V.MIN.HOLE.AREA = 10
```

Related Keywords

V.MAX.AREA (system parameter)
V.MIN.AREA (system parameter)

Syntax

```
...V.MIN.LEN [camera]
```

Function

Set the minimum length of features to be used for feature pairs.

Usage Considerations

This parameter is used by VTRAIN.FINDER and VPLAN.FINDER. See those program instructions for details.

This is an array of parameters—one for each virtual camera. See the *AdeptVision User's Guide* for details on setting parameters.

Details

Features that contain fewer than V.MIN.LEN number of pixels are ignored by the vision system during feature-pairing operations. This parameter should be set large enough so that the vision system does not waste time analyzing noise spots but small enough so that all features of a minimum desired length are included.

This parameter must be assigned an integer value (there is no minimum or maximum value). The default value is 40 pixels.

Example

Ignore features that are smaller than 25 pixels in length.

```
PARAMETER V.MIN.LEN = 25
```

Syntax

```
... V.MIN.MAX.RADII [camera]
```

Function

Enable the feature that, for each region in the image, finds the two points on the perimeter that are closest to and farthest from the region centroid.

Usage Considerations

The system switches V.BOUNDARIES and V.CENTROID must be enabled in order for V.MIN.MAX.RADII to have its effect.

A change to this switch takes effect when the next VPICTURE command or instruction, or VWINDOW instruction, is executed.

This is an array of switches—one for each virtual camera. See the *AdeptVision User's Guide* for details on setting switches.

Details

When the switches V.MIN.MAX.RADII and V.CENTROID are both enabled, the perimeters of all regions are scanned to find the points closest to, and farthest from, the region centroids. This information is available with the VFEATURE function after a VLOCATE instruction has succeeded. The points are represented by their distances and directions (angles) from the region centroid.

This information is also available for holes. To get information about holes, you should enable V.HOLES, disable V.DISJOINT, and do VLOCATEs in get-hole mode.

Example

For each region in the image, the following program segment draws lines from the centroid to the points on the perimeter that are closest to and farthest from the centroid.

```
;Required switches

ENABLE V.BOUNDARIES, V.CENTROID, V.MIN.MAX.RADII
VDISPLAY 3                              ;Special display
VPICTURE ,0                             ;Take a picture with virtual
                                        ; camera #1 and no recognition
ATTACH (vlun, 4) "GRAPHICS"             ;Attach to the vision window
FOPEN (vlun) "Vision /MAXSIZE 640 480" ; & select graphics scaling
GTRANS (vlun, 1)                        ; in real-world millimeters
vf.cx = 42                              ;Indexes of VFEATURE function
vf.cy = 43                              ; for centroid
vf.minr.ang = 44                        ; and min/max radii values
```

```
vf.maxr.ang = 45
vf.minr.dist = 46
vf.maxr.dist = 47
VWAIT                                   ;Wait for image processing to
                                        ; complete for graphics instr.
VLOCATE () $nam                         ;Locate anything in the image

WHILE VFEATURE(1) DO                    ;If a region was found...
   cx = VFEATURE(vf.cx)                 ;Get centroid: Cx,Cy
   cy = VFEATURE(vf.cy)
   rmindist = VFEATURE(vf.minr.dist)    ;Get distance and angle to
   rmaxdist = VFEATURE(vf.maxr.dist)    ; closest and farthest pts
   rminang = VFEATURE(vf.minr.ang)
   rmaxang = VFEATURE(vf.maxr.ang)

   xx = cx+COS(rminang)*rmindist        ;Draw line to closest point
   yy = cy+SIN(rminang)*rmindist
   GLINE (vlun) cx, cy, xx, yy
   xx = cx+COS(rmaxang)*rmaxdist        ;Draw line to farthest point
   yy = cy+SIN(rmaxang)*rmaxdist
   GLINE (vlun) cx, cy, xx, yy
   VLOCATE () $nam                      ;Locate next region in the image
END
```

Related Keywords

V.2ND.MOMENTS (system switch)
V.BOUNDARIES (system switch)
V.CENTROID (system switch)
V.DISJOINT (system switch)
VFEATURE (real-valued function)
V.HOLES (system switch)
VLOCATE (program instruction)
V.PERIMETER (system switch)

Syntax

```
VMORPH (cam, type, dmode, thresh) dest_ibr, count = src_ibr
```

Function

Perform a morphological transform on a binary image frame.

Parameters

`cam`	Not currently used.
`type`	Integer value indicating the type of morphological operation to perform. Type 1 is erosion, 2 is dilation, and types 8 through 16 are user-definable. Types 3 though 7 are undefined, reserved for future built-in operations.
`dmode`	Optional real-valued expression specifying the display mode to use when displaying the border of the area-of-interest: –1 = no draw, 0 = erase, 1 = draw solid, 2 = complement, 3 = draw dashed, 4 = complement dashed. The default is 1 (draw solid).
`thresh`	Not currently used.
`dest_ibr`	Integer value specifying the image buffer region to receive the image data that has been modified with a morphological operation. Image buffer regions specify both a size and a frame store (see the description of VDEF.AOI).
`count`	Optional variable name that receives the sum of binary pixels in the destination image buffer region. This can be used as a stopping criterion for edge thinning operations (to indicate stasis state). If "`count`" is specified, forward processing is suspended until the operation completes.
`src_ibr`	Integer value specifying the image buffer region to apply the morphological operation. The region's AOI must have been defined with a VDEF.AOI instruction.

Details

Binary morphological operations are nonlinear transformations of binary (or edge) images. The VDEF.MORPH instruction defines the operation to perform. The VMORPH program instruction performs the operation.

Morphological operations may be used to eliminate small holes and gaps from the image (by dilating and then eroding or vice versa), to thin edges, to isolate certain features such as straight lines, etc. Multiple operations are often performed in sequence.

Morphological operation types 1 and 2 are predefined to be erosion and dilation, respectively. Types 3 through 7 are undefined, reserved for future built-in operations. Types 8 through 16 are user-definable. (Type 9 is actually predefined to be "Life", a simulation of cellular organisms popularized years ago in "Scientific American", but it can be redefined by the user).

Binary morphological operations are applied to every 3x3 pixel neighborhood in the binary image. Based on the binary pixel values in a neighborhood and the operation to be performed, the center binary pixel value may be changed. See the VDEF.MORPH instruction in this manual for details on how the operation is performed.

The effect of a VMORPH operation is visible in VDISPLAY mode #2. It does not affect the associated grayscale image.

The erosion operation clears binary pixels that have at least one neighboring pixel with the value 0. The dilation operation does the opposite, setting all pixels that have at least one neighboring pixel with the value 1.

The smaller the area of the image to be processed, the faster VMORPH executes.

Examples

```
VDEF.AOI 2000 = 1, 100, 100, 50, 50
VMORPH (, 2) = 2011;Dilate the binary image twice
VMORPH (, 2) = 2011
VMORPH (, 1) = 2011;Then erode the image twice
VMORPH (, 1) = 2011
```

Related Keywords

VDEF.AOI (program instruction)
VDEF.MORPH (program instruction)

Syntax

```
VOCR (cam, op, dmode) data[i], locs[j] = font_num, $expected, ibr

VOCR (cam, op, dmode) data[i], locs[j] = font_num, $expected,
                                         shape, cx, cy, dx, dy, ang
```

Function

Perform Optical Character Recognition (OCR) or text verification in a rectangular image window.

Usage Considerations

A font must have been created previously with the VDEF.FONT and VTRAIN.MODEL instructions.

Parameters

`cam` Optional real-valued expression indicating the virtual camera number to use for selecting various system parameters (such as V.MIN.AREA and V.MAX.AREA). The default camera is 1.

`op` Optional real-valued expression specifying the desired operation to perform (the default is 0):

0 = Quick verification of expected text
1 = Robust verification of expected text, short output
2 = Robust verification of expected text, full output
3 = Recognition (no expected text)

`dmode` Optional real-valued expression specifying the display mode to use when displaying the border of the area-of-interest: –1 = no draw, 0 = erase, 1 = draw solid, 2 = complement, 3 = draw dashed, 4 = complement dashed. The default is 1 (draw solid).

NOTE: The parentheses in the instruction syntax can be omitted if all three of the above parameters are omitted.

`data[]` Real array that is filled with data describing the results of the OCR operation (see below).

`locs[]` Optional real array that is filled with data describing the position of the recognized characters (see below).

`i, j` Optional array index that identifies the first element to be defined in "**`data[]`**" and "`locs[]`", respectively. Zero is assumed for any index that is omitted. If a multiple-dimension array is specified, only the right-most index is incremented as the values are assigned.

font_num	Real-valued expression that specifies the font to be recognized or verified. (The value must be in the range 1 to 99.)
$expected	Optional string containing the text expected to be found within the OCR window. This string is ignored if "op" is 3. This string is required for all other values of "op".
ibr	Integer value specifying the image buffer region within which to search for characters for recognition or verification. Image buffer regions specify both a size and a frame store (see the description of VDEF.AOI).
shape	Optional real-valued expression indicating the shape of the window. Currently, the only choice is 1, for a rectangular window.
cx, cy	Real-valued expressions specifying the center coordinates of the window, in millimeters.
dx	Real-valued expression specifying the width of the window, in millimeters.
dy	Real-valued expression specifying the height of the window, in millimeters.
ang	Optional real-valued expression specifying the orientation of the window, in degrees. The default is 0 degrees.

Details

This instruction performs the OCR functions of text verification or recognition. When text is verified, the vision system knows what text to expect. With text recognition, the vision system tries to recognize the text with no expectations, except that the characters in the image can be found in the designated font.

Before this instruction is used, the specified OCR font must have been defined using the VDEF.FONT instruction and fully trained using the VTRAIN.MODEL instruction. (See the ***AdeptVision User's Guide*** for an overview of the AdeptVision OCR capability.)

VOCR operates on the binary image in the current frame store. The virtual-camera argument to VOCR selects the group of system switches and parameters to use. The following switches and parameters affect VOCR: V.MIN.HOLE.AREA, V.MIN.AREA, V.MAX.AREA, V.MAX.TIME, and V.SHOW.EDGES. The system switch V.SUBTRACT.HOLES is always enabled during VOCR operations.

The VOCR image buffer region should encompass a single line of text. Within the image buffer region, characters are ordered left to right, assuming no rotation (element 5 of the AOI is 0). If rotated, the definition of "left" changes with the rotation. The VOCR operator is drawn like VWINDOW or VWINDOWI, except that the "left" edge is drawn in a special color for reference. To see the character outlines during recognition, enable V.SHOW.EDGES (for the specified virtual camera) and provide a "dmode" value other than –1 (no draw). Note, however, that this slows down the execution of VOCR somewhat.

Planning for OCR needs to be performed (one time) after a font has been trained or loaded from disk (using VLOAD). If the font being used has not been planned, the VOCR instruction automatically initiates planning. Also, the VTRAIN.MODEL instruction may be used to force the vision system to plan for OCR.

VOCR returns a variety of information to satisfy different inspection requirements. The information returned by the four modes of operation (selected with the "op" parameter) is explained below. Each operation mode returns match scores in the range 0 to 100. A score of 0 means no correlation. A score of 100 means a perfect match. Scores of 100 are uncommon. To earn this, a character would have to be the exact average of all its trained examples.

Operation 0: Quick Verification

In this mode of operation, the bounded areas in the image buffer region are compared against the given expected text ($expected) on a 1-to-1 basis. This is the fastest mode of operation. Only three values are returned in the array "**data[]**":

data[i+0] = Number of character regions found and analyzed
data[i+1] = Average score of "$expected" characters verified
data[i+2] = Minimum score of "$expected" characters verified

Operation 1: Robust Verification, Short Output

This is a more robust form of verification than Quick Verification (op = 0), but it is slower. Even though VOCR knows what text to expect, it compares each bounded area in the image buffer region with all the characters in the font. The top two character matches for each bounded area are then compared to the expected character. (The top two matches or "picks" are the two with the highest match scores.) VOCR then returns data indicating whether all of the expected characters were first picks or, at least, first or second picks. If all of the expected characters were first picks and the font is complete, the text is guaranteed to be readable. Even if the expected characters were all first or second picks, the text was probably readable. Some characters in the font, such as "l" and "1", may be very similar and, thus, open to misreading by VOCR, whereas a human would use

context to choose between the two. For example, in the text “War of 16l2”, people readily interpret the “l” (lowercase “L”) between the “6” and the “2” as the numeral “1”. In this mode of operation, the following data is returned in addition to that returned for Quick Verification (op = 0) above:

data[i+3] = TRUE if expected characters were all 1st picks
data[i+4] = TRUE if they were all 1st or 2nd picks

Operation 2: Robust Verification, Full Output

This operation is identical to the one above (op = 1), except more information is returned. In addition to the array elements described above, the top two picks and their scores are returned for each character in the expected text.

data[i+5] = 1st character pick
data[i+6] = Score of 1st character pick
data[i+7] = 2nd character pick
data[i+8] = Score of 2nd character pick
... (These four items repeat for each character region found and
analyzed—a total of “data[i+0]” times)

This additional information may be used in the V+ program to permit certain alternates to the expected.

Operation 3: Recognition

Recognition means there is no expected text. Each character region in the image is compared to each character in the font. For each character region found and analyzed, the two top-scoring characters in the font are returned along with their scores. The data returned is almost identical to that described above.

data[i+0] = Number of character regions found and analyzed
data[i+1] = Average score of all top picks
data[i+2] = Minimum score of all top picks
data[i+3] = 0
data[i+4] = 0
data[i+5] = 1st character pick
data[i+6] = Score of 1st character pick
data[i+7] = 2nd character pick
data[i+8] = Score of 2nd character pick
... (The last four items repeat for each character region found and
analyzed—a total of “data[i+0]” times)

Regardless of the operation performed, position information is returned when the array “locs[]” is specified in the VOCR instruction. Returning this information takes additional time, so if it is not needed, the array “locs[]” should not be specified. When the array “locs[]” is specified, it receives a definition of the

minimum box that surrounds all of the bounded areas. If the operation is "Robust Verification, Full Output" (op = 2) or "Recognition" (op = 3), the center of the bounding box for each character is also returned. All of the position information is in millimeters.

Information returned by all operations:

loc[j+0] = Minimum X of text bounding box
loc[j+1] = Minimum Y of text bounding box
loc[j+2] = Maximum X of text bounding box
loc[j+3] = Maximum Y of text bounding box

Additional information returned when "op" is 2 or 3:

loc[j+4] = X component of center of character bounding box
loc[j+5] = Y component of center of character bounding box
... (These two items repeat for each character region found and analyzed—a total of "data[i+0]" times)

Any bounded area taller than 63 pixels is ignored. If the characters to be recognized are larger than this, normal prototype recognition may be used. VOCR also considers the values of V.MIN.AREA, V.MIN.HOLE.AREA, and V.MAX.AREA. These parameters may be used to filter unwanted regions within the VOCR tool search area.

Up to 80 character regions are analyzed in an OCR window. Any additional bounded areas are ignored. Similarly, when verifying text, extra characters in the window are ignored. For example, if there are 10 characters in the given "$expected" string and 12 bounded areas are found in the window, the right most 2 areas are ignored. That is, VOCR reports that only 10 characters were found.

If fewer character regions are found than expected, the expected characters with no corresponding bounded areas are given a score of 0 when computing the average and minimum scores. Also, "data[i+3]" ("expected characters were all 1st picks") and "data[i+4]" ("they were all 1st or 2nd picks") are both assigned the value FALSE.

Merged characters are automatically split when there is expected text. This includes the VOCR operations "Quick verification" (op = 0) and "Robust verification" (op = 1 or 2), but not "Recognition" (op = 3). Splitting works with rotated VOCR windows.

The splitting criteria are simple. First, there must be fewer regions in the VOCR window than expected. Then, while processing the bounded areas in the VOCR window from left to right, the following test is performed: If the width of the current area is closer to the total width of the current and next expected characters than to the width of the character currently expected, then the bounded area is split.

A bounded area is split based on the average width of the character currently expected. (The average width is based on the trained instances of the character.) This simple strategy for splitting works well when only two characters are merged into a single bounded area, but inaccuracies accumulate if more characters are merged together. As a general rule, VOCR splits two characters very well, splits three merged characters moderately well, and splits four or more characters poorly.

Example

The following V+ program uses the VOCR instruction to perform character recognition or verification (depending on the value of the variable "op"). The results are displayed in the Monitor window, a box is drawn around the text in the Vision display window, and (if "op" is 2 or 3) a dot is drawn at the center of each character found.

> **NOTE:** This example program assumes that the OCR font has already been defined and trained.

```
.PROGRAM test.ocr()

; ABSTRACTExample program showing basic OCR capabilities

   AUTO ang, cam, cc, cnt, cx, cy, dx, dy, dm, font, op
   AUTO $ans, $exp
   LOCAL xx[], data[], locs[]

; Change these values to suit your needs:

   font = 1                    ;Font number
   op = 2                      ;VOCR operation: 0 to 3
   $exp = "ABCDEFG"            ;Expected text (required if op <> 3)
   cx = 256                    ;Center of the box
   cy = 242
   dx = 350                    ;Width and height of the box
   dy = 50
   ang = 0                     ;Orientation of the box

   cam = 1                     ;Camera number
   dm = -1                     ;Display mode

   VDISPLAY (cam) 2, 1
   VPICTURE (cam) 2
   WHILE TRUE DO
      VWINDOWI xx[] = , cx, cy, dx, dy, ang ;Draw window
      VDISPLAY (cam) 0, 1              ;Show live video
      TYPE "Position the text in the window shown. ", /S
      PROMPT "Press Enter when ready. ", $ans
      VPICTURE (cam) 2
      VDEF.AOI 2000 = 1, cx, cy, dx, dy, ang
      VOCR (cam, op, dm) data[], locs[] font, $exp, 2001
```

```
            TYPE "# chars found:", data[0], /S
            TYPE ", Average score:", data[1], ", Min score:", data[2]
            TYPE "# chars found:", data[0], /S
            TYPE ", Average score:", data[1], ", Min score:", data[2]

            IF (op = 1) OR (op = 2) THEN
                TYPE "Expected 1st picks? ", data[3], /S
                TYPE ". 1st or 2nd? ", data[4]
            END
            IF op > 2 THEN              ;If individual picks returned...
                cc = 5
                FOR cnt 1 TO data[0]
                    TYPE "Picks: ", $INTB(data[cc]), /S
                    TYPE " (", /I0, data[cc+1], ")", /S
                    TYPE ", ", $INTB(data[cc+2]), /S
                    TYPE " (", /I0, data[cc+3], ")"
                    cc = cc+4
                END
            END

        ; Draw text bounding box

            GTRANS (20, 1)
            GRECTANGLE (20) locs[0], locs[1], locs[2], locs[3]
            IF op > 2 THEN
                FOR cnt = 0 TO data[0]-1 ;"Dot" centers of regions
                    GARC (20) locs[2*cnt+4], locs[2*cnt+5], 3
                END
            END

            PROMPT "Press Enter to repeat, or Ctrl+Z to quit. ", $ans
        END
    .END
```

Related Keywords

VDEF.AOI (program instruction)
VDEF.FONT (program instruction)
V.MAX.AREA (system switch)
V.MAX.TIME (system switch)
V.MIN.AREA (system switch)
V.MIN.HOLE.AREA (system switch)
V.SHOW.EDGES (system switch)
VSHOW.MODEL (program instruction)
VTRAIN.MODEL (program instruction)

Syntax

```
... V.OFFSET [camera]
```

Function

Set the offset for the incoming video signal (that is, program the zero reference for the A / D converter).

Usage Considerations

Changing this parameter immediately affects the video output of the camera interface board.

This is an array of switches—one for each virtual camera. See the *AdeptVision User's Guide* for details on setting switches.

Details

The V.OFFSET parameter works with the V.GAIN parameter to select the incoming analog video offset and gain, respectively. V.GAIN multiplies (scales) the video signal, whereas V.OFFSET shifts (translates) the video signal.

Before adjusting the values of V.OFFSET and V.GAIN, you should take a picture, compute the histogram, and study the video data displayed in the Vision display window. To take the picture and compute the histogram, enter the V+ monitor commands "VPICTURE (cam) 2" and "VHISTOGRAM ( )", where "cam" is the number of the virtual camera being used. Or, use the mouse to make the menu selections to perform these same operations.

The goal is to have the video data fill most of the intensity range (0 to 127) without spilling over either end. If the video data spills over the left end, the histogram curve shows a spike over the "0" intensity label. Similarly, if the video data spills over the right end, the curve shows a spike at or near the "127" intensity label.

You should increase V.GAIN to expand the intensity range of the video data, or decrease V.GAIN to reduce the intensity range. Similarly, you should increase V.OFFSET to shift the video data toward the right, and decrease V.OFFSET to shift it toward the left. For good results, you may have to repeat the procedure of taking a picture, computing the new histogram, and adjusting V.GAIN and V.OFFSET a few times.

V.OFFSET must be assigned an integer value in the range 0 to 255, inclusive. The parameter is set to 255 when the V+ and AdeptVision systems are loaded into memory from disk.

Example

Shift the incoming video by 96:

```
PARAMETER V.OFFSET = 96
```

Related Keywords

VHISTOGRAM (monitor command and program instruction)
V.GAIN (system parameter)

Syntax

```
... V.OVERLAPPING [camera]
```

Function

Determine whether or not objects may be overlapping in the image.

Usage Considerations

A change to this switch takes effect when the next VPICTURE command or instruction, or VWINDOW instruction, is executed.

If this switch is enabled, the V.TOUCHING system switch is automatically assumed to be enabled.

This is an array of switches—one for each virtual camera. See the *AdeptVision User's Guide* for details on setting switches.

Details

Normally, objects to be recognized in the scene are not overlapping, so V.OVERLAPPING should be disabled. Object recognition is more efficient in this case.

The setting of V.OVERLAPPING affects the recognition strategy used by the vision system and the way recognition proposals are verified in the image. If V.OVERLAPPING is enabled, any edge in the image may be used to verify multiple objects, not just one object. For example, if the corresponding holes of two overlapping objects line up, the one hole seen could be used to verify both objects.

Since edges may be used to verify the presence of more than one object, in some situations the vision system may incorrectly recognize two or more objects where there is only one. A V^+ program could be written that compares the locations of the objects in order to discard the invalid duplicates.

Example

Do not allow objects to overlap in the image:

```
DISABLE V.OVERLAPPING
```

Related Keyword

V.BOUNDARIES (system switch)
V.TOUCHING (system switch)

Syntax

```
... V.PERIMETER [camera]
```

Function

Enable computation of the lengths of region perimeters.

Usage Considerations

The V.BOUNDARIES system switch must be enabled to compute perimeters.

A change to this switch takes effect when the next VPICTURE command or instruction, or VWINDOW instruction, is executed.

This is an array of switches—one for each virtual camera. See the *AdeptVision User's Guide* for details on setting switches.

Details

When this switch is enabled, the outer boundary of each region in the image is scanned and its length is computed. The computation is fast. The information is available from the VFEATURE function after a VLOCATE instruction has succeeded.

This information is also available for holes. To get information for holes, you should enable V.HOLES, disable V.DISJOINT, and do VLOCATEs in get-hole mode.

Example

The following program segment computes a "roundness" factor for each region in the scene based on the region perimeter and area. The roundness factor will be close to 1.0 for circles and less than 1.0 for other shapes. This may be used in place of prototype recognition to recognize circles.

```
ENABLE V.BOUNDARIES, V.PERIMETER ;Necessary switches
DISABLE V.SUBTRACT.HOLE          ;If this is enabled, VFEATURE(40)
                                 ; should be added to the area
vf.area = 10                     ;VFEATURE index for area (pixels)
vf.perimeter = 41                ;VFEATURE index for perimeter (mm)

VGETCAL vb[]                     ;Need items in calibration array
                                 ;Define indexes for:
x.scale = 15                     ; Pixel scale(mm/pixels)
y.scale = 16
scale = SQR((vb[x.scale]+vb[y.scale])/2)

VPICTURE ,0                      ;Take picture with virt. cam#1, no recog
VLOCATE () $nam                  ;Locate anything in the image
```

```
WHILE VFEATURE(1) DO ;If a region was found...
   area = VFEATURE(vf.area)*scale   ;Raw pixel area > mm square
   perim = VFEATURE(vf.perimeter)
   roundness = 4*PI*area/(perim*perim)
   TYPE "Region roundness ", roundness
   VLOCATE () $nam                  ;Locate next region in the image
END
```

Related Keywords

VFEATURE (real-valued function)
V.2ND.MOMENTS (system switch)
V.CENTROID (system switch)
V.HOLES (system switch)
V.MIN.MAX.RADII (system switch)

Syntax

```
VPICTURE (camera, wait, acq_ibr, sel_ibr) mode, how_many
```

Function

Acquire an image into a frame store and / or initiate processing.

Usage Considerations

The VPICTURE program instruction does not suspend program execution while the image is being processed. In other words, program execution continues with the next instruction while the vision system processes the image. However, if the parameter V.IO.WAIT is set to 1, image processing does not begin until the fast digital-input interrupt signal is received. Thus, in that case, program execution can be suspended for an indefinite time while the system waits for the high speed digital input signal.

> **NOTE:** Some vision instructions cause an error if they are executed while vision processing is still active. The VWAIT program instruction will suspend program execution until the vision processor is idle.

Parameters

`camera` Optional real-valued expression that specifies the virtual camera number. The value 1 is assumed if no camera is specified.

`wait` Optional real-valued expression indicating whether or not to wait for the image acquisition to complete. The choices are:

–1	to wait for completion of the image acquire and the first stage of processing (run-length encoding)
0	to wait only for the image acquire to start
1	for no waiting at all

Values of 0 and 1 apply only to quick frame grabs (mode #2, see below). The default is –1 (wait for the completion).

`acq_ibr` Optional real-valued expression that specifies the grayscale frame store into which the image is acquired. Currently, only the virtual frame buffer referenced in the image buffer region is used. The full frame is used regardless of any AOI defined for the image buffer region. The default is 1011.

`sel_ibr` Optional real-valued expression that specifies the grayscale frame store to be selected for subsequent processing. Currently, only the virtual frame buffer referenced in the image buffer region is used.

The full frame is used regardless of any AOI defined for the image buffer region. The default is the value of "acq_ibr", indicating the frame buffer into which the image is acquired. This parameter applies only to quick frame grabs (mode #2). In other modes, the selected frame is made equal to "acq_ibr".

NOTE: The parentheses in the instruction syntax can be omitted if all four of the above parameters are omitted.

`mode` Real-valued expression that specifies the processing mode. The mode is optional and defaults to –1.

–1	Initiates processing of a new camera image
0	Initiates reprocessing of previous camera image
1	Grabs an image and holds it for future processing
2	Quick frame grab

The above list should actually be listed in reverse order because a VPICTURE in mode # –1 is effectively the same as a VPICTURE in mode #2 followed by another in mode #0. Mode #2 acquires an image (as does mode #1), and mode #0 processes the image. Mode # –1 does both.

`how_many` Optional real-valued expression that specifies the maximum number of objects the vision system should try to recognize. This limit is applicable only in VPICTURE modes # –1 and #0. (See below for details.)

–1	Locate as many as possible (the default)
0	Locate none (V.RECOGNITION effectively disabled)
1	Locate only one object
2	Locate at most two objects

etc.

Details

VPICTURE modes #–1 and #0 initiate processing. When this happens, the vision system collects the camera data (grabs a frame), analyzes it to identify objects, and displays the results in the Vision display window in the manner dictated by the current VDISPLAY mode. The computed part information is then stored in the vision system queue and is accessible with the VLOCATE program instruction and the VFEATURE real-valued function. Also, region boundary descriptions are available with the VEDGE.INFO instruction.

The vision system queues (one for each virtual camera) have a specified amount of memory allocated for recognized and unrecognized regions. This allocation can be set with the DEVICE program instruction. Holes are included in the queues if the V.HOLES system switch is enabled and the V.DISJOINT system switch is disabled.

An image is processed when a VPICTURE operation is performed in mode #–1 or #0 and all the object data for a particular virtual camera is retained in that virtual camera's queue until another VPICTURE or VWINDOW operation for the same virtual camera is executed. This means that the user may accumulate located object information gathered from several different virtual cameras before accessing it with VLOCATE instructions.

Keep in mind that a significant time delay (perhaps as long as a minute or more) can occur before image processing commences. This delay occurs the first time a VPICTURE or VWINDOW operation is executed for a particular virtual camera after prototype training (VTRAIN) has been performed, or certain recognition parameters and switches have been altered for that virtual camera. During this interval, the vision system analyzes the trained information to generate the data that it requires to perform the actual image processing. If a VPICTURE monitor command causes this planning, the message "Planning, please wait" is displayed in the Monitor display window. The VPICTURE and VWINDOW program instructions do not give any such notice.

When VPICTURE is executed as a monitor command, several informational messages may be displayed. These indicate which prototypes are known (with their virtual camera associations), and which require more training before they can be reliably recognized.

Execution of a VPICTURE instruction generally completes (and thus the program continues) after the frame grab is done, but before all image processing is completed. Consequently, if the camera is mounted on a robot, the controlling V+ program may have robot motion instructions following a VPICTURE instruction. Then the vision system will process the acquired image while the robot is moving.

There are four exceptions to the statement that a VPICTURE instruction completes after the frame grab is done:

1. If the V.STROBE system switch is enabled, the VPICTURE completes immediately after the strobe signal, before the acquisition is done (see the description of V.STROBE for more details).
2. If the VPICTURE mode is #2 (quick frame grab) and "`wait`" = 0, VPICTURE completes after the acquisition has been started. Not waiting until the acquisition is complete is appropriate only when the camera and scene are both still.

3. If the VPICTURE mode is #2 (quick frame grab) and "`wait`" = 1, VPICTURE completes immediately, before acquisition begins. This mode is useful when the VPICTURE is waiting for a fast digital-input interrupt signal before starting to acquire an image (see below).
4. If "`wait`" = –1, VPICTURE will wait until the first stage of processing (run-length encoding) is complete.

The above exceptions do not apply if planning for prototype recognition is performed. In that case, VPICTURE always waits until the frame grab is done before completing.

If the V.IO.WAIT system parameter is set to 1, a VPICTURE waits for a fast digital-input interrupt signal before **starting** to acquire an image. Consequently, except when the VPICTURE "`wait`" parameter is 1, the program containing the VPICTURE instruction will wait an unlimited amount of time until the input signal is received. (See the description of the V.IO.WAIT parameter for details.)

The "future frame grab" provides the ability to pregrab an image and hold it for processing later (mode #1). This is useful when the vision system is still processing a previous image and the next one is in view of the camera. (The vision system can process only one image at a time.) Instead of making a robot or manufacturing process wait for the current image processing to complete, mode #1 may be used to pregrab the next image and thus allow the robot or process to proceed.

With a future frame grab, no operation may be performed on the frame-grabbed image except for a VPICTURE in mode #0. However, a future frame grab can be converted into a quick frame grab (mode #2, below) by simply VSELECTing the frame. Also, if the frame being acquired into is different from the frame store currently selected, a future frame grab is automatically treated as a quick frame grab. This distinction is important because almost all of the vision instructions operate on images acquired via quick frame grabs.

VPICTURE in mode #0 processes the pregrabbed image. If no pregrabbed image is being held, the previous image is reprocessed. This is useful for experimenting with different switch and parameter settings. Note, however, that the settings of the gain, offset, and threshold parameters do not have any effect upon reprocessing of a picture. (The VEDGE and VTHRESHOLD instructions may be used to compute new edge or binary images from an existing grayscale image.) If there is no pregrabbed image, mode #0 is identical to:

```
VWINDOW(cam, 1, dmode, how_many) 1000
```

Mode #2 is provided as a "quick frame grab" mode for measurement and inspection instructions such as VRULERI and VFIND.LINE, for image processing instructions such as VHISTOGRAM and VSUBTRACT, and for the VWINDOW instruction. VPICTURE in mode #2 only grabs an image, performing no automatic region or boundary analysis.

Mode #2 takes from 16 - 50 milliseconds to complete (if the V.BINARY system switch is enabled). By comparison, VPICTURE operations in modes #–1 and #0 typically take 60 - 80 milliseconds to complete, depending on the complexity of the binary image (assuming that the entire image is being processed as defined by the image buffer region). Thus, VPICTURE operations in mode #2 provide time savings for vision applications that use windows or inspection operators, but no object recognition or region analysis (for example, centroids) outside of windows. For maximum speed of a VPICTURE in mode #2, enable the V.BINARY system switch.

There are two grayscale physical frame stores that can acquire images. While one frame store is acquiring a quick frame grab, the other can be analyzed via inspection operators in a V$^+$ program. This means that images can be continuously acquired (30 frames per second) and analyzed in a "ping-pong" manner. The "`acq_ibr`" parameter to VPICTURE indicates which frame store is to receive the new image. The "`sel_ibr`" parameter indicates which frame store to select for subsequent instructions such as VWINDOWI, VRULER, VFIND.LINE, etc.

The VSELECT instruction performs the same function as the "`sel_ibr`" parameter to VPICTURE. Since VSELECT is a separate instruction, however, a program can acquire two camera images into the two frame stores and then use VSELECT to quickly switch back and forth between them while performing analyses.

The values of the parameters V.FIRST.COL, V.FIRST.LINE, V.LAST.COL, and V.LAST.LINE are used to clip the window to the image. That is, the vision system ignores any portion of the window that is outside these boundaries.

If V.RECOGNITION is enabled (prototype recognition is being performed), the "`how_many`" parameter may be very effective in reducing processing time in VPICTURE modes #–1 and #0. (The "`how_many`" parameter is not applicable in other modes.) If the image contains extraneous edges or is noisy, the system normally locates all instances of prototypes quickly but then spends a relatively long time analyzing the extraneous edges. The extra time analyzing the extraneous edges can be avoided if the programmer knows that there is a certain number of objects in the image and specifies that number as the "`how_many`" parameter.

The V.TOUCHING and V.DISJOINT switches affect the way the "how_many" parameter is interpreted. V.TOUCHING affects the way "how_many" applies to each region in the image, whereas V.DISJOINT affects the way "how_many" applies to the image as a whole. If the V.TOUCHING switch is enabled, up to "how_many" objects are recognized per region.[1] If V.TOUCHING is disabled, no more than one object is recognized per region in the image. If V.DISJOINT is enabled, at most "how_many" objects are recognized in the entire image. If V.DISJOINT is disabled, there is no limit to how many objects are recognized in the entire image. The following table summarizes the combinations.

V.TOUCHING	V.DISJOINT	Number of Objects per Region	Number of Objects per Scene
Off	Off	1	No limit
Off	On	1	how_many
On	Off	how_many	No limit
On	On	how_many	how_many

NOTE: If clear-grip tests have been defined for the prototypes (see VDEFGRIP), objects located that do not have a clear grip are not counted toward the "how_many" limit. Thus, more than "how_many" objects may be found and placed in the vision queue (see VQUEUE).

Examples

```
VPICTURE , 3          ;Find at most 3 objects with virtual camera 1
VPICTURE (4)          ;Find as many objects as possible with camera 4
VPICTURE 1            ;Grab a frame for later processing
VPICTURE (7) 0, -1    ;Find as many objects as possible in the
                      ;  image being held from virtual camera 7
```

Initiate picture processing with a monitor command, producing the following output:

```
VPICTURE

Running. Objects to be recognized:
          0    5    10   15   20   25   30
          |....|....|....|....|....|....|..
CASTING    ***---------------------------- (Only 5 examples trained)
FLANGE     *------------------------------ (Only 1 example, ignored)
CRANK      -*--*-*-----------------------*
BRACKET    ------*--------------*--------- (Only 3 examples trained)
```

[1] The V.TOUCHING switch is automatically considered to be enabled whenever the V.OVERLAPPING switch is enabled.

The following code shows how to perform inspections at frame rates (30 frames per second). For simplicity, this program performs a single window inspection on each grayscale frame.

```
; Define AOIs

insp_ibr1 = 1011                ;aoi 1, virt. frame buffer 11
insp_ibr2 = 2012                ;aoi 2, virt. frame buffer 12
VDEF.AOI insp_ibr1 = 1, 380, 240, 50, 50, 0
VDEF.AOI insp_ibr2 = 1, 150, 150, 100, 10, 0

; Prime the pump by grabbing into frame 2

VPICTURE (cam2, 0, insp_ibr2) 2
WHILE TRUE DO                   ;Loop forever...

    ; Grab into 1, select 2 for a window inspection

    VPICTURE (cam1, 0, insp_ibr1, insp_ibr2) 2
    VWINDOWI (cam2, 5, -1) data[] = insp_ibr2
    IF (data[4] < 250) OR (data[4] > 350) THEN

        ; Report deviation in edge counts here

    END

    ; Grab into 2, select 1 for a window inspection

    VPICTURE (cam2, 0, insp_ibr2, insp_ibr1) 2
    VWINDOWI (cam1, 5, -1) data[] = insp_ibr1
    IF (data[4] < 100) OR (data[4] > 150) THEN

        ; Report deviation in edge counts here

    END
END
```

Related Keywords

VDEF.AOI (program instruction)
VLOCATE (program instruction)
VWINDOW (program instruction)
VSELECT (program instruction)
V.BINARY (system switch)
V.DISJOINT (system switch)
V.OVERLAPPING (system switch)
V.TOUCHING (system switch)

Syntax

VPLAN.FINDER (cam, type, dmode) $fmods[], $bmods[]

Function

Set up the type of "planning" used by the Finder when locating models.

Parameters

cam	Real-valued expression indicating the virtual camera number to use for associating "planning" with and to read parameters from.
type	Real-valued expression indicating the type of planning: 0 = Basic initial planning 1 = Quick initial planning (for multi-instance training)
dmode	Not used.
$fmods[ ]	An array of foreground models. Each list must contain names of valid, trained models, starting with index [0] and terminating with a null string as the last name.
$bmods[ ]	An array of background models. Each list must contain names of valid, trained models, starting with index [0] and terminating with a null string as the last name.

Details

Foreground models are those for which you would like the Finder to return locations. Background models are those that may be present, but which you are *not* interested in locating. Specifying these, when applicable, will help planning tell which features of the foreground models are best for finding proposals that are not ambiguous with respect to other similar-looking objects in the scene.

Description of planning types

Type 0 planning takes the lists of models and does all the preparation work needed for recognition ahead of time, to allow for the fastest possible recognition times. Planning is usually fairly fast but can take many seconds to compute and many Kb to store (from V^+ program memory) if the total number of model pairs is large.

Type 1 planning is faster than Type 1 planning but causes recognition to be slower. (However, you may achieve more reliable results using Type 2.) It does this by several means, most significant of which is that confirmation is not planned for. Therefore, during recognition, *all* proposals go on to verification. This method can be costly since verification is a time-consuming operation.

Input parameters using virtual cameras

The following settings are required:

V.MIN.LEN[vc] = 20 Percentage of features (by length) to ignore.

For Type 0(normal planning), V.MIN.LEN is ignored.

For Type 1(quick planning), V.MIN.LEN[vc] is the number of features to use for planning. For example, a value of 20 means to plan with the 20 longest features.

The following settings are suggested:

V.MAX.VER.DIST[vc] = 5.0

Syntax

VPUTCAL (camera) scalers[i**]**, pmm.to.pix[j,k], pix.to.pmm[l,m], to.cam

Function

Load vision calibration data from 1 to 3 arrays.

Usage Considerations

The VISION switch must be enabled, the vision processor must be idle, and vision model training must not be active for this instruction to be executed.

Parameters

camera	Real-valued expression that specifies the virtual camera.
scalers[]	Real array of calibration values (see below).
pmm.to.pix[,]	Optional, two-dimensional real array that defines the millimeter-to-pixel transformation matrix. If this array is specified, the array "pix.to.pmm[,]" must also be specified.
pix.to.pmm[,]	Optional, two-dimensional real array that defines the pixel-to-millimeter transformation matrix. If this array is specified, the array "pmm.to.pix[,]" must also be specified.
i, j, k, l, m	Optional integer values that identify the first array element to be referenced in the respective array. Zero is assumed for each index that is omitted. If an array is specified that has more dimensions than needed, only the right-most indexes are incremented as the values are referenced.
to.cam	Optional transformation expression that defines the vision transformation to store with the calibration data. (Note: in V+ version 11.0, to.cam must be a variable. In version 11.1 and higher, to.cam will not have to be a variable.)

Details

The calibration programs supplied with the AdeptVision systems fill the calibration arrays and execute a VPUTCAL instruction at the end of a successful calibration. This calibration data is used when performing operations that involve millimeter distances or locations in the image, such as VPICTURE, VWINDOW, VRULER, VLOCATE, and VFEATURE.

Since the calibration arrays are generally filled by an Adept utility program, the programmer should not have to be familiar with all the individual array elements. However, since some elements may be useful to know about, a brief description of the array elements is provided in the table below.

Any undefined elements of the input arrays are automatically assumed to be zero. Subsequent VGETCAL instructions will retrieve arrays that are defined completely.

> **NOTE:** The user is discouraged from setting any of the array values unless explicitly requested to do so by Adept documentation or personnel. The descriptions here are intentionally terse.

Table 2-5. Elements of VGETCAL / VPUTCAL Scaler Calibration Array

Index	Name	Value or Meaning
0	calibrated	Calibration status: 0 => Uninitialized 1 => Initialized for calibration 2 => Okay (calibrated)
1	cam.num	Physical camera number
2	method	Calibration method and link mounting used. The value of this element is determined and used by Adept software. See the documentation for the Adept calibration utilities for specific values. (For camera-only calibrations: method = 1.)
3	link.num	Link number of the robot that the camera is mounted on.
4		Maintained for backward compatibility; should not be used in 11.0 or later systems.
5, 6, 7	not used	Must be 0
8	not used	Must be 0 or 1
9 - 12	not used	
13	flags	Bit field: # Not set Set 1 HPS not used HPS used 2 Full frame Field acquire 3 Normal Synchronous shuttered camera 4 Normal Asynchronous shuttered camera Remaining bits reserved, must be 0.
14	xy.ratio	Camera pixel height / width (converts X to Y). Read only—returned by VGETCAL but not used by VPUTCAL.
15	x.scale	Horizontal scale factor (mm / pixel)
16	y.scale	Vertical scale factor (mm / pixel)
17	cam model	0 = Normal RS-170 camera (Panasonic GP-MF702, Sony XC-77, etc.) 1 = Panasonic GP-CD40 2 = Reserved 3 = MF-702 asynch reset mode (noninterlaced full frame 30Hz) 4 = MF-702 asynch reset mode (interlaced full frame) 5 = MF-552 asynch reset mode (interlaced full frame) 6, 7 = Reserved 8-15 = Initially, same as 0.

Table 2-5. Elements of VGETCAL / VPUTCAL Scaler Calibration Array (Continued)

Index	Name	Value or Meaning
18	pixel compensation table	Index 1-8 of pixel compensation table used for filling in missing pixels; 0 = no pixel compensation
19 - 20	not used	Must be 0

Several of the entries in the scaler array documented here are not actually used by the vision system but are generated by the Adept calibration utility programs and are used by other Adept software. Those entries, identified here only for completeness, are:

Element 2:	Calibration method
Element 3:	Link number
Element 13, bit 1:	HPS-used flag

The value for "xy.ratio" must be in the range 0.5 to 1.5. If the "x.scale / y.scale" supplied to VPUTCAL is not in the acceptable range, the error

```
*Invalid vision X/Y ratio*
```

is reported when the next VPICTURE operation is processed.

The arrays "`pmm.to.pix[,]`" and "`pix.to.pmm[,]`" contain two-dimension homogeneous transformations. These transformations implicitly contain the x and y millimeter-per-pixel scaling contained in the array "**`scaler[]`**". In addition, the transformations may contain terms that correct for perspective distortion. If the two transformations are not specified with VPUTCAL, correction for perspective distortion is automatically disabled. (See **Appendix C** for more information on perspective distortion.)

Example

Calibrate virtual camera 2 using the data in "cal.array[ ]":

```
VPUTCAL (2) cal.array[]
```

Related Keyword

VGETCAL (program instruction)

Syntax

```
VPUTPIC (camera, type, zoom) $pic[r,c], x0, y0
```

Function

Store into a frame store an image saved previously with VGETPIC.

Usage Considerations

The VISION switch must be enabled, the vision processor must be idle, and vision model training must not be active for this instruction to be executed.

Parameter

`camera` Optional real-valued expression that specifies the virtual camera number. If not specified, the camera number in the picture header at "$pic[r,c]" is used. (This parameter is currently ignored.)

`type` Optional real-valued expression indicating the type of data to expect in the "**`$pic[,]`**" array. If not specified, the value in the picture header is used.

1 = Grayscale image and binary (or edge) image
2 = Binary (or edge) image only

`zoom` Optional real-valued expression indicating the zoom factor to use when writing the picture to the frame store. If zoom is N, every pixel is blown up to an NxN square. The zoom factor must be in the range 1 to 100. The default is 1.

NOTE: The parentheses in the instruction syntax can be omitted if all three of the above parameters are omitted.

`$pic[,]` Array of picture data, including a header string. The data in this array is assumed to have been created by the VGETPIC program instruction.

`r,c` Row and column indexes into the array "**`$pic[,]`**", indicating where the header string and picture data are. The defaults are 0 for both "`r`" and "`c`". The header string is in the element "**`$pic[r,c]`**". The picture data starts at element "$pic[r+1,c+1]".

`x0, y0` Optional real-valued expressions specifying the coordinate of the lower-left corner of the image where the picture is to be stored. If not specified, the starting coordinate stored in the header string is used. "`x0`" and "`y0`" are in pixel units and must be in the image. The maximum value is the size of the virtual frame buffer the image data is being written to. Virtual frame buffer sizes are configured

with the DEVICE instruction. See the *AdeptVision User's Guide* for details.

Details

This instruction provides a means for replacing picture data extracted using the VGETPIC instruction. Normally, this is used for reviewing an old picture. However, for diagnostic purposes, saved camera images may be VPUTPIC'd back into the image memory and reprocessed.

To restore an old image for reprocessing, a "quick frame grab" (VPICTURE in mode #2) should be performed first. Then the saved grayscale and binary image may be restored via a type #1 VPUTPIC, or a binary image may be restored via a type #2 VPUTPIC. The VWINDOW instruction or a VPICTURE in mode 0 (repicture) processes the data in the binary image. The VWINDOWI and ruler instructions use either the binary or the grayscale image, depending on the type of operation.

If all the optional parameters are defaulted, all the pertinent information is taken from the header string. However, a VPUTPIC of a saved grayscale image places the image in the frame store currently selected. (See the VSELECT program instruction.)

See the description of the VGETPIC instruction for detailed information on the header string.

Example

Reduce the full grayscale image by a factor of 4 in both the X and Y directions and then put it back, zoomed up by the same amount. Use averaging (mode #1) when reducing the image.

```
VGETPIC (,,4,1) $picture[,]
VPUTPIC (,,4) $picture[,]
```

Restore the same image, but at its reduced size. The image is repeated 16 times to fill the Vision display window.

```
FOR y0 = 1 TO 480 STEP 480/4
    FOR x0 = 1 TO 640 STEP 640/4
        VPUTPIC (,,1) $picture[,], x0, y0
    END
END
```

Related Keywords

VDISPLAY (monitor command and program instruction)
VGETPIC (program instruction)
VSELECT (program instruction)

Syntax

VQUEUE (camera)

Function

Display object information for any objects queued in the vision system awaiting retrieval by VLOCATE instructions.

Usage Considerations

This command may be used when a program is executing, but vision model training must not be active.

Parameter

camera — Optional real-valued expression that specifies the virtual camera number. If this parameter is specified and is nonzero, only objects found for that virtual camera are listed. Otherwise, regions found for all cameras are listed starting with those most recently VPICTUREd or VWINDOWed.

The parentheses can be omitted if no virtual camera is specified.

Details

After the vision system has processed an object, the object information generated by the recognition algorithms is stored in the vision system until a VLOCATE instruction is executed. The VQUEUE command displays the following information for each queued object, grouped by the virtual camera number they were found in:

Name	The name of the prototype that matches the object. If no match is found, the object name is “?”.
Verify Percent	Weighted percentage of the prototype boundary that matches the processed image.
Area	The number of camera pixels enclosed by the object’s region boundary.
X	This is the “X” component of the object transformation.
Y	This is the “Y” component of the object transformation.
RZ	This is the “r” component of the object transformation.
Instance ID	Number of the object. Since an image may contain several objects that match the same prototype, this number is provided

to allow a program to unambiguously reference a specific instance. The first object encountered after a VPICTURE or VWINDOW instruction is assigned an ID of 1, and each subsequent object is assigned the next sequential number.

Camera — This is the virtual camera number of the object.

Example

```
VQUEUE (4)

Name       Verify    Area      X      Y        RZ   Instance  Camera
          Percent                                      ID
FLANGE      99.9      2000    -10.1   123.0    30.00    1         4
CASTING    100.0      1234    101.0    49.2   170.14    2         4
?            0.0    165432    765.3    30.9     0.00    3         4
```

Related Keyword

VQUEUE (real-valued function)

Syntax

VQUEUE (camera, $name)

Function

Return the number of objects in the vision system queue.

Usage Considerations

Vision model training must not be active for this operation to execute.

Parameters

camera	Optional real-valued expression that specifies the virtual camera number. The default is 0, meaning all cameras.
$name	Optional string expression that specifies the name of a prototype.

Details

If a prototype name is specified, VQUEUE returns the number of them in the queue for the given virtual camera. If the virtual camera is not specified or is 0, VQUEUE returns the total number of prototype instances in all virtual camera queues. If neither a camera nor a name is specified, the total number of all objects in all queues is returned.

Examples

Determine the total number of objects seen by all virtual cameras and the number seen by virtual camera #3:

```
total.objects = VQUEUE()
object.cnt[3] = VQUEUE(3)
```

Find out the total number of objects matching the prototype “CASTING” that have been seen by all virtual cameras and the number seen by virtual camera #7:

```
casting.cnt = VQUEUE( , "CASTING")
num.castings = VQUEUE(7, "CASTING")
```

Related Keywords

VLOCATE (program instruction)
VQUEUE (monitor command)

Syntax

```
... V.RECOGNITION [camera]
```

Function

Enable or disable prototype recognition by the vision system.

Usage Considerations

A change to this switch takes effect when the next VPICTURE command or instruction, or VWINDOW instruction, is executed.

This is an array of switches—one for each virtual camera. See the *AdeptVision User's Guide* for details on setting switches.

Details

This switch is normally enabled. When V.RECOGNITION is disabled, the vision system operates as if no prototypes have been defined. This is a useful operating mode when calibrating, for example, because then only the centroids of regions are needed. If no prototypes are defined, the state of V.RECOGNITION has no effect.

Example

Disable object recognition (for all virtual cameras)—do only boundary analysis:

```
ENABLE V.BOUNDARIES
DISABLE V.RECOGNITION
```

Related Keywords

VLOCATE (program instruction)
V.SHOW.RECOG (system switch)

Syntax

VRENAME new_name = old_name

Function

Rename a prototype or subprototype.

Usage Considerations

This command may be used even while a main control program is executing.

Parameters

new_name The new name of a prototype or subprototype. If a subprototype is being named, this parameter must have the form “name1:name2” (entered without quotation marks), where “name1” is the name of the prototype containing the subprototype “name2”. Either name1, name2, or both may be new.

old_name The name of an existing prototype or subprototype. See above for the syntax of subprototype names.

Examples

Rename the prototype:

```
VRENAME SPADE = SHOVEL
```

Rename the subprototype:

```
VRENAME GASKET:NOTCH = GASKET:INDENT
```

Details

The type of model being supported is determined by the name, which has been trained as either a prototype or an ObjectFinder model. The name is checked first against the lists of prototypes for backward compatibility. Therefore, you should not use the same name for a prototype model and an ObjectFinder model.

Related Keywords

VDELETE (monitor command and program instruction)
VSHOW (monitor command)
VSHOW (program instruction)
VTRAIN (monitor command and program instruction)

Syntax

```
VRULERI (cam, type, dmode, maxcnt, edge.dir) data[i], mags[j] = ibr

VRULERI (cam, type, dmode, maxcnt, edge.dir) data[i], mags[j] =
                                                    1, x0, y0, len, ang

VRULERI (cam, type, dmode, maxcnt, edge.dir) data[i], mags[j] =
                                          shape, cx, cy, radius, ang0, angn
```

Function

Obtain edge information or graylevels along a line or circular arc in the current image.

Usage Considerations

If the calibration being used includes correction for perspective distortion, the correction is applied to both the placement and results of the ruler.

Adept recommends the first syntax.

Parameters

`cam` Optional real-valued expression indicating the virtual camera number to use for selecting the V.EDGE.STRENGTH parameter that identifies strong edges for a fine-edge (type #2) or fine-pitch (type #3) ruler. The default camera is 1.

`type` Optional real-valued expression specifying the type of ruler:

–2	dynamic binary
–1	raw binary
0	run-length binary (default)
1	graylevel
2	fine-edge
3	fine-pitch

`dmode` Optional real-valued expression indicating the display mode to use when displaying the ruler: –1 = no draw, 0 = erase, 1 = draw solid, 2 = complement, 3 = draw dashed, and 4 = complement dashed. The default is 1 (draw solid).

`maxcnt` Optional real-valued expression specifying the maximum number of values to return. For an edge-finding ruler, this causes processing to stop after the specified number of edges are found. For a graylevel (type #1) ruler, this specifies the exact number of pixel values to return, starting from the start of the ruler. The default is –1, to return as many values as possible.

`edge.dir` Optional real-valued expression specifying the polarity of the edges to be considered by the ruler:

–1 Consider only light-to-dark edges
0 Consider all edges (default)
1 Consider only dark-to-light edges

NOTE: The parentheses in the instruction syntax can be omitted if all five of the above parameters are omitted.

`data[]` Real array into which the transitions detected by the ruler are placed. The data format of this array is:

```
count, color/clipped, value_1, value_2, ...,
                                        value_n
```

"count" is the number of transitions detected by the ruler.

The "color / clipped" value is the starting color of a binary ruler (ruler types # –2, # –1 and #0). For a type #1, type #2, or type #3 ruler, the value is –1 if any part of the ruler is "clipped" by an edge of the frame store. Otherwise, the value is 0.

The values (value_i) represent the data returned by the ruler. The interpretation of these values depends on the shape and type of ruler.

For a linear ruler, the values are (millimeter) distances along the ruler from the start to each transition.

For an arc ruler, the values are angular distances, in degrees, from the start to each transition.

For a graylevel (type #1) ruler, the values are pixel graylevel values along the ruler. The graylevels are not valid if the ruler was clipped, as described above (indicated by –1 as the second value in the array).

`i, j` Optional integer values that identify the first array elements to be defined in "data[ ]" and "mags[ ]", respectively. Zero is assumed for any array index that is omitted. If a multiple-dimension array is specified, only the right-most index is incremented as the values are assigned.

`mags[]` Optional array into which the signed edge magnitudes are placed for each of the edges returned in "data[ ]". For convenience, the "count" and "color / clipped" values are duplicated into the first two

elements defined in this array. This array is valid only for type #2 and type #3 rulers.

ibr Integer value specifying the image buffer region for the ruler. Image buffer regions specify both a size and a frame store (see the description of VDEF.AOI).

shape Real-valued expression indicating how the ruler path is described: 1 for line, 2 for counter-clockwise arc, 3 for clockwise arc. The default is 1 (linear ruler).

x0, y0 Real-valued expressions indicating the coordinates (in millimeters) of the starting point for the linear ruler.

len Real-valued expression specifying the length of the linear ruler, in millimeters.

ang Optional real-valued expression indicating the angle of the linear ruler in degrees, measured counter-clockwise from the X axis, using the start of the ruler as the origin. The default is 0 (horizontal ruler).

cx, cy Real-valued expressions specifying the center of the arc ruler's circle, in millimeters.

radius Real-valued expression specifying the radius of the arc ruler's circle, in millimeters.

ang0, angn Optional real-valued expressions specifying the angular range of the arc ruler, in degrees. Each value defaults to 0. A full circular arc is performed if the two angles are equal.

Details

Ruler types # –1 and #0 operate on binary data. These rulers determine locations, along a directed line or circular arc, of black-to-white or white-to-black transitions. The starting color returned is the color (black=0, white=1) of the pixel nearest to the start of the ruler. Each color change along the ruler is returned as a transition point, measured in millimeters or degrees from the start of the ruler. The transition points returned are accurate within about one pixel.

Ruler types # –2, #1, #2, and #3 operate on grayscale data. These ruler types must be preceded by a VPICTURE in modes # –1, #0 or #2 (that is, any mode but #1—future frame-grab). The setting of the V.BINARY system switch has no effect on these rulers. These ruler types can operate on either of the two grayscale frame stores (see the VSELECT program instruction).

Ruler types # –2, # –1, #0, #2, and #3 are “edge rulers”, and they all return the same type of information. The only differences are the sources of the edge information.

Type # –2 (dynamic binary) rulers use the grayscale frame store. (This is the data visible in VDISPLAY mode 1.) The current values of the threshold parameters V.THRESHOLD and V.2ND.THRESH **at the time the ruler is executed** are used to determine which pixels are considered foreground and which are background. Therefore, a different binary threshold can be used for each ruler.

Type # –1 (raw-binary) rulers use the raw-binary frame store. This is the data visible in VDISPLAY mode 2. This is binary or edge data, depending on the setting of the V.BINARY system switch.

Ruler type #0 (run-length binary) uses processed binary data produced by VWINDOW or VPICTURE in modes # –1 or #0. (This data is shown in VDISPLAY mode #3.) Note that system parameters such as V.MIN.AREA and V.MAX.AREA are used during picture processing, whereas they are not taken into account with ruler type # –1. Again, the image data is binary or edge data, depending on the setting of the V.BINARY system switch.

The edges returned by a type #2 (fine-edge) ruler are based on the graylevels on and near the path of the ruler. The edge detection technique used is more sophisticated than the one used at recognition time. This ruler type is designed to detect edges that are close to perpendicular to the direction of the ruler. If a ruler is far from perpendicular to an edge at the point where the ruler and edge intersect, the ruler is less accurate and less sensitive to that edge. A good rule of thumb is to keep the ruler within 20 degrees of perpendicular for best results. The sensitivity of the ruler to edges is determined by the V.EDGE.STRENGTH parameter. The higher the parameter value, the stronger an edge must be in order to be detected.

The edges returned by a type #3 (fine-pitch) ruler are similar to a type #2 ruler except the edge detection algorithm has been optimized for finding closely spaced edges. For maximum accuracy, this ruler type must be placed as perpendicular as possible to the edges being detected.

For straight rulers, the distance to grayscale edges is accurate with varying degrees of subpixel accuracy, depending mostly on ruler and edge orientation. When both image edge and a linear ruler are exactly horizontal or vertical, the distance to an edge has a standard deviation of about 0.04 pixel (with a maximum error of 0.1 pixel). For angled image edges, the standard deviation is around 0.1 pixel (with a maximum error of 0.3 pixel) if the ruler is kept within 20 degrees of perpendicular to the edge. The repeatability and absolute accuracy will depend on camera and lens quality, as well as calibration and lighting conditions. For arc rulers, the angles are accurate to about half a pixel.

For type #2 or #3 rulers, you can include the "mags[]" parameter to request that the gradient magnitudes for each of the edges found be returned into that array. Positive gradients indicate a dark-to-light edge and negative gradients indicate a light-to-dark edge. The absolute values of these gradient magnitudes are compared against the V.EDGE.STRENGTH parameter to determine if pixels are edge candidates.

For ruler types # –2, # –1, #0, #2, and #3 (that is, edge rulers), the starting element in the array ("data[i]") contains the number of transitions found. Therefore, the number of elements in the array defined after a VRULERI is (data[i] +2). The maximum number of transitions that VRULERI will locate and report is 100 (or the value of the "maxcnt" parameter). Any additional transitions are ignored. If a ruler is requested that lies entirely off the image, zero transitions are returned. If a ruler lies only partially on the image, transitions are detected only for the section of the ruler that is in the image.

When an arc ruler is clipped, the "maxcnt" parameter is ignored and all the transitions along the ruler are returned.

The coordinates of the end points of a linear edge ruler, or the center of an arc ruler, may be outside of the image—but they must be within a 1000-pixel radius of the image origin (bottom left corner) after being transformed from millimeters to image pixels.

Ruler type #1 returns the graylevel values at each pixel along the path of the ruler. The first element in the array is the count of pixels along the ruler. The starting color is always 0. Graylevel rulers can be used to determine the average graylevel along a line.[1] The maximum length of a type #1 ruler is 508 pixels. If a linear ruler starts outside the frame store, the off-frame pixels are padded with zeros. If the end of a linear ruler goes outside the frame store, no padding is done. In that case, the array is just shorter. If any part of a graylevel (type #–1) arc ruler falls outside the frame store, no information is returned other than an indication that the ruler was clipped.

Edge rulers do not report edges that occur within one pixel (two pixels for arc rulers) of the edge of the image, the start of the ruler, or the end of the ruler. This is because the information needed to compute the edge position is not complete in these areas.

The time required to process a ruler depends on the orientation and length of the ruler and the number of transitions encountered. (The parameter "maxcnt" may be used to limit the latter.) In addition, the speed is affected if you are having the ruler displayed or are getting grayscale edge magnitudes returned. All these things affect the speed of each ruler type differently.

[1] The VWINDOWI instruction can be used to extract information about **areas** of an image.

For run-length (type #0) linear rulers, the more horizontal the ruler, the faster it is processed. Exactly horizontal rulers are a special case and process fastest—their speed is independent of their length. Horizontal or vertical type # –2 and # –1 rulers are slightly faster than those with random angle orientations. With the exception of exactly horizontal type #0 rulers, type #–1 rulers are typically the fastest rulers. For fine-edge (types #2 and #3) rulers, orientation has no effect, but they are about half as fast as a run-length binary ruler. The speed of graylevel (type #1) rulers is dependent only on their length and shape.

When no region-oriented information is needed and single-pixel precision is sufficient, the fastest cycle time can be achieved by using a "quick frame grab" (VPICTURE in mode #2) followed by type # –1 rulers. These raw-binary rulers are very fast. Enable the V.BINARY system switch for maximum speed of a VPICTURE in mode #2. If object recognition is not needed, but the filtering effects of parameters such as V.MIN.AREA are needed, the time delay before rulers are processed can be minimized by disabling the V.BOUNDARIES system switch.

The following comments on perspective distortion may be of interest to the expert user.

If the calibration being used includes correction for perspective distortion, the correction is applied to the placement of the ruler. Since perspective transformations of lines are lines, the two end points of a linear ruler are transformed to the perspectively corrected plane. Thus, the results from the ruler (millimeter distances to edge points along the ruler) are automatically corrected for distortion.

The center of an arc ruler is similarly transformed. However, the path of an arc ruler that is followed in the image memory is an upright ellipse. This only approximates the perspectively distorted arc. (A perspectively distorted circle is neither a circle nor an ellipse.) The two radii of the ellipse are determined by the X/Y ratio in the camera calibration data. The angles to edges will be correct in the perspectively corrected plane, but edge points computed based on those angles (using the center and radius of the arc ruler) will not be exact because the ruler shape is not perfectly elliptical when there is perspective distortion.

Examples

Cast a run-length (type #0) binary ruler along a horizontal line starting at the point (30,20) of length 80 and put the detected binary transition information into the array "xx[ ]". For maximum speed: don't draw the ruler (dmode = –1), and stop after finding two edges (maxcnt = 2):

```
VRULERI (, 0, -1, 2) xx[] = 1, 30, 20, 80
```

Cast a fine-edge (type #2) ruler along the same line as in the previous example and find the edges (using grayscale calculations with subpixel precision) and put the signed gradient magnitudes in the array "mags[ ]". Draw the ruler so it can be seen (dmode = 1), along with the edges found. Find only light-to-dark edges (edge.dir = –1):

```
VRULERI (, 2, 1, 2) xx[], mags[] = 1, 30, 20, 80
```

Related Keywords

VDEF.AOI (program instruction)
VFIND.POINT (program instruction)
VWINDOWI (program instruction)
V.BINARY (system parameter)
V.EDGE.STRENGTH (system parameter)

Syntax

```
VSELECT (mode) ibr = camera
```

Function

Select a virtual frame buffer for processing or display and, optionally, select a virtual camera and its calibration to be associated with the frame buffer.

Parameters

`mode`	0 = select a virtual frame buffer 1 = deselect a frame buffer (see Details below)
`ibr`	Real value, variable, or expression that specifies an image buffer region. The last three digits of the image buffer region, which specify a virtual frame buffer, are used to select the virtual frame buffer. The special values 1011 and 1012 select virtual frame buffers 11 and 12, which are full-frame buffers. In addition, for convenience and backward compatibility, the values 1, 11, and 1001 select frame 11, and values 2, 12, and 1002 select frame 12.
`camera`	Optional real value, variable, or expression that specifies a virtual camera number to be associated with the selected frame store. By default, the previous association with the frame store is unchanged if "`camera`" is omitted.

Details

There are 32 (maximum) virtual grayscale frame buffers.[1] The VSELECT instruction selects the virtual frame buffer to access. Operations such as VRULERI, VWINDOWI, VFIND.LINE, VHISTOGRAM, and VGETPIC access the currently selected frame buffer.

By default, the "current" frame buffer is the one selected by a VPICTURE operation. Using VPICTURE, an image can be acquired into any frame buffer.

The VSELECT instruction performs the same function as the "sel_ibr" parameter to VPICTURE. Since VSELECT is a separate instruction, however, a program can acquire two camera images into different frame buffers and then use VSELECT to quickly switch back and forth between them while performing analyses.

[1] The actual number is configurable with the DEVICE program instruction (see the *AdeptVision User's Guide*).

When a frame buffer is selected, the virtual camera number used with the VPICTURE operation that acquired that image is automatically selected, along with the calibration data and all the system switches and parameters associated with that camera. However, if a "camera" parameter is supplied in the VSELECT instruction, the calibration data, switches, and parameters for that camera are selected. This capability is useful with split images, where one half of the image is calibrated differently from the other half.

If a quick frame grab is still acquiring an image into the frame store being selected, the VSELECT instruction causes program execution to wait for the acquisition to complete.

For some vision operations (such as binary correlation), two scratch frame stores are needed. The display frame store will always be used for one. For the other, the system will use any frame store that does not contain valid image data. However, if all frame stores contain valid data, then an error code will be returned. Once a frame store has valid data, it will continue to have valid data until an ENABLE VISION instruction is processed. VSELECT with "`mode`" = 1 will deselect a frame buffer, making it available as a scratch frame store (the optional "`camera`" argument is ignored).

Related Keyword

VPICTURE (monitor command and program instruction)
VDEF.AOI (program instruction)

Syntax

VSHOW proto_name, grip, edge_num

Function

List the defined prototypes or display a vision prototype, a subprototype, or a specific prototype edge in the Vision display window. Edge numbers are optionally shown.

Usage Considerations

The VISION switch must be enabled, the vision processor must be idle, and vision model training must not be active for this instruction to be executed.

This command supports the ObjectFinder tool. Except where noted, the usage is the same as for prototypes.

This command may be used even while a control program is executing.

Parameters

`proto_name` Optional name of a prototype or subprototype to be displayed in the Vision display window. If no name is specified, the list of all defined prototypes is displayed in the Monitor display window.

If a subprototype is desired, this parameter must have the form "name1:name2" (entered without quotation marks), where "name1" is the name of the prototype containing the subprototype "name2".

`grip` Optional real-valued expression that specifies the grip position to be displayed along with the prototype. The default grip value is 0, which means no grip positions are shown. The value –1 causes all grip positions to be displayed. A value in the range 1 to 4 displays the corresponding single grip position.

`edge_num` Optional real-valued expression that indicates:

0	Do not show the edge numbers
–1	Show all edge numbers
>0	Show specific edge number

Details

The type of model being supported is determined by the name, which has been trained as either a prototype or an ObjectFinder model. The name is checked against the lists of prototypes first, for backward compatibility. Therefore, you should not use the same name for a prototype model and an ObjectFinder model.

When a single prototype is specified, its prototype image is displayed in the Vision display window, and a list of its descriptor values is displayed in the Monitor display window, along with the names of any subprototypes associated with the prototype. The prototype is displayed at its nominal location in the Vision display window. This is the location of the first instance of the prototype.

If display of a subprototype is requested, the prototype is displayed in gray and the subprototype, in color.

If the "`grip`" parameter is nonzero, the specified grip position is shown overlaying the prototype in the Vision display window.

If "`edge_num`" is nonzero, one or more edge numbers are shown. You may need to know these edge numbers for the program instructions VEDGE.INFO, VDEF.SUBPROTO, VSUBPROTO, and VGAPS.

The following are explanations of the descriptor values displayed:

Color	Color of the object when it was trained. During subsequent image processing, in order for a region to be recognized as an object, it must have the same color as the prototype. The value of "color" will be either "Black" or "White".
Number of taught examples	Number of images that were used during training to generate the prototype. The more images used, the more reliable the prototype.
Area	Area of the prototype in camera pixels including any holes. This is the area of the first trained instance of the prototype.
Minimum area	Minimum region area for recognition.
Maximum area	Maximum region area for recognition.
Effort level	Effort level associated with the prototype.
Verify percentage	Minimum percentage of the weighted prototype boundary that has corresponding image edges before the prototype is accepted as a correct match.
Camera associations	List of the virtual cameras associated with the prototype.
Symmetry	Special information that is displayed if the part is symmetric.
Subprototypes	List of the subprototypes associated with the prototype.

Example

Display the outline of the prototype “flange” in the Vision display window together with gripper definition #2 (two subprototypes are associated with the prototype: “right_side” and “top.part”):

```
VSHOW flange, 2
    Color:                        White
    Number of taught examples:    5
    Area:                         10521
    Minimum area:                 6000
    Maximum area:                 14000
    Effort level:                 3
    Verify percentage:            70%
    Camera associations:          1,2,4
    subprototypes:                right_side
                                  top.part
```

List all the prototypes currently loaded in the system.

```
VSHOW
              0    5   10   15   20   25   30
              |....|....|....|....|....|....|..
    CASTING    *-*----------------------------
    FLANGE     **-*----------------------------
```

This indicates that “FLANGE” and “CASTING” are the only object prototypes defined. Their virtual camera associations are as shown.

Related Keywords

VDEF.SUBPROTO (program instruction)
VEDGE.INFO (program instruction)
VGAPS (program instruction)
VSHOW (program instruction)
VSUBPROTO (program instruction)

Syntax

VSHOW mode, **$proto_name**, trans_var, grip, edge_num

Function

Display a vision model (ObjectFinder, prototype, or subprototype) and make information about it available through the VFEATURE real-valued function.

Usage Considerations

The VISION switch must be enabled, the vision processor must be idle, and vision model training must not be active for this instruction to be executed.

Parameters

mode	Optional bit field expression indicating which of several possible events should result from this instruction. (Defaults to 0; see below for details.)
$proto_name	String variable to be assigned the name of the vision model accessed (for "get-next" mode), or a string expression representing the name of the vision model to be shown (for "get-specific" mode). If a subprototype is desired, this parameter must have the form "name1:name2", where "name1" is the name of the prototype containing the subprototype "name2".
trans_var	Optional transformation variable to be assigned the location of the prototype in the Vision display window. This parameter is not accepted if a subprototype is being accessed.
grip	Optional real-valued expression that specifies the grip position to be displayed along with the model. The default value is 0, which means no grip positions are shown. The value –1 causes all defined grip positions to be displayed. A value in the range 1 to 4 displays the corresponding single grip position. This parameter is ignored if "no display" mode is requested.
edge_num	Optional real-valued expression that indicates if the model edge numbers are to be shown. The default is 0, which means the numbers are not shown. All edge numbers are shown if "edge_num" is –1. If "edge_num" is a positive number, the number of that edge is shown.

Details

The type of model being displayed is determined by the name, which has been trained as either a prototype or an ObjectFinder model. The name is checked against the lists of prototypes first, for backward compatibility. Therefore, you should not use the same name for a prototype model and an ObjectFinder model.

The VSHOW instruction may be used to display and acquire information about a single model. After each use of the VSHOW instruction, the VFEATURE real-valued function may be used to obtain information about the model. VSHOW is similar to the VLOCATE program instruction, except that VSHOW works with pretaught models and VLOCATE works with objects found by VPICTURE or VWINDOW.

VSHOW optionally displays the specified model. Thus, with display disabled, you can use the instruction to get the names of the loaded models (one at a time) without disturbing the display.

The mode bits are defined as follows (they all default to zero):

Bit 1 (LSB) Show model location constraints (mask value = 1)

Don't display constraints (0): Location constraints are not displayed.

Display constraints (1): Any position or orientation constraints associated with the prototype are shown in the Vision display window. (Mode bit #4 must be 0 so that the prototype is displayed.) After the prototype is VSHOWed, the prototype's nominal location and relative constraints can be seen overlaying live video (use the monitor command "VDISPLAY (camera) –1,1") to see how the prototype's nominal location compares with the object currently in the camera field of view.

Bit 2 Get-next (0) versus Get-specific (1) (mask value = 2)

Get-next (0): If bit #3 is set (get-hole mode, see below), the next hole is returned. If bit #3 is clear and bit #6 is set (get-subproto mode), the next subprototype is returned. If bits #3 and #6 are both clear, the next prototype is returned. If the next prototype or subprototype is requested, the "proto_name" parameter must be a string **variable**. It is filled in with the name of the prototype or subprototype found. A VSHOW in get-first mode (see bit #5 below) must be executed before the "next" prototype can be found. If the last prototype, hole, or subprototype was already returned, VFEATURE(1) returns FALSE after a get-next VSHOW.

Get-specific (1): A specific prototype or subprototype is accessed. Its name must be specified (by "proto_name") as a string **expression**. Bits #3, #5, and #6 are ignored in this case.

Bit 3 Get-prototype (0) versus Get-hole (1) (mask value = 4)

NOTE: Get-hole is not applicable for ObjectFinder models.

This bit is ignored if bit #2 is set (get-specific mode).

Get-prototype (0): Requests information about a prototype. (See bit #2, "Get-next (0)".)

Get-hole (1): VSHOW in this mode requests hole information. In order for get-hole mode to be used, a prototype first has to be VSHOWed. The subsequent get-hole VSHOWs refer to that prototype. Each get-hole VSHOW returns one hole of the prototype. (Note: When bit #3 is set, bit #6 is ignored.)

In get-hole mode, no string variable is needed for the "proto_name" parameter to the VSHOW instruction.

Bit 4 Display (0) versus Don't-display (1) (mask value = 8)

Display (0): The model is displayed in the vision display window. The model is positioned at its nominal location—the location of the first instance of the model in the field of view.

Don't display (1): Don't display the model in the Vision display window.

Bit 5 Get-next (0) versus Get-first (1) (mask value = 16)

This bit is ignored if bit #2 is set (get-specific mode).

Get-next (0): Requests information about the next item. (See bit #2, "Get-next (0)".)

Get-first (1): Return the first model in the system or return the first hole or subprototype associated with the model last VSHOWed. The specific action depends on the settings of bits #3 (get-hole mode) and #6 (get-subproto mode). (The order of the models in the system is determined by the order in which they are created using VTRAIN or loaded into the system from disk.)

Bit 6 Get-proto / hole (0) versus Get-subproto (1) (mask value = 32)

NOTE: Get-proto / hole and Get-subproto are not applicable for ObjectFinder models.

This bit is ignored if bit #2 (get-specific mode) or #3 (get-hole mode) is set.

Get-proto / hole (0): Requests information about a prototype or hole. (See bit #2, "Get-next (0)".)

Get-subproto (1): This mode requests information about a subprototype. In order for this mode to be used, a prototype first has to be VSHOWed. The subsequent get-subproto VSHOWs refer to that prototype. Each get-subproto VSHOW returns one subprototype of the prototype. In this mode, the "**proto_name**" parameter must be a string **variable**.

VSHOW in get-hole or get-subproto mode refers to the prototype most recently VSHOWed, regardless of which program task executed the VSHOW instruction. Consequently, there is possible confusion when more than one program task executes VSHOW instructions. Thus, for predictable operation with these request modes, applications should be organized to have only one program task execute VSHOW instructions.

If the "edge_num" parameter is nonzero, one or more edge numbers are shown. These edge numbers may be needed for use of the program instructions VEDGE.INFO, VDEF.SUBPROTO, VSUBPROTO, and VGAPS.

Examples

Display the model named "casting" and put its location on the display into the transformation "cx":

```
VSHOW 2, "CASTING", cx
```

Create in "$protos[ ]" a list of the models currently loaded:

```
i = 0

VSHOW ^B11000, $protos[i]         ;Get first model

WHILE VFEATURE(1) DO
        i = i + 1
        VSHOW ^B01000, $protos[i] ;Get next in the list
END
```

Related Keywords

VEDGE.INFO (program instruction)
VFEATURE (real-valued function)
VLOCATE (program instruction)
VSHOW (monitor command)

Syntax

```
... V.SHOW.BOUNDS [camera]
```

Function

Enable the special display of the lines and arcs fit to the boundaries of regions.

Usage Considerations

A change to this switch takes effect when the next region is displayed in the Vision display window with a graphical display mode in effect.

This is an array of switches—one for each virtual camera. See the *AdeptVision User's Guide* for details on setting switches.

Details

When V.SHOW.BOUNDS is enabled, the connected sequence of lines and arcs that bound regions are displayed in the Vision display window overlaying the displayed regions. Each connected pair of lines or arcs is joined at a corner, which looks like a little knot. The bounds are displayed in a unique color to distinguish them from other graphics.

If V.SHOW.BOUNDS, V.RECOGNITION, and V.EDGE.INFO are all disabled, the system does not perform arc and line fitting at all, since it is not needed (see below). Thus, when V.RECOGNITION and V.EDGE.INFO are disabled, the V.SHOW.BOUNDS switch controls whether or not arc and line fitting is done. For applications that do not use recognition, disabling V.SHOW.BOUNDS will decrease processing time, because arc and line fitting takes a significant amount of time.

The three conditions that require the system to perform arc and line fitting are:

1. If V.RECOGNITION is enabled and prototypes are loaded in the system, the system must perform arc and line fitting as a normal step in the recognition process.
2. If V.EDGE.INFO is enabled, the system must perform arc and line fitting so that the information is available for VEDGE.INFO requests.
3. If the V.SHOW.BOUNDS switch is enabled, arcs and lines must be fit so that they can be shown in the Vision display window.

Related Keywords

VEDGE.INFO (program instruction)
V.RECOGNITION (system switch)

Syntax

```
... V.SHOW.EDGES [camera]
```

Function

Enable the special display of edges—both the primitive edges that are fit to the boundaries of regions, and the edge points that are found by the finders VFIND.LINE, VFIND.ARC, and VFIND.POINT.

Usage Considerations

A change to this switch takes effect when a VFIND.LINE, VFIND.ARC, VFIND.POINT, VPICTURE (in mode #0 or # –1), or VWINDOW is executed. The Vision display window must have a graphical display mode in effect.

This is an array of switches—one for each virtual camera. See the *AdeptVision User's Guide* for details on setting switches.

Details

When V.SHOW.EDGES is enabled, edges are displayed when certain vision operations are performed. There are two groups of operations that display edges when V.SHOW.EDGES is enabled.

The first group consists of the finders: VFIND.LINE, VFIND.ARC, and VFIND.POINT. When V.SHOW.EDGES is enabled, the edge points that are found in the area of interest of a finder are displayed in the Vision display window. VFIND.LINE and VFIND.ARC display in white the edge points that were used to fit a line or an arc, respectively. Edge points that were filtered out are displayed in gray. VFIND.POINT displays in white all the edge points it found. (VFIND.POINT does not filter out edge points.) The display of edge points by the finders is very costly in terms of processing time, so V.SHOW.EDGES should be enabled only for investigative purposes.

The other operations that display edges when V.SHOW.EDGES is enabled involve boundary analysis. Both VPICTURE (in modes #0 and # –1) and VWINDOW perform boundary analysis. When V.SHOW.EDGES is enabled, the connected sequence of primitive straight lines that bound the regions are displayed in the Vision display window, overlaying the displayed regions. The lines are joined by corners, which look like little knots. The edges are displayed in a unique color to distinguish them from other graphics.

Example

Show the results of subsequent edge fitting:

```
ENABLE V.SHOW.EDGES
```

Related Keywords

VFIND.ARC (program instruction)
VFIND.LINE (program instruction)
VFIND.POINT (program instruction)
VPICTURE (monitor command and program instruction)
VWINDOW (program instruction)

Syntax

```
... V.SHOW.FEATS [camera]
```

Function

Enable the special display of features used for ObjectFinder recognition.

Usage Considerations

A change to this switch takes effect when the next region is displayed in the Vision display window with a graphical display mode in effect.

This is an array of switches—one for each virtual camera. See the *AdeptVision User's Guide* for details on setting switches.

Details

When V.SHOW.FEATS is enabled, the features used for ObjectFinder recognition are displayed in the Vision display window overlaying the displayed regions. Each connected pair of lines or arcs is joined at a corner, which looks like a little knot. The bounds are displayed in a unique color to distinguish them from other graphics.

If V.SHOW.FEATS is disabled, the system does not perform arc and line fitting for ObjectFinder recognition since it is not needed. For applications that do not use recognition, disabling V.SHOW.FEATS will decrease processing time because arc and line fitting takes a significant amount of time.

Related KeywordS

V.SHOW.BOUNDS (system switch)

Syntax

```
... V.SHOW.GRIP [camera]
```

Function

Enable the special display of clear-grip tests.

Usage Considerations

A change to this switch takes effect when the next region is displayed in the Vision display window with a graphical display mode in effect.

This is an array of switches—one for each virtual camera. See the *AdeptVision User's Guide* for details on setting switches.

Details

When V.SHOW.GRIP is enabled, clear-grip tests are shown in a special display mode (for example, VDISPLAY mode #3). That is, each rectangle of a grip definition is displayed as it is tested. Of course, this is performed only when gripper positions have been defined for the prototype and the prototype has been recognized.

Grips are tested in the order of their numbering, 1 through 4, as defined with VDEFGRIP instructions. Once a grip is found to be clear, further testing is halted, because only one clear grip is required. Each grip is modeled by one to five rectangles. The algorithm for testing for clear grips checks each of the rectangles until one is found to be not clear or until all are found to be clear. Consequently, when a grip is not clear, the operator may see only some of the rectangles that define the grip.

Example

Show the clear-grip tests:

```
ENABLE V.SHOW.GRIP
```

Related Keyword

VDEFGRIP (program instruction)

Syntax

```
VSHOW.MODEL (mode) $chars, data[i] = $model_num
```

Function

Display a model—either a correlation template or an Optical Character Recognition (OCR) font—and return information about it, or return information about all the defined templates or OCR fonts.

Usage Considerations

The VISION switch must be enabled, the vision processor must be idle, and vision model training must not be active for this instruction to be executed.

Parameters

`mode` Optional expression that is interpreted as a bit field to control the behavior of this instruction. Currently, only bit 1 (mask value 1) is used. If "`mode`" is 0, the font(s) or template(s) is (are) displayed. If "`mode`" is 1, nothing is displayed. The default is 0.

`$chars` Optional string variable to be assigned the set of characters in the font specified by the parameter "**`$model_num`**". (This parameter is ignored if "**`$model_num`**" refers to all fonts or to a correlation template.)

`data[]` Optional array into which font or template information is to be placed. The amount of information depends on whether the model specified by "**`$model_num`**" (see below) is a font or a correlation template. The information returned further depends on whether all models (that is, all fonts or all templates) or a single model is specified.

If a specific OCR font is referenced:

data[i+0] = Number of characters in the font
data[i+1] = Height of the font in pixels
data[i+2] = Font color: TRUE = black, FALSE = white
data[i+2+1] = Number of trained instances of the first character in "`$chars`"
...
data[i+2+n] = Number of trained instances of the nth character in "`$chars`"

If a specific correlation template is referenced:

data[i+0] = Width of the template in pixels
data[i+1] = Height of the template in pixels

If all fonts or templates are referenced:

data[i+0] = Number of fonts or templates defined
data[i+1] = Number of first font or template defined
...
data[i+n] = Number of nth font or template defined

`i` Optional array index that identifies the first element to be defined in "`data[]`". The default is 0. If a multiple-dimension array is specified, only the right-most index is incremented as the values are assigned.

`$model_num` String variable specifying a font, all fonts, a template, or all templates for display (if `mode` = 0) and return of information.

The string naming a font has the form "FONT_n", where "n" is the number (in the range 1 to 99) of a single font, or "FONT_0" specifies all fonts.

Similarly, the name of a template has the form "TMPL_n", where "n" is the number (in the range 1 to 99) of a single template; "TMPL_0" specifies all templates.

Details

The VSHOW.MODEL instruction may be used to display and make queries about one or all correlation templates in vision memory. The vision system displays a specific template by copying the template from system memory into the frame store currently displayed. The template is centered in the frame store. If all templates are specified for display, a list of the identifying numbers of all known templates is displayed.

The VSHOW.MODEL instruction may also be used to display and make queries about one or all OCR fonts in vision memory. Fonts are displayed in the Vision display window. If all fonts are specified, a list of the identifying numbers of all known fonts is displayed. If a specific font is specified, the characters in the font are displayed along with the following information:

- The approximate height of the font as specified in the VDEF.FONT instruction
- The color of the font characters: black on white or white on black

- The base orientation
  This angle is set when the font is first trained (see the instruction VTRAIN.MODEL). When the font is subsequently trained or recognition / verification is performed (see VOCR), the features in the OCR window are rotated as necessary to match the base orientation.
- For each character in the font, the vision system displays the number of times that the character has been trained

The above information, except for the base orientation, is also returned in the "data[ ]" parameter of this instruction.

Examples

Display a list of all defined templates:

```
VSHOW.MODEL () = "TMPL_0"
```

Get all the information about font "font.num", but don't display it:

```
VSHOW.MODEL (1) $str, data[] = "FONT_"+$ENCODE(/I0,font.num)
```

Related Keywords

VCORRELATE (program instruction)
VDEF.FONT (program instruction)
VOCR (program instruction)
VTRAIN.MODEL (program instruction)

Syntax

```
... V.SHOW.RECOG [camera]
```

Function

Enable the special display of the objects recognized.

Usage Considerations

A change to this switch takes effect when the next region is displayed in the Vision display window with a graphical display mode in effect.

This is an array of switches—one for each virtual camera. See the *AdeptVision User's Guide* for details on setting switches.

Details

This switch is normally enabled so that the operator can see what the vision system is recognizing. That is, when this switch is enabled, the silhouettes of all recognized prototypes are drawn in the Vision display window overlaying the displayed regions. The silhouettes are displayed in a unique color to distinguish them from other graphics. If no prototypes are loaded in the vision system, the switch has no effect.

Example

Show the objects recognized:

```
ENABLE V.SHOW.RECOG
```

Related Keyword

V.RECOGNITION (system switch)

Syntax

```
... V.SHOW.VERIFY [camera]
```

Function

Enable the special display of the verification step in the recognition process.

Usage Considerations

A change to this switch takes effect when the next region is displayed in the Vision display window with a graphical display mode in effect.

This is an array of switches—one for each virtual camera. See the *AdeptVision User's Guide* for details on setting switches.

Details

This switch controls a diagnostic tool. When the switch is enabled, all attempts to verify proposed matches are drawn in the Vision display window, overlaying the displayed regions. That is, as the lines and arcs of the prototype are compared to the edges in the image, they are drawn in the Vision display window. The verified parts are displayed in one color and the unverified parts, in a different color.

Usually, when a verification fails (the proposed match fails), not all of the prototype boundary is drawn. This simply illustrates the early cutoff of the verification process. That is, the amount of the prototype boundary that did not verify exceeded the maximum allowed, according to the prototype edge weights and "verify percentage" threshold.

> **NOTE:** This display option is often very time-consuming. The parameter V.MAX.TIME may have to be increased if the verification of all match proposals is to be seen.

Related Keywords

V.RECOGNITION (system switch)
V.SHOW.BOUNDS (system switch)
V.SHOW.EDGES (system switch)

Syntax

VSTATUS

Function

Display vision system status information in the Monitor display window.

Usage Considerations

This command may be used even while a main control program is executing.

Details

This command displays key vision status information in the Monitor display window for all 32 virtual cameras. However, the status line for a particular camera is not displayed if the run state is **Idle** and the other entries are 0. This reduces the amount of unimportant text displayed. Usually only one or a few cameras have any useful status information.

The following are explanations of the values that are displayed:

Camera number	Virtual camera number.
Run state	Indicates the current operational mode of a particular virtual camera: **Idle**, **Running** (processing images), **Busy** (processing a vision tool), **Waiting** (waiting for a fast digital-input interrupt signal to acquire an image), **Frame held** (for future processing), or **Training**.
Objects recognized	Number of objects matched to a prototype.
Regions not recognized	Number of closed regions that were not matched to a prototype.

Both of the above numerical values are zeroed for a particular virtual camera each time a VPICTURE or VWINDOW operation is performed for that camera.

Example

```
VSTATUS
    Camera   Run      Objects      Regions not
    number   state    recognized   recognized
      1         Idle      79            1
      2      Waiting       0            0
      4      Running       0            0
      5         Idle       0           56
```

Related Keyword

VSTATUS (program instruction)

Syntax

VSTATUS (camera, type) **array[**index**]**

Function

Return vision system status information in a real array.

Parameters

camera	Optional real-valued expression that specifies the virtual camera number. The default is 1.
type	Optional real-valued expression that specifies the group of information wanted. The default is 0. See below for details.
array[]	Real array that receives the data. The data are stored in sequential array elements.
index	Optional integer value that identifies the first array element to be defined in "**array[]**". Zero is assumed if the index is omitted. If a multiple-dimension array is specified, only the right-most index is incremented as the values are assigned.

Details

This is functionally equivalent to the VSTATUS monitor command except that more information is available and it is placed into an array instead of being displayed.

There are two different sets of information that can be requested (with the "type" parameter). The values stored for each set are as follows (the actual element containing a particular item is "index" **plus** the item's "relative index"):

Relative Index	Contents of Array Elements for "type" = 0
0	Run state: 0 = Idle 1 = Running 2 = Training 3 = Frame held 4 = Busy 5 = Waiting
1	Number of objects recognized
2	Number of regions not recognized

The above set of information is the same as that displayed by the VSTATUS monitor command. (See the description of the VSTATUS monitor command for an explanation of the values.)

Relative Index	Contents of Array Elements for "type" = 1
0	Selected frame: 1 or 2
1	Virtual camera: 1 through 32
2	Display mode: –1, 0, 1, 2, 3, 4, or 5
3	Overlay mode: 0, 1, or 2
4	Displayed frame: 1 or 2
5	Displayed virtual camera: 1 through 32
6	Valid image data: 0 (false) or –1 (true)
7	Valid run-lengths: 0 (false) or –1 (true)

The above items are returned when the VSTATUS "type" parameter is 1. The first two items, "Selected frame" and "Virtual camera", are the frame store and virtual camera currently selected. The VPICTURE and VSELECT instructions can change the current frame and camera, as can menu selections made with the mouse.

The next four items from a type #1 VSTATUS describe what is being displayed in the Vision display window. "Display mode" and "Overlay mode" have the same meaning as the corresponding parameters to the VDISPLAY operator. "Displayed frame" indicates which of the two frames is being displayed, assuming one is being displayed (infer from the current display mode). Finally, the "Displayed virtual camera" is the virtual camera associated with the displayed frame.

The last two items from a type #1 VSTATUS indicate whether the selected frame contains valid grayscale image data or valid run-lengths.

"Valid run-lengths" is true if a VPICTURE in mode –1 or 0, or a VWINDOW operation, was performed. If run-lengths are valid, a type #0 rulers and finders instruction may be performed.

The "camera" parameter is not relevant for VSTATUS type #1.

VSTATUS is a "nonsequential" instruction. The distinction between sequential and nonsequential processing applies only when multiple V+ tasks are executing vision instructions. Then, while the vision system is processing a vision instruction for one task and additional vision instructions are queued up from other tasks, another task can execute a VSTATUS instruction, and it will complete immediately (before completion of the instruction that was being processed or of queued instructions).

Example

Store type #0 status information for virtual camera #1 in the array “status[ ]” in elements 10 through 12:

```
VSTATUS (1) status[10]
```

Related Keyword

VSTATUS (monitor command)

Syntax

```
VSTORE  file_spec = model_name, model_name, ...
```

Function

Store in a disk file selected (or all) vision prototypes (and their subprototypes), Optical Character Recognition (OCR) fonts, or correlation templates.

Usage Considerations

The VISION switch must be enabled *and* the vision processor must be idle for this command to be executed.

Parameters

`file_spec` Specification of the disk file into which the models are to be stored. This consists of an optional physical device, an optional disk unit, an optional directory path, a filename, and an optional file extension. Uppercase or lowercase letters can be used.

The current default device, unit, and directory path are considered as appropriate (see the DEFAULT command in the ***V+ Operating System User's Guide***).

If no filename extension is specified, the extension ".VS" is appended to the name given.

`model_name` Optional name of a current prototype, OCR font, or correlation template to be saved in the file. Uppercase or lowercase letters can be used. If no model names are listed, all prototypes are stored. In this case, the equal sign can be omitted.

Font names have the form "FONT_n", where "n" is the number of the font, in the range 1 to 99. The special font name "FONT_0" refers to all the fonts.

Template names have the form "TMPL_n", where "n" is the number of the template, in the range 1 to 99. The special template name "TMPL_0" refers to all the templates.

Details

The type of model being stored is determined by the name, which has been trained as either a prototype or an ObjectFinder model. The name is first checked against the lists of prototypes for backward compatibility. Therefore, you should not use the same name for a prototype model and an ObjectFinder model.

> **NOTE:** When no model names are specified, VSTORE stores *all* the models in memory to the disk file. There is no way to globally store only the ObjectFinder models. If you want to do this, you must store each model individually by specifying the model name. (It will assume all prototypes if no names are given.)

This command saves the specified prototypes, fonts, or correlation templates in the given disk file. Note, however, that prototypes, fonts, and templates may **not** be mixed in a file. Thus, all the model names must be prototypes, or they must all be fonts, or they must all be templates. If no model names are specified, all the prototypes currently in system memory are stored in the file.

As prototypes are stored, their names are displayed in the Monitor display window along with their virtual camera associations. When prototypes are stored, any subprototype definitions are stored along with them.

As fonts or templates are stored, the numbers of the fonts or templates are displayed in the Monitor window. If the model name “FONT_0” is specified, all the defined fonts are stored in the file. Similarly, if the model name is “TMPL_0”, all the defined templates are stored.

As the file is created, the model information is written in a special format for later recall with the VLOAD command or instruction.

Examples

The following command stores all the prototypes from the vision system into a file named “OBJECTS.VS”. This file will contain two prototypes: “casting” and “flange”, with the virtual camera associations shown:

```
VSTORE objects
                  0    5    10   15   20   25   30
                  |....|....|....|....|....|....|..
         CASTING   ***-----------------------------
         FLANGE    *------------------------------
```

The following command stores the prototypes “smd1” and “quad” in a file named “PROTOS.VS” on disk drive “B”:

```
VSTORE B:protos = smd1, quad
                  0    5    10   15   20   25   30
                  |....|....|....|....|....|....|..
         SMD1      *-*----------------------------
         QUAD      **-----------------------------
```

Related Keywords

VLOAD (monitor command)
VLOAD (program instruction)
VSTORE (program instruction)

Syntax

```
VSTORE (lun) $file_spec = $model_name, $model_name, ...
```

Function

Store in a disk file selected (or all) vision prototypes (and their subprototypes), Optical Character Recognition (OCR) fonts, or correlation templates.

Parameters

`lun` Real-valued expression that specifies the logical unit number to be associated with the operation. This must be one of the logical unit numbers for a disk device (see the ATTACH instruction in the *V+ Language Reference Guide*). The logical unit number used must **not** already be in use by the program for another disk access.

`$file_spec` String expression that specifies the disk file into which the models are to be stored. This consists of an optional physical device, an optional disk unit, an optional directory path, a filename, and an optional file extension. Uppercase or lowercase letters can be used.

The current default device, unit, and directory path are considered as appropriate (see the DEFAULT command in the *V+ Operating System User's Guide*).

If no filename extension is specified, the extension ".VS" is appended to the name given.

`$model_name` Optional string expression that specifies the name of a current prototype, OCR font, or correlation template to be saved in the given file. Uppercase or lowercase letters can be used. If no model names are listed, all prototypes are stored. In this case, the equal sign can be omitted.

Font names have the form "FONT_n", where "n" is the number of the font, in the range 1 to 99. The special font name "FONT_0" refers to all the fonts.

Template names have the form "TMPL_n", where "n" is the number of the template, in the range 1 to 99. The special template name "TMPL_0" refers to all the templates.

Details

The type of model being stored is determined by the name, which has been trained as either a prototype or an ObjectFinder model. The name is first checked against the lists of prototypes for backward compatibility. Therefore, you should not use the same name for a prototype model and an ObjectFinder model.

> **NOTE:** When no model names are specified, VSTORE stores *all* the models in memory to the disk file. There is no way to globally store only the ObjectFinder models. If you want to do this, you must store each model individually by specifying the model name. (It will assume all prototypes if no names are given.)

Like the VSTORE monitor command, this instruction saves the specified prototypes, fonts, or templates in the given disk file. Note, however, that prototypes, fonts, and templates may **not** be mixed in a file. Thus, all the model names must be prototypes, or they must all be fonts, or they must all be templates. If no model names are specified, all the prototypes currently in system memory are stored in the file. If the model name “FONT_0” is specified, all the defined fonts are stored in the file. Similarly, if the model name is “TMPL_0”, all the defined templates are stored.

When prototypes are stored, any subprototype definitions are stored along with them. As the file is created, the prototype or font information is written in a special format for later recall with the VLOAD command or instruction. The IOSTAT real-valued function can be used after this instruction to determine if any error occurred during the disk operation.

> **NOTE:** The application program must not have attached the logical unit since the VSTORE instruction automatically attaches and detaches the logical unit.

Examples

The following instruction stores all the prototypes from the vision system into a file named “OBJECTS.VS” on the default system disk. Logical unit number 5 is associated with the operation. That logical unit number could be used to check for any error during the operation (see below):

```
VSTORE (5) "objects"
```

The following example stores a single font in the file “FONT.VS”. The number of the font is defined by the real variable “font.num”, which is used to construct the name of the font in the form “FONT_n”:

```
VSTORE (5) "font.vs" = "FONT_"+ENCODE(/IO, font.num)
```

The following example stores the prototypes “smd1” and “quad” in a file named “PROTOS.VS” on the hard disk (“C”). Logical unit number 6 is associated with the operation and is used to check for a successful completion:

```
$pname1 = "smd1"
VSTORE (6) "C:PROTOS" = $pname1, "quad"
IF IOSTAT(6) < 0 THEN
    TYPE /C1, "VSTORE failure: ", $ERROR(IOSTAT(6)), /C1
    HALT
END
```

Related Keywords

VLOAD (monitor command)
VLOAD (program instruction)
VSTORE (monitor command)

Syntax

```
... V.STROBE [camera]
```

Function

Enable the firing of a strobe light in synchronization with taking pictures (VPICTUREs).

Usage Considerations

A change to this switch takes effect with the next VPICTURE command or instruction.

This is an array of switches, one for each virtual camera. See the *AdeptVision User's Guide* for details on setting switches.

There are two strobe outputs. Strobe 1 is used for physical cameras 1 and 3. Strobe 2 is used for physical cameras 2 and 4. See the *Adept MV Controller User's Guide* for cable details.

Details

When V.STROBE is enabled for a virtual camera, the strobe light is fired when a VPICTURE operation is performed for that camera. Strobe lights are usually used to take pictures of moving objects without getting a blurred image. For example, if objects are riding on a moving conveyor belt, a trigger mechanism (such as a photoelectric cell) may be connected to a digital input line that signals a V+ application program when a VPICTURE instruction should be executed.

> **NOTE:** For the fastest, most consistent response to an external event, the fast digital-input interrupt line should be used. For more information, see the description in this manual of the **V.IO.WAIT** system parameter.

The timing of the strobe signal within the picture acquisition and with respect to the VPICTURE request depends on the system parameter V.SYNC.STROBE. This parameter determines if the strobe is fired synchronously or asynchronously with respect to the camera read-out cycle, and if there is to be a reset of the camera's vertical drive signals. (See the description in this manual of **V.SYNC.STROBE** for details.)

The following constraints on the strobe device must be met for successful strobe operation:

1. The combination of the flash latency (from trigger input) and the flash duration must not exceed 100 microseconds.

2. The polarity required by the strobe device must match the strobe polarity that is specified in the camera model. All the default camera models are set for "active high". This can be changed with the DEVICE instruction.
3. The duration of the strobe signal (120 microseconds) must be acceptable to the strobe device.

A VPICTURE instruction in "wait" mode normally waits for the acquisition of the camera image to finish before the next instruction in the V+ program starts to execute. However, if the V.STROBE switch is enabled, the VPICTURE "completes" after the strobe signal, letting the V+ program resume earlier than usual. That is done because the strobe light is assumed to be sufficiently bright to freeze the image, even if the camera is mounted on a robot and the next instruction is a robot motion. If this assumption is not valid for the application, the application program should execute a VWAIT instruction after the VPICTURE and prior to any other instructions that may cause the image to change.

Simultaneous with the firing of the strobe light, the positions of all the encoders connected to the Adept system are automatically "latched". That is, the positions are read and stored internally. In addition to external encoders (for example, those on conveyor belts), motor encoders (with the AdeptMotion VME option) and the joint encoders on the Adept robot are latched when the strobe light fires.

The latched encoder positions can be accessed with the V+ DEVICE function. (For version 10.x and older: If an object in the image has been VLOCATEd, the position of external encoder #1 is also available as VFEATURE(8).) The position of a belt encoder may be used with the V+ belt-tracking features to locate and pick up objects from a moving belt. The V+ LATCH and #PLATCH functions return the latched robot position as a transformation and as a precision point, respectively. (See the *V+ Language Reference Guide* for details on the DEVICE, LATCH, LATCHED, and #PLATCH functions.)

Example

Enable the strobe light for virtual camera #1:

```
ENABLE V.STROBE[1]
```

Related Keywords

VPICTURE (monitor command and program instruction)
V.IO.WAIT (system parameter)
V.SYNC.STROBE (system parameter)

Syntax

```
VSUBPROTO ver_percent, trans_var = subproto_name, edge_num
```

Function

Determine the percentage of an edge or subprototype that was verified during recognition. Also, this instruction can have the prototype position refined, based on only a subprototype or a single edge, producing an adjusted location for the prototype.

Usage Considerations

The VISION switch must be enabled, the vision processor must be idle, and vision model training must not be active for this instruction to be executed.

The V.LAST.VER.DIST system parameter must have been nonzero when the last VPICTURE or VWINDOW was performed to make the necessary information available to the VSUBPROTO instruction.

The VSUBPROTO instruction refers to the object most recently VLOCATEd, regardless of which program task executed the VLOCATE instruction. Consequently, for predictable operation, applications should be organized to have only one program task execute VLOCATE instructions.

Parameters

`ver_percent` Optional real-valued variable that is to be assigned the percentage of the subprototype or edge that was verified (0 to 100).

`trans_var` Optional transformation variable to be assigned the refined location of the prototype object.

NOTE: "`ver_percent`" and/or "`trans_var`" must be specified.

`subproto_name` Optional string expression specifying the name of the subprototype in the form "name1:name2", where "name1" is the name of the prototype last VLOCATEd and "name2" is the name of the subprototype.

`edge_num` Optional real-valued expression that specifies the edge number of the prototype or subprototype for which information is requested. If "`subproto_name`" is specified, "`edge_num`" is optional and refers to an edge of the subprototype, using its edge numbering. If "`edge_num`" is omitted (or has the value 0), the entire subprototype is referenced.

If "subproto_name" is not specified, "edge_num" must be specified (and cannot be zero), and refers to an edge in the last prototype VLOCATEd, using the edge numbering for that prototype.

Details

After a prototype has been recognized and VLOCATEd, the VSUBPROTO instruction may be used to find what percentage of a subprototype or individual edge was verified. The verify percentage returned is an unweighted quantity. That is, it is not adjusted by the edge weights that the user is able to assign during training.

This instruction also can be used to refine the position information for the prototype, based on only a subprototype or a single edge. The adjusted location for the prototype is returned via the parameter "trans_var". The adjustment computation is based on the final verification of the prototype. (Information is retained in the vision system, recording which portion of each prototype edge was verified by which portion of each image edge. The position refinement calculations account for the lengths of the edges verified, their positional variance [based on the training instances], the verified corners of the edges, and their positional variances.)

If you are interested in only one edge or a few edges, the VFIND.LINE and VFIND.ARC instructions provide greater subpixel location accuracy.

You can use the VSHOW command or instruction to determine edge numbers for the "edge_num" parameter in the VSUBPROTO instruction.

VSUBPROTO provides the requested information only if it refers to the most recent picture taken (with a VPICTURE or VWINDOW operation) and the most recent object located (with VLOCATE). Also, the V.LAST.VER.DIST system parameter must have been nonzero when the VPICTURE was performed. Otherwise, the needed information is lost and VSUBPROTO produces the error message "*Information not available*".

Examples

The following examples show all the valid combinations of the input parameters for the instruction (that is, the parameters on the right of the equal sign). For each example shown, one or the other (but not both) of the output variables "ver" or "vloc" could be omitted:

```
; Consider all of the subprototype FLANGE:ASIDE
    VSUBPROTO ver, vloc = "FLANGE:ASIDE"
; Consider only edge #2 of the subprototype FLANGE:BSIDE
    VSUBPROTO ver, vloc = "FLANGE:BSIDE", 2
; Consider only edge #16 of the latest prototype
    VSUBPROTO ver, vloc = , 16
```

Related Keywords

VDEF.SUBPROTO (program instruction)
VGAPS (program instruction)
VSHOW (monitor command)
VSHOW (program instruction)
V.LAST.VER.DIST (system parameter)

Syntax

VSUBTRACT (cam, type, dmode) dest_ibr = **src1_ibr, src2_ibr**

Function

Subtract two binary or grayscale images.

Parameters

cam	Selects the virtual camera to supply threshold values used in creating the binary image.
type	Optional integer value indicating the type of subtraction: 1 for binary, 2 for grayscale averaging, or 3 for grayscale subtraction. The default is 1 (binary).
dmode	Optional real-valued expression specifying the display mode for this operator. The choices are: –1 = no draw, 0 = erase, 1 = draw solid, 2 = complement, 3 = draw dashed, and 4 = complement dashed. The default is 1 (draw solid).
dest_ibr	Optional integer value specifying the image buffer region that will receive the result of the image subtraction. Image buffer regions specify both a size and a frame store (see the description of VDEF.AOI). If specified, the resulting image is stored in this image buffer region. Otherwise, the result replaces the source image buffer region (**src1_ibr**).
src1_ibr **src2_ibr**	Integer values specifying the image buffer regions to subtract. The image buffer regions' AOIs must have been defined with a VDEF.AOI instruction. **src2_ibr** will be subtracted from **src1_ibr**.

Details

Image subtraction may be useful to subtract fixtures from an image or to inspect a part by comparing it with the stored image of a golden part. Of course, precisely repeatable placement of the parts is necessary in this latter case.

Image subtraction is also useful in combination with other image transformations such as the VCOPY and VMORPH instructions. When subtracting binary images, the two source image buffer regions are exclusive ORed (XOR) and the result is stored in "dest_ibr". The underlying grayscale image is left unmodified.

Grayscale averaging is defined as follows:

```
dest_ibr = (src1_ibr - src2_ibr + 128) / 2
```

The "+128" normalizes the output so that no difference has the mid-gray value 64. When the "**src1_ibr**" pixel value is greater than the "**src2_ibr**" value, the result is light (more than 64). Otherwise, the result is dim (less than 64).

Grayscale subtraction subtracts the graylevel values in **src2_ibr** from **src1_ibr** and clips the result at 0 so negative values are not created. **src1_ibr** and **src2_ibr** must be in different frame stores.

If "**src1_ibr**" is the same as "**src2_ibr**", a binary subtraction simply zeros the binary frame store. A grayscale subtraction would fill the frame store with a uniform brightness of 64.

The smaller the area of the image to be processed, the faster the subtraction executes.

Example

Subtract virtual frame buffers #11 and #12 and store the result in frame #11:

```
VSUBTRACT (,1) 1011 = 1011,1012
```

Related Keywords

VADD (program instruction)
VCOPY (program instruction)
VEDGE (program instruction)
VTHRESHOLD (program instruction)

Syntax

```
... V.SUBTRACT.HOLE [camera]
```

Function

Determine whether or not hole areas are to be subtracted from region areas.

Usage Considerations

A change to this switch takes effect when the next VPICTURE command or instruction, or VWINDOW instruction, is executed.

This is an array of switches—one for each virtual camera. See the *AdeptVision User's Guide* for details on setting switches.

Details

If V.SUBTRACT.HOLE is enabled, region areas do not include the areas of any holes in the regions. Otherwise, region areas do include the hole areas. This difference changes the effects of the values of V.MIN.AREA, V.MAX.AREA, V.MIN.HOLE.AREA, and the minimum / maximum areas associated with prototypes. Furthermore, the areas reported by VQUEUE and VFEATURE are affected by the setting of this switch.

When operating in grayscale (nonbinary) mode, an object region in the image consists of edges (so most of the object area is considered to be a hole). Therefore, if you need to monitor the total area of an object when operating in grayscale mode, it is useful to disable V.SUBTRACT.HOLE (item B). With this switch disabled, any break in the edge would cause the hole area to merge with the background and would be immediately discounted in the area calculation (item C). With V.SUBTRACT.HOLE enabled, the hole area would *always* be subtracted from the total area (item A). A small break in the edge may not cause a significant enough difference in the total area to be detected.

A

V.SUBTRACT.HOLE enabled
Total area=edge region only
(gray area is excluded)

B

V.SUBTRACT.HOLE disabled
Total area=edge region + hole
(gray area is included)

C

V.SUBTRACT.HOLE disabled
Break causes hole area to merge with background. Total area is significantly reduced.

This switch does *not* affect the calculations enabled by the system switches V.CENTROID, V.PERIMETER, V.MIN.MAX.RADII, and V.2ND.MOMENTS.

Example

Subtract hole areas when computing region areas:

```
ENABLE V.SUBTRACT.HOLE
```

Related Keywords

V.MAX.AREA (system parameter)
V.MIN.HOLE.AREA (system parameter)
V.MIN.AREA (system parameter)

Syntax

```
... V.SYNC.STROBE [camera]
```

Function

Select synchronous or asynchronous firing of a strobe light when a picture is taken (that is, when a VPICTURE is executed).

Usage Considerations

A change to this parameter takes effect when the next VPICTURE command or instruction is executed.

The V.STROBE switch must be enabled in order for the V.SYNC.STROBE parameter to have any effect.

This is an array of parameters—one for each virtual camera. See the *AdeptVision User's Guide* for details on setting parameters.

Details

The value of this parameter determines when a strobe light should be fired with respect to the camera read-out cycle. The choices are synchronous, asynchronous, and asynchronous with reset.

If V.SYNC.STROBE is set to 2, strobing is asynchronous and the camera vertical drive is reset.

If V.SYNC.STROBE is set to 1, strobing is synchronous and image quality is highest.

If V.SYNCH.STROBE is set to 0, strobing is asynchronous and response time is shortest.

Only the values 0, 1, or 2 can be assigned to V.SYNC.STROBE. The default setting is 0.

Normally, the cameras used with AdeptVision VXL comply with the interlaced, RS-170 video standard. Every 60th of a second, one field (even or odd) is read from the camera. A full frame consists of two fields: an even field and an odd field.

When the value of the system parameter V.SYNC.STROBE is 2, the vertical drive will be reset when a picture is taken. The strobe signal will be output at the same line as for synchronous strobe (V.SYNC.STROBE of 0). This line is defined in the camera model as “delay_strobe”.

CAUTION: Only some cameras have this feature. And there are always some internal or external switches that need to be set properly for this mode to operate. See your camera's manual for details.

Camera model 2 supports this mode. Camera models 3, 4, and 5 MUST have this mode selected.

When V.SYNC.STROBE is set to 0, the strobe light is fired asynchronously—as soon as a VPICTURE request occurs. However, for interline transfer CCD cameras, there is a specific time during vertical blank when charges are transferred out of the pixels. There can be no strobing during this time. Therefore, there is a "no-strobe" region defined that prevents strobing. If the picture is requested (triggered) during the no-strobe time, the strobe is postponed until just after the no-strobe region. Since this region is 1.25ms long, there can be a worst-case latency from picture request to image acquire of 1.25ms when using this mode. Note also that if the V^+ program containing the VPICTURE instruction uses an external signal to initiate the VPICTURE request, a delay occurs between the external event and firing of the strobe. (See the description of the V.IO.WAIT parameter for information about this "activation" delay.)

If V.SYNC.STROBE is set to 1, the strobe light is fired synchronously, always at a fixed time with respect to the camera read-out cycle. Normally, this time is during the read-out of the 4th horizontal line. Sometimes with CCD cameras, firing a bright strobe light can cause a ghost image to appear in one field of the image. This is caused by excessive light that leaks into the vertical shift registers. This ghosting can be minimized or even eliminated by turning down the aperture of the camera lens to reduce the amount of light entering the camera. If this is not possible, the DEVICE instruction can be used to set the strobe output at line 20.

By firing the strobe at this time, ghosting never occurs and overall image quality is best. However, since the light is strobed immediately after a field transfer, the frame grabber must wait one field time before acquiring a frame, because the first field will not have been exposed to the strobe light. This adds a constant 16.7 milliseconds to the execution time of a VPICTURE. Consequently, using this method, the fastest ping-pong processing rate is 20 frames per second instead of 30.

Another problem with firing the strobe at a fixed time is the variable time between the VPICTURE request (or the external event triggering the VPICTURE) and the firing of the strobe. The worst-case delay is 16.7 milliseconds. This worst-case delay should be planned for if objects in the scene are moving fast with respect to the camera field of view.

What is the effect of a worst-case delay of 16.7 milliseconds? Consider, for example, a field of view that is 6.4 centimeters wide. Then each pixel is 0.1 millimeter wide.[1] If objects in the field of view are moving at the (high) speed of 1 meter per second, a worst-case delay of 16.7 milliseconds corresponds to a shift of 167 pixels.[2] Since the frame store is 640 pixels wide, a 167-pixel shift could shift part of the object being analyzed out of the field of view.

When you are using a strobe light synchronously, you should perform the above calculations for your application and adjust the setup accordingly. For example, a camera lens with a shorter focal length may be needed to increase the field of view. Or, increase the distance between the camera and objects to be viewed. Also, the vision tools used to locate the object in the image will need a wider search space along the direction of travel of the object.

Example

Select asynchronous strobing for camera #1:

```
V.SYNC.STROBE[1] = 0
```

Related Keywords

VPICTURE (monitor command and program instruction)
V.IO.WAIT (system parameter)
V.STROBE (system switch)

[1] 6.4 cm per 640 pixels = 0.01 cm / pixel = 0.1 mm / pixel
[2] (0.0167 second) * (1000 mm / second) / (0.1 mm / pixel) = 167 pixels

Syntax

```
VTHRESHOLD (cam, type, dmode) dest_ibr = src_ibr
```

Function

Threshold a grayscale image, producing a binary image.

Parameters

`cam` Optional integer specifying the virtual camera that will supply values for V.THRESHOLD and V.2ND.THRESHOLD.

`type` Optional integer identifying the image type.

–1 Same result as type 1 (see below). Allows the switch V.BACKLIGHT to be passed as the type value.

0 Reverse the output polarity (substitute a binary bit value of 0 for each 1, and a binary bit value of 1 for each 0).

1 Default. No change to output polarity.

`dmode` Optional real-valued expression specifying the display mode for this operator. The choices are: –1 = no draw, 0 = erase, 1 = draw solid, 2 = complement, 3 = draw dashed, and 4 = complement dashed. The default is 1 (draw solid).

`dest_ibr` Optional integer identifying the image buffer region that will receive the thresholded values.

`src_aoi` Optional integer identifying the image buffer region that will be thresholded.

Details

This instruction thresholds the frame store with the given threshold(s). This is useful after a VCONVOLVE, VADD, or VSUBTRACT operation. It may also be used to threshold one part of an image differently from another.

The effect of a VTHRESHOLD operation is visible in VDISPLAY mode #2. It does not affect the associated grayscale image.

Example

```
cam.virt = 1
PARAMETER VTHRESHOLD[cam.virt] = 54

;Threshold the current image at graylevel 54

VTHRESHOLD (cam.virt)
```

Related Keyword

VEDGE (program instruction)

Syntax

```
... V.THRESHOLD [camera]
```

Function

Set the camera grayscale value that separates black pixels from white pixels.

Usage Considerations

Changing this parameter immediately affects the video output of the camera interface board, assuming the VISION switch is enabled.

This is an array of parameters—one for each virtual camera. See the *AdeptVision User's Guide* for details on setting parameters.

Details

This parameter sets the software threshold that is used to determine whether a camera pixel is to be interpreted as being white or black. All pixels with an intensity less than or equal to this threshold are set to black, and pixels with intensities greater than this value are set to white.

The V.2ND.THRESH system parameter provides a dual-threshold capability. See the description of V.2ND.THRESH for more information. V.THRESHOLD and V.2ND.THRESH are for use during **binary** image processing. For grayscale processing, there is an edge threshold parameter called V.EDGE.STRENGTH.

The correct value for V.THRESHOLD depends on the particular application. The VAUTOTHR program instruction may be used in most cases to automatically determine the best threshold.

V.THRESHOLD must be assigned an integer value in the range 0 to 127, inclusive. It has the initial value 63.

Example

Set all grayscale pixel values above 100 to white, and all others to black:

```
PARAMETER V.THRESHOLD = 100
```

Related Keywords

VAUTOTHR (monitor command and program instruction)
V.2ND.THRESH (system parameter)
V.BINARY (system parameter)

Syntax

```
... V.TOUCHING [camera]
```

Function

Determine whether or not objects may be touching in the image.

Usage Considerations

This switch is for use only with prototype recognition. V.RECOGNITION must be enabled for this switch to have any effect.

A change to this switch takes effect when the next VPICTURE command or instruction, or VWINDOW instruction, is executed.

V.TOUCHING is assumed to be enabled if the V.OVERLAPPING system switch is enabled. That is, the actual setting of V.TOUCHING is then ignored.

This is an array of switches—one for each virtual camera. See the *AdeptVision User's Guide* for details on setting switches.

Details

If objects touch in the image, their regions merge into a single region. In this case, V.TOUCHING should be enabled so that the vision system will attempt to recognize multiple objects per region. Otherwise, the system will recognize at most one object per region.

The V.TOUCHING and V.DISJOINT switches affect the interpretation of the "how_many" parameter to the VPICTURE and VWINDOW instructions. That parameter specifies the maximum number of objects the vision system should try to recognize. V.TOUCHING affects how the "how_many" parameter applies to each region in the image, whereas V.DISJOINT affects how it applies to the image as a whole. If the V.TOUCHING switch is enabled, up to "how_many" objects will be recognized per region.[1] If V.TOUCHING is disabled, at most one object will be recognized per region in the image.

[1] The V.TOUCHING switch is automatically considered to be enabled whenever the V.OVERLAPPING switch is enabled.

The following table summarizes the relationship between V.TOUCHING, V.DISJOINT, and the "how_many" parameter to VPICTURE and VWINDOW.

V.TOUCHING	V.DISJOINT	Number of Objects per Region	Number of Objects per Scene
Off	Off	1	No limit
Off	On	1	how_many
On	Off	how_many	No limit
On	On	how_many	how_many

Example

Inform the vision system that objects may touch in the image:

```
ENABLE V.TOUCHING
```

Related Keywords

V.BOUNDARIES (system switch)
V.DISJOINT (system switch)
V.OVERLAPPING (system switch)
V.RECOGNITION (system switch)

Syntax

VTRAIN $prototype, shape, cx, cy, width, height, ang

VTRAIN $prototype, ibr

Function

Initiate training of the prototype whose name is specified.

Usage Considerations

The command and the instruction both require that all virtual cameras be idle (that is, no VPICTURE or VWINDOW operation can be executing).

Prototype training is sensitive to camera calibration. Before training prototypes, mount, focus, and securely fix your cameras and lenses. Then calibrate the camera(s) that will be used to train and recognize the prototypes. The same camera(s) with the same calibration and position used to train a prototype must be used to recognize the prototype.

Parameters

$prototype Name of the prototype to be trained. For the monitor command, the name of the prototype must be a string constant (not surrounded by quotes). For the program instruction, however, the prototype name may be a string variable, constant (including quotes), or a string expression.

shape Optional real-valued expression indicating the shape of the window. Currently, the only choice is 1, for rectangular.

NOTE: If any of the following four parameters (cx, cy, width, or height) is specified, all four parameters must be specified.

cx, cy Optional real-valued expressions specifying the center coordinate of the window, in millimeters.

width Optional real-valued expression specifying the width of the window, in millimeters.

height Optional real-valued expression specifying the height of the window, in millimeters.

ang Optional real-valued expression specifying the orientation of the window, in degrees. The default is 0 degrees.

ibr Optional integer value specifying the image buffer region for training a prototype. Image buffer regions specify both a size and a

frame store (see the description of VDEF.AOI). The image buffer region's AOI must specify a rectangular shape.

Details

VTRAIN is used to create a new object prototype or to modify an existing prototype. If the prototype exists, the user may show the vision system a new instance or change the prototype's virtual camera associations, effort level, edge weights, verification threshold, minimum / maximum areas, or position constraints.

A window may be specified when creating a new object prototype or training another instance of a prototype. The window can be used to limit the portion of the image that is considered. (The window specification is the same as that for the VWINDOW instruction, which is used to perform object recognition within windows.)

Once VTRAIN has been initiated, the mouse is used to interact with the system. The user can abort a VTRAIN monitor command at any time by entering Ctrl+C.

See the ***AdeptVision User's Guide*** for a description of the training process.

Example

Initiate vision training for a prototype named "flange":

```
VTRAIN flange
```

Related Keywords

VDEF.AOI (program instruction)
VLOCATE (program instruction)
VSTORE (program instruction)

Syntax

VTRAIN (cam, mode, arg) **$prototype, ibr** = value

VTRAIN cam, mode, arg) **$prototype,** shape, **cx, cy, dx, dy,** ang = value

Function

Initiate training of the prototype whose name is specified.

Usage Considerations

This instruction requires that all virtual cameras be idle (that is, no VPICTURE or VWINDOW operation can be executing).

Prototype training is sensitive to camera calibration. Before training prototypes, mount, focus, and securely fix your cameras and lenses. Then calibrate the camera(s) that will be used to train and recognize the prototypes. The same camera(s) with the same calibration and position used to train a prototype must be used to recognize the prototype.

Parameters

`cam` Optional integer specifying the virtual camera (and associated calibration, switches, and parameters) to use when acquiring an image for defining a prototype. This camera will also be the default camera associated with the prototype (see the *AdeptVision User's Guide* for details on associating cameras with a prototype).

`mode` Optional integer interpreted as follows:

- 0 A new image is acquired when prototype editing begins. Repeated training samples are allowed, and a new picture is acquired for each.
- 1 The existing image is used. This allows you to preprocess an image before training. After one sample, training exits.
- 2 Disable options in the training pull-down menus. "`value`" controls which options are disabled. If "New Example" is disabled, training proceeds similar to "`mode`" = 1.
- 3 Change a prototype parameter but do not train new samples. The parameter to change is specified in "`arg`". The new value for the parameter is specified in "`value`".

`arg` If "`mode`" = 3, a prototype parameter is being changed. This parameter identifies the parameter to change ("`value`" supplies the value):

arg	Parameter	Range (allowed for "value")
1	Verify percent	0 to 100
2	Effort level	0 to 4
3	Min area (in pixels)	1 to (proto area – 1)
4	Max area (in pixels)	(proto area + 1) to image size
5	±X constraint (in mm)	
6	±Y constraint (in mm)	
7	±Rotational constraint	
8	Virtual camera bit mask	hex 1 to F
9	Virtual camera bit mask	hex 10 to FF

`$prototype` Name of the prototype to be trained. For the monitor command, the name of the prototype must be a string constant (not surrounded by quotes). For the program instruction, however, the prototype name may be a string variable, constant (including quotes), or a string expression.

`ibr` Integer value specifying the image buffer region for training a prototype. Image buffer regions specify both a size and a frame store (see the description of VDEF.AOI). The image buffer region's AOI must specify a rectangular shape.

`shape` Optional real-valued expression indicating the shape of the window. Currently, the only choice is 1, for a rectangular window.

NOTE: If any of the following four parameters (cx, cy, dx, and dy) are specified, all four parameters must be specified.

`cx, cy` Real-valued expressions specifying the center coordinate of the window, in millimeters.

`dx` Real-valued expression specifying the width of the window, in millimeters.

`dy` Real-valued expression specifying the height of the window, in millimeters.

`ang` Optional real-valued expression specifying the orientation of the window, in degrees. The default is 0 degrees.

`value` If "`mode`" = 3, the instruction is changing a prototype parameter, and this parameter supplies the new value.

If "`mode`" = 2, the instruction is altering the training pull-down menus, and this parameter is a bit mask indicating which options should be disabled. The mask values for each option are:

Pull Down Item	Mask Value
New example	1
Verify percent	2
Effort level	4
Min / max areas	8
Limit position	16
Edge weights	32
Assign cameras	64

Details

VTRAIN is used to create a new object prototype or to modify an existing prototype. If the prototype exists, the user may show the vision system a new instance or change the prototype's virtual camera associations, effort level, edge weights, verification threshold, minimum / maximum areas, or position constraints.

A window may be specified when creating a new object prototype or training another instance of a prototype. The window can be used to limit the portion of the image that is considered. (The window specification is the same as that for the VWINDOW instruction, which is used to perform object recognition within windows.)

Once VTRAIN has been initiated, the mouse is used to interact with the system. Training initiated with a VTRAIN instruction can be terminated by aborting the program that contains the instruction.

See the *AdeptVision User's Guide* for a description of the training process.

Example

Initiate vision training for a prototype named "flange":

```
VTRAIN flange
```

Related Keywords

VDEF.AOI (program instruction)
VLOCATE (program instruction)
VSTORE (program instruction)

Syntax

```
VTRAIN.FINDER (cam, mode, dmode, arg, arg2, arg3)
                                $model_name, ibr = value, value2, value3
```

Function

Initiate training of the finder model whose name is specified.

Usage Considerations

This instruction requires that all virtual cameras be idle (that is, no VPICTURE or VWINDOW operation can be executing).

Training is sensitive to camera calibration. Before training begins, mount, focus, and securely fix your cameras and lenses. Then calibrate the camera(s) that will be used to train and recognize the prototypes. The same camera(s) with the same calibration and position used for training must also be used for recognition.

Parameter

`cam` — Optional integer specifying the virtual camera (and associated calibration, switches, and parameters) to use when acquiring an image for defining a prototype. This camera will also be the default camera associated with the prototype (see the *AdeptVision User's Guide* for details on associating cameras with a prototype).

`mode` — Optional integer specifying the operating mode for this instruction. See below for details.

`dmode` — Optional real-valued expression that specifies the display mode to use when displaying the border of the window:

–1	No draw
0	Erase
1	Default. Draw solid
2	Complement
3	Draw dashed
4	Complement dashed

`arg, arg2, arg3` — Integers whose meaning depends on the value of the mode parameter. See below for details.

`$model_name` — String containing the name for the model (up to 15 characters).

`ibr` — Integer value specifying the image buffer region for training a model. Image buffer regions specify both a size and a frame store (see the description of VDEF.AOI). The image buffer region's

AOI must specify a nonrotated rectangular shape. For correlation templates, the AOI is reduced, if necessary, to a multiple of four pixels.

`value` Floating point expressions whose meaning depends on the value of the mode parameter. See below for details.

Description of the mode parameter:

This section describes the meanings of the different values used with the mode parameter, and the meanings of other input parameters used with a particular mode. If an input parameter is not shown, it cannot be used for that mode.

Mode 1: Train first instance (new model)
- arg Hierarchical level (0 - 2) is the result of the subsampled image
  - 0 Full resolution
  - 1 One-half resolution
  - 2 One-quarter resolution
- arg2 Verify percent
- arg3 Symmetry override

Mode 2: Display features only
- arg Hierarchical level (0 - 2) is the result of the subsampled image
  - 0 Full resolution
  - 1 One-half resolution
  - 2 One-quarter resolution
- arg2 Edges only (0,1)
  - 0 Features are fit to the edges and the features are displayed
  - 1 Only the results of edge detection are displayed

Mode 3: Set finder model parameters (after last instance)
To train new instances during multi-instance training, use VFINDER with type =2.
- arg Indicates the value:
  - 1 Symmetry of model
  - 2 Unused
  - 3 Verify percent to assign to model
- value The value to assign to the designated parameter

Mode 4: Feature weight setting (also feature deletion)
- arg Feature number (1 to num_features)
  (–1 means all features)
- value New weight to assign
  - 0 deletes the feature (deletion means that the feature is not used)
  - 1 - 100 relative weighting for verification. The feature is still used to form feature-pairs.

Mode 6: Display a model feature
- arg Number of the feature to display

Mode 7: Display a model feature in RED
- arg Number of the feature to display

Mode 8: Accept the last trained additional instance
arg Low weight cutoff (used for multi-instance training and is the threshold on feature weights [on a scale of 0 - 100]; feature weights below the low weight cutoff are set to zero)

Mode 9: Do not accept the last trained additional instance
arg Low weight cutoff

Mode 10: Clear all multi-instance training statistics. Leave the first instance.
arg Low weight cutoff

Mode 11: Apply low-weight filtering
arg Low weight cutoff

Mode 12: Undo the last accepted instance
arg Low weight cutoff

Details

When normal (initial) training is performed, a new model is created and assigned the given input values. Additionally, the ibr is processed to create edge pixels and these are further processed to extract lines and arcs. The lines and arcs are paired up in various ways to form intrinsic "feature pairs". These can be later used to match with similar pairs formed from recognition images to make proposals of instances of the model. These lines, arcs, and pairs define the geometric portion of the model.

There are three system parameters (described below) that control the processing.

1. The system parameter V.EDGE.STRENGTH[vc] controls the threshold for edge points extracted during preprocessing.
2. The system parameter V.MAX.PIXEL.VAR[vc] controls the primary fitting of lines and arcs. A secondary operation performs a least-squares fitting to the edge points, which greatly improves the accuracy.
3. The system parameter V.MIN.LEN[vc] restricts the pairs by setting a minimum length for the features allowed to be used to make pairs.

The system switch V.FIT.ARCS[vc] should normally be enabled, but can be disabled to some advantage in certain trouble cases. If an object is mostly lines, but contains some segments that occasionally fit as arcs due to noise in the image, you can force the system to fit only lines and avoid the additional computation time required to fit arcs.

Input parameters using virtual cameras

All switches and parameters should be at the default settings, except as noted here:

The following settings are required:

PARAMETER V.MIN.AREA[vc] = 4

PARAMETER V.MIN.HOLE.AREA[vc] = 4

The following settings are suggested:

ENABLE V.FIT.ARCS[vc]	(May be disabled if the image consists of mostly lines. See the description above for details.)
V.MIN.LEN[vc] = 10	(See the description above for details.)
V.MAX.PIXEL.VAR[vc] = 2.5	(See the description above for details.)
V.EDGE.STRENGTH[vc] = 9	(See the description above for details.)

Syntax

```
VTRAIN.MODEL (cam, plan) $model_name, $text, ibr

VTRAIN.MODEL (cam, plan) $model_name, $text,
                                    shape, cx, cy, dx, dy, ang
```

Function

Train on a vision "model"—a correlation template or an Optical Character Recognition (OCR) font. For correlation, this instruction defines the template. For OCR, this instruction trains the vision system to recognize characters in a font, or causes the vision system to plan the recognition strategy for a fully trained font.

Usage Considerations

The frame store currently selected or the specified image buffer region must contain a valid picture. Otherwise, an error results.

The VISION switch must be enabled, the vision processor must be idle, and vision model training must not be active for this instruction to be executed.

Font training is sensitive to camera calibration. Before training fonts, mount, focus, and securely fix your cameras and lenses. Then calibrate the camera(s) that will be used to train and perform OCR. The same camera(s) with the same calibration and position used to train a font must be used to recognize characters in the font.

Correlation templates are independent of camera calibration but are sensitive to the image size, which can change by moving the camera, changing lenses, or refocusing a lens. Make sure the camera setup remains unchanged during template training and matching.

Adept recommends the first syntax.

Parameters

`cam` Optional real-valued expression indicating the virtual camera number to use for selecting various parameters for training an OCR font (such as V.THRESHOLD and V.MIN.AREA). The default camera is 1.

`plan` If "`$model_name`" specifies an OCR font, optional real-valued expression that specifies the operation desired. (When planning is requested, only the "`plan`" and "`$model_name`" parameters are considered. All the others can be omitted.)

0 Directs the vision system to perform OCR training

1 Directs the system to plan recognition and to display associated graphics (see below)

–1 Directs the system to plan recognition, but without display of associated graphics (see below)

If "`$model_name`" specifies a correlation template's mode of training, as follows:

0 Default, creates a hierarchical grayscale template

1 No longer used.

2 Creates a nonhierarchical grayscale template

`$model_name` String variable specifying a template for definition, a font for training, or all fonts for planning.

The string naming a font has the form "FONT_n", where "n" is the number (in the range 1 to 99) of a single font, or "FONT_0" specifies all fonts.

The string name of a template has the form "TMPL_n", where "n" is the number of the template, in the range 1 to 99.

`$text` (For OCR training only.) Optional string variable that specifies the characters in the font that are to be trained. The text may contain duplicate characters. Spaces are ignored.

`ibr` Integer value specifying the image buffer region for training a model. Image buffer regions specify both a size and a frame store (see the description of VDEF.AOI). The image buffer region's AOI must specify a nonrotated rectangular shape. For correlation templates, the AOI is reduced, if necessary, to a multiple of four pixels.

`shape` Optional real-valued expression indicating the shape of the window. Currently, the only choice is 1, for a rectangular window.

NOTE: If any of the following four parameters (cx, cy, width, and height) are specified, all four parameters must be specified. They are always needed for defining a template, or training on a font, but they are never needed for planning a font.

`cx, cy` Real-valued expressions specifying the center coordinate of the rectangular training window (or template definition window), in millimeters.

dx	Real-valued expression specifying the width of the window, in millimeters.
dy	Real-valued expression specifying the height of the window, in millimeters.
ang	Optional real-valued expression specifying the orientation of the OCR training window, in degrees. The default is 0 degrees. When defining a correlation template, this must be omitted or have the value 0.

Details

This instruction is used for both correlation templates and OCR fonts.

Correlation Templates

VTRAIN.MODEL defines and fully trains a correlation template in one step. (This is simple compared to prototype or font models, which require a definition step followed by multiple training steps with different example images.) A correlation template is merely a grayscale window extracted from the image. VTRAIN.MODEL analyzes the template and determines the default depth for hierarchical search and skip patterns for each level of hierarchy. The new information is stored in the template.

The position and size of the window is specified with the VTRAIN.MODEL instruction. When VTRAIN.MODEL is executed, the pixels within the window are copied from the frame store into the vision CPU memory. It remains there until it is deleted (via VDELETE).

> **NOTE:** Template widths must be a multiple of 4 pixels and will be reduced if they are not an exact multiple.

You should consider the memory requirements of templates when using correlation in applications. A large template with a size of 400x400 pixels will use 160,000 bytes of vision CPU memory. The FREE monitor command and program instruction report both the amount of vision CPU memory available and the amount used for all models—templates, fonts, and prototypes.

OCR Fonts

After a font has been defined with the VDEF.FONT instruction, the characters in the font can be trained. The vision system is taught to recognize characters by showing it what they look like. Each character should be shown to the system a number of times because the system accumulates statistics on the appearance of the characters. (See the *AdeptVision User's Guide* for an overview of the OCR capability in AdeptVision.)

To train the vision system on characters, you must define an image buffer region that encompasses them in the current image. Together with the image buffer region, you tell the vision system what characters are inside the region, ordered left to right as you would normally read the text. Since the region can be rotated, the "left" edge of the window is drawn a different color for reference. For a rotation of 180 degrees, the "left" edge is actually on the right and the text inside should look upside down. When the vision system analyzes the bounded areas in the image buffer region, it orders the bounded areas by distance from the "left" edge. The characters specified in the "**`$text`**" parameter are assumed to be in this order.

Each character in a font may be trained a different number of times. Also, a character may appear more than once in a training window. The vision system ignores the spacing between characters. Thus, for example, training on the characters "Lot 34" would have the same effect as training on "Lot34".

All characters in the font should be trained 5 to 15 times for reliable operation. The vision system does not allow a character to be trained more then 30 times. When training the vision system, samples of the text representing the range of acceptable appearance should be used. If only a single sample of the text is available, you should still show it to the vision system multiple times, moving it around a little and rotating it slightly in both directions (such as plus and minus 5 to 10 degrees) to provide some variety in appearance.

Note that training on a font is different from prototype training. With font training, there is no system interaction with the keyboard or pointing device. In particular, there is no "Are you sure?" confirmation. Consequently, you must make sure the window encompasses the given text before executing the VTRAIN.MODEL instruction. Otherwise, you could corrupt the system's models of the characters. The Example section below contains a simple V$^+$ routine to help perform font training.

VTRAIN.MODEL returns an error and does not modify the current character models if the number of regions in the window is not identical to the number of characters in the given "$text" parameter. However, if you make a mistake that does not result in VTRAIN.MODEL returning an error, you should VDELETE the font and start over by redefining the font.

Each character must appear as a single bounded binary area. Thus, for example, the characters ";" and ":" cannot be trained. The V.MIN.AREA system parameter should be adjusted to filter out the dots over the letters "i" and "j". Also, if characters touch, the system processes them as a single character. (Automatic splitting of touching characters is not done during training.) VTRAIN.MODEL displays the outline of each bounded area in the Vision display window, so you can see the bounded areas that were analyzed. Before training on a font, you should run the program "set.gain" (in the file SET_GAIN.V2 on the Adept Utility Disk) to adjust the gain and offset. Next, take a picture and find a good threshold

using the VAUTOTHR monitor command or with the pull-down menus. You should also take some sample pictures in VDISPLAY mode 3 and adjust the parameter V.MIN.AREA to filter out dots and noise. Keep V.MIN.HOLE.AREA low enough so that the holes in letters do not disappear.

Characters may be trained in one orientation and recognized in a different one. In fact, characters in a font may be trained in different orientations (such as first horizontally, then diagonally). The system automatically rotates the character features so that the given window orientation matches the "base orientation" of the font.

All characters in a font share the same "base orientation". The base orientation is defined by the AOI used during the first training instance with the font. Thus, if you first trained "A" using an AOI with the angle 30 degrees, and then trained "B" using an AOI with the angle 0, the "B" would be rotated 30 degrees to bring it into alignment with the font's "base orientation" of 30.

In addition to training, VTRAIN.MODEL may be used to make the vision system "plan" a font for recognition. This is not a crucial step, however, because the VOCR instruction automatically plans a font that has not been planned. However, since planning may take a few minutes, you may prefer to control **when** that occurs by executing VTRAIN.MODEL in an initialization program.

When a font is planned and its display is enabled (plan = 1), the characters in the font are displayed at the top of the Vision window as they are planned. Then a discrimination matrix is displayed so that you can watch the effects of the vision system's auto-weighting of features. The color coding in the matrix indicates how the average character (horizontally) scores with respect to the model (vertically). The colors and associated score ranges are listed below, where a score of 100 indicates complete ambiguity (or "perfect match" if in the diagonal, where the character is compared to its own model):

Color	Score
Dark gray	0 to 10
Medium gray	10 to 20
Dark green	20 to 30
Green	30 to 40
Yellow	40 to 50
Orange	50 to 60
Red	60 to 100

The vision system tries to make all the scores outside the diagonal below 30. When a score cannot be reduced more for some reason, a small black plus sign is drawn in the patch of color.

When a font is stored to disk (via VSTORE), the character information accumulated as a result of training is stored, but the results of planning are not stored. The data structures created during planning are very large. Thus, rather than storing and loading large disk files (which in itself takes considerable time), smaller files are used. Consequently, after a font is VLOADed, it must be planned before use.

A 26-character font requires approximately 35 Kb of memory.

Example

This routine is a simple interface for training on a subset of the characters in a font. A window is displayed in the Vision display window overlaying live video so you can position the text inside it. After you press **Enter**, a picture is taken and the characters in the window are processed. To stop training, type any character before pressing the **Enter** key.

```
.PROGRAM train($str)

; ABSTRACT: Initiate font training on an input text string. The font must have
; already been defined.
;
; INPUT PARAM: $str - the text to train on
;
        AUTO ang, cam, cx, cy, dx, dy, font, vlun
        AUTO $ans, $font
        LOCAL data[]

            ; Define these for your specific needs:

        cam = 1                             ;Virtual camera number
        font = 1                            ;Font number: 1 to 99
        cx = 256                            ;Center of the window
        cy = 242
        dx = 350                            ;Width of the window
        dy = 50                             ;Height of the window
        ang = 0                             ;Orientation of the window
        model.ibr = 3001                    ;Define aoi variable

        VDEF.AOI model.ibr = 1, cx, cy, dx, dy, ang

        $font = "FONT_"+$ENCODE(/I0,font) ;Compose name of the font

        VDISPLAY (cam) 2, 1                 ;Display binary image with overlay
        VPICTURE (cam) 2                    ;Do quick frame grab
        ATTACH (vlun, 4) "GRAPHICS"         ;Attach to the vision window
        FOPEN (vlun) "Vision /MAXSIZE 640 480"
        GTRANS (vlun, 1)                    ;Select scaling in real-world mm

        VWINDOWI data[] = , cx, cy, dx, dy, ang
```

```
        TYPE /C1, "Place sample text in window. Press Enter to train, type ", /S
        TYPE "any character and Enter to halt training." /C1


        WHILE TRUE DO
            VDISPLAY (cam) 0, 1
            GTYPE (vlun) 50, 100, $str
            PROMPT "Press Enter to train on "+$str+": ", $ans
            IF $ans <> "" GOTO 10
            VPICTURE (cam) 2
            VTRAIN.MODEL (cam) $font, $str, model.ibr
        END
    10  RETURN
.END
```

Related Keywords

VDEF.AOI (program instruction)
VCORRELATE (program instruction)
VDEF.FONT (program instruction)
VDELETE (monitor command and program instruction)
VOCR (program instruction)
VSHOW.MODEL (program instruction)

Syntax

VWAIT (type) ibr

Function

Delay program execution until processing of a VPICTURE or VWINDOW operation is complete.

Usage Considerations

VWAIT is not considered during vision model training.

Parameters

type Optional integer expression having one of the following values:

0 Wait for the vision system to become idle (default)
1 Wait for the acquire into the frame specified in "ibr" to complete
2 Wait for the acquire into the frame specified in "ibr" to start

ibr Optional integer value specifying an image buffer region. Only the frame store element of the image buffer region is used (see the description of VDEF.AOI). This parameter is ignored for type #0 VWAITs.

Details

A type #0 VWAIT waits for the vision processor to become idle following a VPICTURE or VWINDOW operation.

A type #0 VWAIT is necessary before executing any of the following instructions:

VDEF.FONT	VDEF.MORPH	VGETPIC
VHISTOGRAM	VPUTCAL	VPUTPIC
VSHOW	VSHOW.MODEL	VSUBPROTO
VTRAIN.MODEL		
VDELETE (monitor command only)		

A type #2 VWAIT is essentially a wait for a strobe light or camera shutter to operate.

Example

```
VPICTURE (cam)          ;Take a picture using virtual camera "cam"
VWAIT                   ;Wait until vision processor is idle
```

Related Keywords

VDEF.AOI (program instruction)
VSTATUS (monitor command)
VSTATUS (program instruction)

Syntax

```
VWINDOW (cam, type, dmode, how_many) ibr

VWINDOW (cam, type, dmode, how_many) shape, cx, cy, width, height, ang
```

Function

Perform processing (blob finder, prototype recognition, or ObjectFinder) within a rectangular window in the image.

Usage Considerations

The VWINDOW program instruction does not suspend program execution while the image is being processed. In other words, program execution continues with the next instruction while the vision system processes the window image.

> **NOTE:** Some vision instructions cause an error if they are executed while window processing is still active. You can use the VWAIT program instruction to suspend program execution until the vision processor is idle.

Adept recommends the first syntax.

Parameters

`cam` Optional real-valued expression that specifies a virtual camera number (see below). The default is 1.

`type` Optional real-valued expression indicating the type of the window. Currently, the only choice is 1.

`dmode` Optional real-valued expression specifying the display mode to use when displaying the border of the window: –1 = no draw, 0 = erase, 1 = draw solid, 2 = complement, 3 = draw dashed. The default is 1 (draw solid).

`how_many` Optional real-valued expression that specifies the maximum number of objects the vision system should try to recognize in the window. (See the description of VPICTURE for a detailed explanation.)

–1	Locate as many objects as possible (the default)
0	Locate none (V.RECOGNITION effectively disabled)
1	Locate only one object
2	Locate at most two objects.
n	Locate at most n objects

NOTE: The parentheses in the instruction syntax can be omitted if all four of the above parameters are omitted.

ibr	Integer value specifying the image buffer region for the window. Image buffer regions specify both a size and a frame store (see the description of VDEF.AOI). The image buffer region's AOI must specify a rectangular shape.
shape	Optional real-valued expression indicating the shape of the window. Currently, the only choice is 1, for rectangular.
cx, cy	Real-valued expressions specifying the center coordinate of the rectangular window, in millimeters.
width	Real-valued expression specifying the width of the window, in millimeters.
height	Real-valued expression specifying the height of the window, in millimeters.
ang	Optional real-valued expression specifying the orientation of the window, in degrees. The default is 0 degrees.

Details

VWINDOW performs image processing within an area-of-interest called a window. A window is a rectangle of any size and orientation in the image. Multiple windows in the same image may be processed. Windows may overlap or even be nested inside each other. Many small windows in an image may be processed in less time than it would take to process the entire image.

The virtual camera parameter ("cam") selects the group of prototypes, system switches, and system parameters to use when processing the window. This also selects the calibration array to be accessed.

The VWINDOW instruction is essentially a repicture ("VPICTURE 0") inside a window. Almost all of the system switches and parameters apply. Thus, recognition may be performed in one window, connectivity statistics computed in another, etc.

However, some system features are applied only during the actual acquisition of an image (that is, when a VPICTURE instruction acquires an image). The following system switches and parameters are not referenced by VWINDOW: V.GAIN, V.OFFSET, V.BINARY, V.THRESHOLD, and V.2ND.THRESH.

The V.BORDER.DIST parameter is considered only with orthogonal windows. It is ignored for rotated windows.

A VPICTURE operation must be initiated before window processing so that an image is available for analysis. Any VPICTURE mode may be used, except for future frame grab (mode #1). Quick frame grab (mode #2) is the recommended mode. Either of the two frame stores may be used (see the VSELECT program instruction).

Like VRULER and VWINDOWI, VWINDOW instructions queue on the vision processor. Thus, an application program does not have to wait for picture processing to complete before executing a VWINDOW instruction.

The values of the parameters V.FIRST.COL, V.FIRST.LINE, V.LAST.COL, and V.LAST.LINE are used to clip the window to the image. That is, the vision system ignores any portion of the window that is outside these boundaries.

Type #0 rulers and finders and clear-grip tests are clipped to the bounds defined by the latest VWINDOW instruction. Only transitions within those bounds are found. Rulers and finders other than type #0, and inspection (VWINDOWI) windows are not limited by the bounds defined by the latest VWINDOW.

Example

Perform a quick frame grab and process two windows. The first window is centered at the point (140,260), 20mm x 30mm, and rotated 45 degrees. The second one is centered at (300,330), 40mm x 40mm, with no rotation. The virtual camera for the first window is 2, and the virtual camera for the second window defaults to 1.

```
VDEF.AOI 2000 = 1, 140, 260, 20, 30, 45
VDEF.AOI 3000 = 1, 300, 330, 40, 40

VPICTURE (cam) 2

VWINDOW (2) 2000
VWINDOW 3000
```

Related Keywords

VDEF.AOI (program instruction)
VPICTURE (monitor command and program instruction)
VWINDOWB (program instruction)
VWINDOWI (program instruction)

Syntax

```
VWINDOWB (cam, mode, dmode) data[index] = ibr
```

Function

Extract image information from within a rotatable, rectangular window.

Usage Considerations

All bounded regions in the image buffer region are considered one blob. The statistics returned are for all bounded regions regardless of whether they are disjoint.

The perimeter calculations for this instruction are less accurate than for VWINDOWI and are available only with the AdeptVision Enhanced VXL Interface option.

Parameters

`cam` Optional real-valued expression that specifies a virtual camera number. The system parameters V.THRESHOLD and V.2ND.THRESH and the system switch V.BACKLIGHT for this virtual camera are used during processing. The default is 1.

`mode` Optional real-valued expression that specifies what image statistics are to be computed and returned in the data array, as follows:

1 Default. Computes the area, bounding box, and centroid. (Time=17ms)
2 For EVI board option only. Computes all of the above, plus perimeter (Time=25ms)

`dmode` Optional real-valued expression that specifies the display mode to use when displaying the border of the window:

–1 No draw
0 Erase
1 Default. Draw solid
2 Complement
3 Draw dashed
4 Complement dashed

NOTE: The parentheses in the instruction syntax can be omitted if all three of the above parameters are omitted.

`data[]` Variable name specifying an array into which the image information is to be placed. The amount of information depends on the “type” parameter, as described below.

`index` Optional array index that identifies the first element to be defined in "data[ ]". The default is 0. If a multiple-dimension array is specified, only the right most index is incremented as the values are assigned.

`ibr` Optional integer value specifying the image buffer region for the inspection window. Image buffer regions specify both a size and a frame store (see the description of VDEF.AOI). A VWINDOWB tool must be an orthogonal rectangle, so the image buffer region's AOI must use a rectangular shape, and any angle specifications are ignored. Default is current frame, full size (image buffer region = 1000).

Details

This instruction quickly calculates basic window statistics. Since all regions are considered as one blob, execution time is independent of scene complexity.

When calculating perimeter, VWINDOWB counts any foreground pixel as a perimeter pixel if it has one or more background pixels as neighbors in a 3x3 neighborhood.

A VPICTURE instruction must be issued before window processing, so that an image is available for analysis. Any VPICTURE mode may be used, except future frame grab (mode #1). Quick frame grab (mode #2) is the recommended mode.

The window types are:

1 Returns the area, bounding box, and centroid of a blob.
2 Returns the area, bounding box, centroid, and perimeter of a blob (perimeter available only with the AdeptVision Enhanced VXL Interface option).

The statistics returned are for all bounded regions within the area-of-interest. The "data" array returns the following information:

data[i]	Is the area of interest clipped by the field of view (–1 = yes, 0 = no)?
data[i+1]	Area of blobs (in pixels)
data[i+2]	X component of the blob centroid (in millimeters)
data[i+3]	Y component of the blob centroid (in millimeters)
data[i+4]	X component of closest point on blob perimeter (in millimeters)
data[i+5]	X component of furthest point on blob perimeter (in millimeters)
data[i+6]	Y component of closest point on blob perimeter (in millimeters)
data[i+7]	Y component of furthest point on blob perimeter (in millimeters)
data[i+8]	Perimeter of all bounded areas (type #2 window only) (in millimeters)

Related Keywords

VDEF.AOI (program instruction)
VPICTURE (program instruction)
VWINDOWI (program instruction)

Syntax

```
VWINDOWI (cam, type, dmode, sample) data[index] = ibr

VWINDOWI (cam, type, dmode, sample) data[index] =
                                                shape, cx, cy, dx, dy, ang

VWINDOWI  (cam, type, dmode, sample) data[index] =
                                              shape, cx, cy, or, ir, ang0, angn
```

Function

Extract image information from within a window with any of the following shapes: rectangle, circle, pie cut, ring, or ring segment.

Usage Considerations

The frame store currently selected or the image buffer region specified must contain a valid picture. Otherwise, an error results.

Adept recommends the first syntax.

Parameters

`cam` Optional real-valued expression that specifies a virtual camera number. The system parameters and switches for this virtual camera are used during some modes of processing (see below for details). The default is 1.

`type` Optional real-valued expression that specifies what image statistics are to be computed and returned in the data array. The default type is 0. See the details below.

`dmode` Optional real-valued expression that specifies the display mode to use when displaying the border of the window: –1 = no draw, 0 = erase, 1 = draw solid, 2 = complement, 3 = draw dashed, 4 = complement dashed. The default is 1 (draw solid).

`sample` Optional real-valued expression that specifies the sampling density, either 1 or 2. If the value is 1, every pixel in the window is considered. If the value is 2, every other pixel on every other line is considered. The default is 1.

NOTE: The parentheses in the instruction syntax can be omitted if all four of the above parameters are omitted.

`data[]` Array into which the image information is to be placed. The amount of information depends on the "`type`" parameter, as described below.

`index`	Optional array index that identifies the first element to be defined in "**`data[]`**". The default is 0. If a multiple-dimension array is specified, only the right-most index is incremented as the values are assigned.
`ibr`	Integer value specifying the image buffer region for the inspection window. Image buffer regions specify both a size and a frame store (see the description of VDEF.AOI).
`shape`	Optional real-valued expression indicating the shape of the window: 1 = rectangular (the default), 2 = circular or ring-shaped.
`cx, cy`	Real-valued expressions specifying the (x,y) position of the center of the window, in millimeters.
`dx, dy`	Real-valued expressions specifying the width (dx) and height (dy) of a rectangular window, in millimeters.
`ang`	Optional real-valued expression specifying the orientation of the window, in degrees. The default is 0 degrees.
`or`	Real-valued expression specifying the outer radius of a circular or ring-shaped window, in millimeters.
`ir`	Optional real-valued expression specifying the inner radius of a ring-shaped window, in millimeters. The default is 0, meaning that the window is not ring-shaped (that is, it has no inner circle).
`ang0, angn`	Optional real-valued expressions specifying the angular range of a circular or ring-shaped window, in degrees. Each parameter defaults to 0. A full circle or ring is indicated when both values are 0.

Details

A VPICTURE instruction must be issued before window processing, so that an image is available for analysis. Any VPICTURE mode may be used, except future frame grab (mode #1). Quick frame grab (mode #2) is the recommended mode.

The information that VWINDOWI returns in its array parameter is determined by the "`type`" parameter. The table below lists the value each array element holds for each window type (the actual index for a data item is "`index`" **plus** the item's "relative index" shown).

.

Window Type	Relative Array Index 0	1	2	3	4	5	6	7	8	9
0	clipped?	width	ht	area	count	0	0	0	0	0
1	clipped?	width	ht	area	avg	0	0	0	0	0
2	clipped?	width	ht	area	avg	min	max	0	0	0
3	clipped?	width	ht	area	avg	min	max	object	bkgd	0
4	clipped?	width	ht	area	avg	min	max	object	bkgd	stdev
5	clipped?	width	ht	area	ecount	0	0	0	0	0
6	(128 histogram pixels counts–see below)									

where:

clipped? TRUE if the window extended past the edge of the image. FALSE otherwise.

width, ht The actual pixel width and height of the window after millimeter-to-pixel conversion and compensation for any nonsquareness of pixels.

area The actual number of pixels considered in the window. If "`sample`" is 2, this value is close to 1/4 the total number of pixels in the window.

count The number of nonzero pixels in the window of the binary frame store (as seen in VDISPLAY #2). This is only for type-0 windows. See below.

avg Average graylevel in the window.

min,max The minimum and maximum graylevel in the window.

object, bkgd These are the numbers of pixels that are part of an object or part of the background (bkgd).

stdev The standard deviation of the graylevels in the window from the mean (avg).

ecount The number of edge points in the window.

A type #6 window returns histogram data in its array parameter. The array is filled with the pixel counts for each of the possible intensity values, 0 to 127.

All the other types of windows return the boolean “clipped?”, indicating whether or not the window was completely in the image. Nonrectangular windows (circles, rings, etc.) that are clipped are not evaluated. That is, zeros are returned in the array elements for all the items other than “clipped?”. Rectangular windows that are clipped are still evaluated, but the fact that part of the window was out of the image could invalidate usage of the window information.

All windows (except type #6 windows) also return their width, height, and area in pixels. For windows orthogonal to the image boundaries, the area is simply width times height. If the “sample” parameter is 2, however, the area is approximately:

$$(\text{width} * \text{height})/4$$

You may notice that a window defined to be square in millimeters does not have equal width and height in pixels. This happens because the pixels are not square or there is some tilt to the camera. The vision system compensates for this so that the window covers a square area in the scene. For “shape” = 2 windows, the width and height are the X radius and Y radius, respectively, of the outer edge, in pixels. This reflects the millimeter-per-pixel ratio and the X/Y ratio in the calibration.

Type #0 windows differ from all the other types in that they are applied to the frame-grabbed binary image that you see in VDISPLAY mode #2. Type #0 windows simply return a count of the white pixels in the window. If the V.BINARY system switch is enabled, this is the number of background or foreground pixels, depending on the physical lighting setup. If V.BINARY is disabled, this is the number of edge pixels in the window.

All windows except type #0 are applied to the frame-grabbed grayscale image you see in VDISPLAY mode #1. Window types #1, #2, #3, and #4 return the average graylevel for the pixels in the window. Types #2, #3, and #4 also return the minimum and maximum graylevels found in the window. For types #3 and #4, you also get a count of the background and foreground pixels, as determined by the V.THRESHOLD and V.2ND.THRESH system parameters and the V.BACKLIGHT system switch for the virtual camera specified. The criterion for classifying a pixel as being background or foreground is the same as that used during a VPICTURE operation (see the descriptions of V.THRESHOLD and V.BACKLIGHT).

For type #4 windows, the standard deviation of the graylevels from their mean is returned. This is the most time-consuming value to compute. However, it provides a more reliable measure of graylevel distribution (or “edginess”) than the minimum and maximum graylevel values that are returned.

A type #5 window finds omnidirectional edge points in the grayscale image and counts them. The edge detection method used is the same as that used by VPICTURE when the V.BINARY system switch is disabled. (However, the setting of V.BINARY has no effect on this window operation.) The V.EDGE.STRENGTH system parameter is the threshold criterion used. As V.EDGE.STRENGTH is reduced, more edges are found in the window. As it is increased, fewer edges are found. The edge points in an image may be viewed in VDISPLAY mode #2 after performing a VEDGE operation of type 1 for cross gradient.

Rectangular, nonrotated windows (that is, ones with sides parallel to the sides of the image) are processed faster than circular windows or rotated, rectangular windows.

Examples

Using the system switches and parameters for virtual camera #14 during the processing, ask for all the information possible about a type #4, rectangular window centered at the point (230,340), 20 millimeters wide and 10 millimeters high, with a 15-degree rotation. Draw the window boundary in the Vision display window (dmode = 1):

```
VWINDOWI (14,4,1) xx[] = 1, 230, 340, 20, 10, 15
```

Use a type #5, ring-segment (shape-2) window (with the system parameters for virtual camera #1) to perform an inspection of a half-eaten doughnut, looking for debris (edge points). The center of the doughnut's circle is located at the point (25.2,37). The outer and inner radii are 50 millimeters and 18 millimeters, respectively. The doughnut is oriented so that its angular range is 0 to 180 degrees:

```
VWINDOWI (, 5, 1) xx[] = 2, 25.2, 37, 50, 18, 0, 180
```

Related Keywords

VDEF.AOI (program instruction)
VHISTOGRAM (monitor command and program instruction)
VRULERI (program instruction)
VWINDOW (program instruction)
V.2ND.THRESH (system parameter)
V.BACKLIGHT (system switch)
V.BINARY (system parameter)
V.THRESHOLD (system parameter)

AdeptVision Quick Reference

V.2ND.MOMENTS [camera]
Enable computation of the best-fit ellipse for each region in the image.

V.2ND.THRESH [camera]
Set a second threshold for use during binary image processing.

VABORT task_id
Abort any active vision processing and all pending vision operations associated with the given task number.

VADD (cam, type, dmode) **dest_ibr = src1_ibr, src2_ibr**
Add two binary or grayscale images.

VAUTOTHR (dmode, start, end) array[index] = ibr
Determine good thresholds for binary image processing based on the gradients in a grayscale frame store.

V.BACKLIGHT [camera]
Define which color (black or white) is to be considered the background.

V.BINARY [camera]
Enable or disable automatic edge-image generation at VPICTURE time.

V.BORDER.DIST [camera]
Define an image border reduction (in pixels) to mask out regions clipped by the image border.

V.BOUNDARIES [camera]
Enable or disable boundary analysis by the vision system.

V.CENTROID [camera]
Enable computation of the centroid of each region in the image.

VCONVOLVE (cam**, type**, dmode**)** dest_ibr = src_ibr
Perform an image convolution on a grayscale frame, possibly storing the result in a different frame store.

VCOPY (cam, scale, dmode, lut) dest_ibr = src_ibr
Copy the image from one image buffer region to another.

VCORRELATE (cam, mode, dmode, max_depth, accept, give_up) **data[**i**]**, act_depth **= tplnum**, ibr
VCORRELATE (cam, mode, dmode) data[i] = tplnum, shape, cx, cy, dx, dy, ang
Perform a normalized grayscale or binary correlation, comparing a predefined template with a rectangular image window, or searching for a closest match within the window to a predefined template.

VDEF.AOI aoi = shape, dim1, dim2, dim3, dim4, ang1, ang2
Define an area-of-interest (AOI). Areas-of-interest are used by most vision tools to specify the tool placement within an image.

VDEF.CONVOLVE type = array[i**,**j**]**
Define an image convolution.

VDEF.FONT (op) **font_num, $chars**, height, black_chars
Define, replace, or modify an Optical Character Recognition (OCR) font.

VDEFGRIP $proto, grip, mode, num_fngrs, trans[i**], w[**j**], h[**k**]**
VDEFGRIP $proto, 0
Define the shape and position of a robot gripper for clear-grip tests.

VDEF.LUT lut.num = lut[]
Define a grayscale and binary "look-up table" for mapping graylevel and binary values during a VCOPY operation.

VDEF.MORPH (mode) type = array[], dx, dy
Define a binary morphological operation.

VDEF.SUBPROTO proto:subname, first_edge, last_edge
Define a subprototype.

VDEF.TRANS (mode) dx, dy, angle, scale
Define a transformation to apply to the location of all vision tools placed until the next VDEF.TRANS instruction.

VDELETE model_name
Delete a specified prototype, subprototype, Optical Character Recognition (OCR) font, or correlation template in the vision system.

V.DISJOINT [camera]
Determine whether or not prototypes may be matched to multiple disjoint regions.

VDISPLAY (camera) **mode**, overlay, xpan, ypan, zoom
Select the current vision display mode or the display mode to be used when the vision system performs its normal image processing functions.

V.DRY.RUN
Enable graphics-only mode for various vision operators.

VEDGE (cam, type, dmode) dest_ibr = **src_ibr**
Compute edges in the grayscale image and threshold the edges, replacing the binary image, using either a cross-gradient or Sobel algorithm.

VEDGE.INFO data[i**] =** proto_nam**, edge_num**
Retrieve information about the edges and corners of a prototype or of a region in the image.

V.EDGE.INFO [camera]
Enable saving of information about edges in the image for recall via the VEDGE.INFO instruction.

V.EDGE.STRENGTH [camera]
Set the edge threshold for grayscale image processing and fine-edge rulers.

V.EDGE.TYPE [camera]
Determine the type of edge operator to use, cross-gradient or Sobel, when a VPICTURE instruction is performed.

VFEATURE (index)
Return specified information about the object most recently VLOCATEd or the prototype most recently displayed by the VSHOW program instruction.

VFIND.ARC (cam, mode, dmode, effort, type) **data[**i**] = ibr**
VFIND.ARC (cam, mode, dmode, effort, type) data[i] = 1, xc, yc, r, rr, ang0, angn
Fit a circular arc to an image edge bounded by a window that is shaped like a ring or a ring segment.

VFINDER (cam, type, dmode, how_many_total {, times[]}**)** ibr
VFIND.LINE (cam, pos, dmode, effort, type) **data[**i**] = ibr**
VFIND.LINE (cam, pos, dmode, effort, type) data[i] = 1, xc, yc, length, width, angle
Fit a straight line to an image edge within a window.

VFIND.POINT (cam, pos, dmode, effort, type) **data[**i**] = ibr**
VFIND.POINT (cam, pos, dmode, effort, type) data[i] = 1, xc, yc, length, width, angle
In a search window, find the edge point that is nearest to one side of the window.

V.FIRST.COL [camera]
Set the number of the first column of pixels to be processed.

V.FIRST.LINE [camera]
Set the number of the first line of pixels to be processed.

V.FIT.ARCS [camera]
Enable or disable the fitting of circular arcs when performing boundary analysis.

V.GAIN [camera]
Set the gain for the incoming video (camera) signal.

VGAPS data[i**]** = proto_name, edge_num
Find the unverified gaps in a match with a prototype or subprototype.

VGET.AOI array[i**] = aoi**
Return the definition of an area-of-interest.

VGETCAL (cam) **scalers[**i**]**, pmm.to.pix[j,k], pix.to.pmm[l,m], to.cam
Ask the system to fill in arrays with the previously defined vision calibration data for a given virtual camera.

VGETPIC (cam, type, s_rate, s_mode) **$pic[**r,c**]** = shape, x0, y0, dx, dy
Read all or part of an image into a string array.

VGET.TRANS array[i**]**
Return the value of the current vision transformation.

VHISTOGRAM (dmode) array[index] = ibr
Compute the histogram for a grayscale frame store.

V.HOLES [camera]
Enable or disable the accounting of interior features in all objects.

V.IO.WAIT [camera]
Enable the synchronization of taking pictures (VPICTUREs) with an external event that triggers the fast digital-input interrupt line.

VISION
Enable the entire vision system.

V.LAST.COL [camera]
Set the number of the last column of pixels to be processed.

V.LAST.LINE [camera]
Set the number of the last line of pixels to be processed.

V.LAST.VER.DIST [camera]
Enable an extra verification of prototype-to-image matches and specify the pixel tolerance to use when determining boundary coincidence.

VLOAD file_spec
Load vision models (ObjectFinder models, prototypes, Optical Character Recognition fonts, or correlation templates) from a disk file.

VLOAD (lun) $file_spec
Load vision models (ObjectFinder models, prototypes, Optical Character Recognition fonts, or correlation templates) from a disk file.

VLOCATE (camera, mode, order) $name, trans_var
Identify and locate an object in the scene.

V.MAX.AREA [camera]
Set the maximum area above which the vision system ignores regions.

V.MAX.PIXEL.VAR [camera]
During a VFIND.LINE or VFIND.ARC operation, this parameter specifies the maximum pixel distance from the fit edge beyond which edge points may be filtered out.

During boundary analysis, this parameter sets the maximum pixel deviation allowed when fitting lines and arcs to region edges.

V.MAX.SD [camera]
Set the distance (in units of standard deviation) from the fit line or arc beyond which edge points should be filtered out.

V.MAX.TIME [camera]
Set the maximum time allowed for the vision system to analyze a region during object finding, prototype recognition, or OCR. A value of 0 means that there is no time limit.

V.MAX.VER.DIST [camera]
Set the pixel tolerance for determining boundary coincidence during the verification of prototype-to-image matches.

V.MIN.AREA [camera]
Set the minimum area below which the vision system ignores regions.

V.MIN.HOLE.AREA [camera]
Set the minimum area below which the vision system ignores holes.

V.MIN.LEN [camera]
Set the minimum length of features to be used for feature pairs.

V.MIN.MAX.RADII [camera]
Enable the feature that, for each region in the image, finds the two points on the perimeter that are closest to and farthest from the region centroid.

VMORPH (cam, **type**, dmode, thresh**)** dest_ibr, count **= src_ibr**
Perform a morphological transform on a binary image frame.

VOCR (cam, op, dmode) **data[**i**]**, locs[j] **= font_num**, $expected, **ibr**
VOCR (cam, op, dmode) data[i], locs[j] = font_num, $expected, shape, cx, cy, dx, dy, ang
Perform Optical Character Recognition (OCR) or text verification in a rectangular image window.

V.OFFSET [camera]
Set the offset for the incoming video signal (that is, program the zero reference for the A/D converter).

V.OVERLAPPING [camera]
Determine whether or not objects may be overlapping in the image.

V.PERIMETER [camera]
Enable computation of the lengths of region perimeters.

VPICTURE (camera, wait, acq_ibr, sel_ibr) mode, how_many
Acquire an image into a frame store and/or initiate processing.

VPLAN.FINDER (cam, type, dmode) $fmods[], $bmods[]
Set up the type of "planning" used by the Finder when locating models.

VPUTCAL (camera) scalers[i**]**, pmm.to.pix[j,k], pix.to.pmm[l,m], to.cam
Load vision calibration data from 1 to 3 arrays.

VPUTPIC (camera, type, zoom) **$pic[**r,c**]**, x0, y0
Store into a frame store an image saved previously with VGETPIC.

VQUEUE (camera)
Display object information for any objects queued in the vision system awaiting retrieval by VLOCATE instructions.

VQUEUE (camera, $name)
Return the number of objects in the vision system queue.

V.RECOGNITION [camera]
Enable or disable prototype recognition by the vision system.

VRENAME new_name = old_name
Rename a prototype or subprototype.

VRULERI (cam, type, dmode, maxcnt, edge.dir) **data[**i**]**, mags[j] **= ibr**
VRULERI (cam, type, dmode, maxcnt, edge.dir) data[i], mags[j] **=** 1, x0, y0, len, ang
VRULERI (cam, type, dmode, maxcnt, edge.dir) data[i], mags[j] **=** shape, cx, cy, radius, ang0, angn
Obtain edge information or graylevels along a line or circular arc in the current image.

VSELECT (mode) ibr = camera
Select a virtual frame buffer for processing or display and, optionally, select a virtual camera and its calibration to be associated with the frame buffer.

VSHOW proto_name, grip, edge_num
List the defined prototypes or display a vision prototype, a subprototype, or a specific prototype edge in the Vision display window. Edge numbers are optionally shown.

VSHOW mode, **$proto_name**, trans_var, grip, edge_num
Display a vision model (ObjectFinder, prototype, or subprototype) and make information about it

available through the VFEATURE real-valued function.

V.SHOW.BOUNDS [camera]
Enable the special display of the lines and arcs fit to the boundaries of regions.

V.SHOW.EDGES [camera]
Enable the special display of edges—both the primitive edges that are fit to the boundaries of regions, and the edge points that are found by the finders VFIND.LINE, VFIND.ARC, and VFIND.POINT.

V.SHOW.FEATS [camera]
Enable the special display of features used for ObjectFinder recognition.

V.SHOW.GRIP [camera]
Enable the special display of clear-grip tests.

VSHOW.MODEL (mode) $chars, data[i] **= $model_num**
Display a model—either a correlation template or an Optical Character Recognition (OCR) font—and return information about it, or return information about all the defined templates or OCR fonts.

V.SHOW.RECOG [camera]
Enable the special display of the objects recognized.

V.SHOW.VERIFY [camera]
Enable the special display of the verification step in the recognition process.

VSTATUS
Display vision system status information in the Monitor display window.

VSTATUS (camera, type) **array[**index**]**
Return vision system status information in a real array.

VSTORE file_spec = model_name, model_name, ...
Store in a disk file selected (or all) vision prototypes (and their subprototypes), Optical Character Recognition (OCR) fonts, or correlation templates.

VSTORE (lun) $file_spec = $model_name, $model_name, ...
Store in a disk file selected (or all) vision prototypes (and their subprototypes), Optical Character Recognition (OCR) fonts, or correlation templates.

V.STROBE [camera]
Enable the firing of a strobe light in synchronization with taking pictures (VPICTUREs).

VSUBPROTO ver_percent, trans_var = subproto_name, edge_num
Determine the percentage of an edge or subprototype that was verified during recognition. Also, this instruction can have the prototype position refined, based on only a subprototype or a single edge, producing an adjusted location for the prototype.

VSUBTRACT (cam, type, dmode) dest_ibr = **src1_ibr, src2_ibr**
Subtract two binary or grayscale images.

V.SUBTRACT.HOLE [camera]
Determine whether or not hole areas are to be subtracted from region areas.

V.SYNC.STROBE [camera]
Select synchronous or asynchronous firing of a strobe light when a picture is taken (that is, when a VPICTURE is executed).

VTHRESHOLD (cam, type, dmode) dest_ibr = src_ibr
Threshold a grayscale image, producing a binary image.

V.THRESHOLD [camera]
Set the camera grayscale value that separates black pixels from white pixels.

V.TOUCHING [camera]
Determine whether or not objects may be touching in the image.

VTRAIN $prototype, shape, cx, cy, width, height, ang
VTRAIN $prototype, ibr
Initiate training of the prototype whose name is specified.

VTRAIN (cam, mode, arg) **$prototype, ibr = value**
VTRAIN cam, mode, arg) **$prototype,** shape, **cx, cy, dx, dy,** ang **= value**
Initiate training of the prototype whose name is specified.

VTRAIN.FINDER (cam, mode, dmode, arg, arg2, arg3)$model_name, ibr = value, value2, value3
Initiate training of the finder model whose name is specified.

VTRAIN.MODEL (cam, plan) **$model_name,** $text**, ibr**
VTRAIN.MODEL (cam, plan) **$model_name,** $text, shape, **cx, cy, dx, dy**, ang
Train on a vision "model"—a correlation template or an Optical Character Recognition (OCR) font. For correlation, this instruction defines the template. For OCR, this instruction trains the vision system to recognize characters in a font, or causes the vision system to plan the recognition strategy for a fully trained font.

VWAIT (type) ibr
Delay program execution until processing of a VPICTURE or VWINDOW operation is complete.

VWINDOW (cam, type, dmode, how_many) **ibr**
VWINDOW (cam, type, dmode, how_many) shape, cx, cy, width, height, ang
VWINDOWB (cam, mode, dmode) **data[**index**] =** ibr

Extract image information from within a rotatable, rectangular window.

VWINDOWI (cam, type, dmode, sample) **data[**index**] = ibr**
VWINDOWI (cam, type, dmode, sample) data[index] = shape, cx, cy, dx, dy, ang
VWINDOWI (cam, type, dmode, sample) data[index] = shape, cx, cy, or, ir, ang0, angn

Extract image information from within a window with any of the following shapes: rectangle, circle, pie cut, ring, or ring segment.

Prototype Recognition Algorithms B

Overview

When AdeptVision VXL processes an image, doing boundary analysis and recognition, the operations on the image data are performed in sequence. The camera image is frame grabbed and stored in memory. Then connectivity is performed on the binary (possibly thresholded edges). The boundaries of the regions found are then processed.

Image processing involves multiple steps, producing multiple boundary representations, the last of which is a connected sequence of lines and arcs. Finally, recognition is performed by comparing boundary features with the features of the prototype models.

More specifically, there are seven steps to image processing:

1. Connectivity analysis
2. Chain encode perimeters
3. Fit primitive edges to chains
4. Fit lines and arcs to edges
5. Classify features
6. Propose prototype-to-image matches
7. Verify match

An object is located and properly identified when all seven steps are successfully completed. At that time, the object's identification, position, 2-D orientation, and "goodness-of-appearance" measure (verified percentage) are sent to the main system processor. This information is then available to application programs written in the V^+ language.

> **NOTE:** An additional tool, called the ObjectFinder, provides an alternate method for locating objects in an image. Refer to Appendix C of the ***AdeptVision User's Guide*** for information on the ObjectFinder tool.

Connectivity Analysis

This is the common technique developed at SRI International for separating closed regions from the background. The processing is performed in a single pass across the image. The binary image is scanned and converted into run-length encodings on a line-by-line basis. This compact image representation is simply a

list of the column numbers where there are “color” transitions of black-to-white or white-to-black. Connectivity analysis groups contiguous horizontal pixels of the same color into run-lengths and then groups contiguous vertical run-lengths of the same color, thereby completely segmenting the image into black and white regions. While processing the run-length encodings, the algorithm also computes the area and perimeter of each region.

Finally, as regions close off, they are interrelated via parent, child, and sibling relationships. A hole in a region is a child of the outer region and the outer region is a parent of the inner region. Two holes in a region are mutual siblings.

The vision system completes connectivity analysis before proceeding on to the other processing steps.

Chain Encode Perimeters

Processing proceeds with this step after the outermost region, the region whose parent is the background, closes off. This step produces a chain encoding representation of the region perimeter. Using the stored run-length encodings with their region associations provided by connectivity analysis in step 1, a string of right, up, left, and down moves are produced that will traverse the region perimeter.

Fit Primitive Edges to Chains

This step fits straight-line segments to the chain encodings in order to produce a more succinct representation of the region perimeter. The algorithm used is efficient, and the edges produced are an accurate approximation of the chain encoding. In this manual, these are called the primitive edges. A benefit of the algorithm is that the jagged (quantized) edges represented by the chain encoding are smoothed. After this step, the chain-encoding data are discarded.

Fit Lines and Arcs to Edges

This processing step fits straight-line and circular-arc segments to the edges produced in step 3. This step is relatively expensive in processor time, but the expense is well justified. First of all, lines and arcs are fit to within a user-specified tolerance (V.MAX.PIXEL.VAR), so image smoothing is controlled. Secondly, the

result is further data reduction, especially when the parts are circular or have circular holes. Studies of industrial parts have shown that the majority of parts are prismatic and rotational in shape, so their 2-D images are naturally represented by lines and arcs. The final justification for fitting lines and arcs (as opposed to fitting only lines) is that they provide a greater variety of edge and corner types, making recognition easier.

Classify Features

This step classifies all connected edge-edge pairs, where the edges are the lines and arcs from step 4, and associates the pairs with the feature classes of the known prototypes. The feature classes are chosen during training, based on the objects the user wants the vision system to recognize. Generically, there are only a few types of corner classes: line-line, line-arc, arc-line, arc-arc, and circle. In addition, there are the "wildcard" variations: line-any, any-line, arc-any, and any-arc. The "any" refers to an edge that may belong to a different object because two parts touch or overlap.

Although there are only a few generic types of feature classes, each of the feature classes of a prototype have specific parameterized definitions. An acceptable minimum and maximum included angle (the angle formed by the two edges) is associated with each corner class. Also, acceptable minimum and maximum lengths are associated with each line edge of a feature class. For arcs, there are acceptable minimum and maximum angular ranges, minimum and maximum radii, and a convex-or-concave indication.

Each corner and edge in the image is compared with all the feature classes. When the types correspond and the measures of the corner or edge fall within the acceptable ranges of the class, the corner or edge is placed on the association list for the feature class. Some corners and edges in the image may not be associated with any classes, whereas others may be associated with multiple classes.

Propose Prototype-to-image Matches

This step proposes that prototypes are present in the image at specific positions and orientations. These prototype-to-image match proposals are based on the feature classifications made in the previous step. During training, the feature classes are ordered according to uniqueness and "clarity". At run-time, in this

order, each class is considered until all image edges have been accounted for or until all possible matches have been proposed. Each image feature and prototype feature associated with a class is a possible match. However, such matches must be confirmed before an official proposal can be made and step 7 begins.

Confirmation is a partial verification (see step 7 below) that is required when more than one prototype feature is associated with a feature class. Even if there is only one feature associated with a class, confirmation will be used if the prototype is complex, having many edges, so that mismatches will be detected before the expensive verification step. The need for confirmation is determined during training when feature classes are chosen. Associated with each prototype feature is a list of the confirming features that, if present in the image at the same relative position, would distinguish it from other features.

Nearby features are chosen as confirming features when appropriate because they are more likely to be visible (not occluded) along with the feature to be confirmed.

Verify Match

Given a single prototype-to-image match proposal, this step verifies the presence of the prototype in the image. The prototype boundary representation is transformed to the image via the 2-D translation and rotation proposed in step 6, and a search is made for image edges that align with the prototype edges. The close-enough test is to within a user-specified tolerance (V.MAX.VER.DIST). Although the signs of the edges (black-to-white versus white-to-black) must correspond, the edge types do not. So lines verify arcs and vice versa. If enough of the prototype boundary is present in the image, the match is accepted. “Enough” is defined by a weighted threshold, based on the edge weights and the verify percentage provided during prototype training.

C Perspective Distortion

Overview of Perspective Distortion

The camera calibration programs supplied by Adept relate the pixels (picture units) of the vision system to millimeters in your workspace. These programs produce two types of calibration data: “scale transformations” and “perspective transformations”. The scale transformation is simple and fast. The perspective transformation is more complex, and a little slower, but more accurate. The vision system can use both the scale and perspective transformations, or it can use only the scale transformation.

The scale transformation consists primarily of two scalers for converting pixels to millimeters. Since the height and width of a pixel are not necessarily the same, the X and Y components of coordinates are scaled differently. The two scalers are “X scale” and “Y scale” in terms of millimeters per pixel. If (X,Y) is a coordinate in pixel units, the corresponding coordinate in millimeters is (X∗X.scale, Y∗Y.scale).

The values for “X.scale” and “Y.scale” are stored in the array specified as a parameter for the VPUTCAL program instruction.

The perspective transformation is a two-dimensional homogeneous transformation that contains the scaling described above, plus correction for perspective distortion. Furthermore, the transformation may contain a small translation and rotation. (Note that the scale transformation is a simplification of the perspective transformation.)

The Adept camera calibration programs (including the built-in VisionWare calibration) perform calibrations that correct for perspective distortion. After a calibration, these programs tell the user how much perspective distortion is present in the image, and the direction the image plane is tilted, so that the user can adjust the camera orientation or the workspace to compensate for the distortion. The user can elect to disable correction for perspective distortion in order to maximize run-time speed. If enabled, perspective calibration is used to correct the results of linear and arc rulers, and line, point, and arc finders. All other vision tools return results based on the scale transformation.

The two V+ instructions that deal with vision calibration are VGETCAL and VPUTCAL. Normally, you will never need to use these instructions directly. The calibration programs supplied by Adept pass calibration data to the vision system with the VPUTCAL instruction. The “load.area” subroutine may be included in your application program to (1) read calibration data previously saved in a disk file and (2) perform the VPUTCAL operation.[1]

[1] The “load.area” subroutine for loading calibration data can be found in the file “LOADAREA.V2” on Adept Utility Disk #2. See the ***AdeptVision User's Guide*** for information on use of this subroutine.

Calibration Arrays

The instructions VGETCAL and VPUTCAL have three real-valued array parameters: “scalers[ ]”, “pmm.to.pix[,]”, and “pix.to.pmm[,]”.

The one-dimensional array “scalers[ ]” contains the X-scale and Y-scale information described earlier. By itself, this array is sufficient for the scale transformation.

The arrays “pmm.to.pix[,]” and “pix.to.pmm[,]” are optional 3-by-3 arrays. These arrays contain two-dimensional homogeneous transformations that contain millimeters / pixel scaling, correction for perspective distortion, and a small translation and rotation. The array “pmm.to.pix[,]” transforms millimeter coordinates on a single plane in the workspace into pixel coordinates in the image. The array “pix.to.pmm[,]” does the reverse transformation. These transformations have the form:

A	B	C
D	E	F
G	H	1

Elements “A”, “B”, “D”, and “E” encapsulate scaling and rotation. Elements “C” and “F” are the translations in X and Y, respectively. Elements “G” and “H” correct for perspective distortion—the smaller these numbers, the less perspective distortion is present.

Using the 3-by-3 transformations instead of just the scale transformation takes more processing time, so their use is optional. The vision system uses only the scale transformation if the “G” and “H” elements in the “pmm.to.pix[,]” transformation are zero.

When shown a square calibration target, the Adept calibration program always computes the perspective transformation and then derives the scale transformation from it. The small translation component in the perspective transformation aligns the perspective and scale transformations in the center of the screen. The small rotation component aligns the 45-degree lines in millimeter space and pixel space.

Based on the “pmm.to.pix[,]” and “pix.to.pmm[,]” transformations, the LOADAREA program derives two additional transformations: “pmm.to.mm[,]” and “mm.to.pmm[,]”. These are provided to allow V+ application programs to correct for perspective distortion when the vision system does not do so.

In summary, the four transformations are:

pmm.to.pix[,] Perspective millimeters to raw pixels. The vision system uses this transformation to convert user-specified millimeter coordinates (such as the start of a ruler) to pixels.

pix.to.pmm[,] This is the inverse of "pmm.to.pix[,]". The vision system applies this transformation when returning results to the application program in millimeter coordinates (for example, the locations of edge transitions found by a ruler).

pmm.to.mm[,] Perspective millimeters to "simple" millimeters. This transformation corrects only for perspective distortion. The user can use this to precisely place tools in cases where the vision system does not apply the transformation "pmm.to.pix[,]" (such as VWINDOW and graphics instructions).

mm.to.pmm[,] This is the inverse of "pmm.to.mm[,]". This transformation can be used to correct for perspective distortion in vision results when the vision system does not transform the coordinates with the transformation "pix.to.pmm[,]" (such as with blob or prototype locations).

Figure C-1 shows the three spaces or coordinate systems that have been discussed above and how the various transformations can be used to convert from one coordinate system to another.

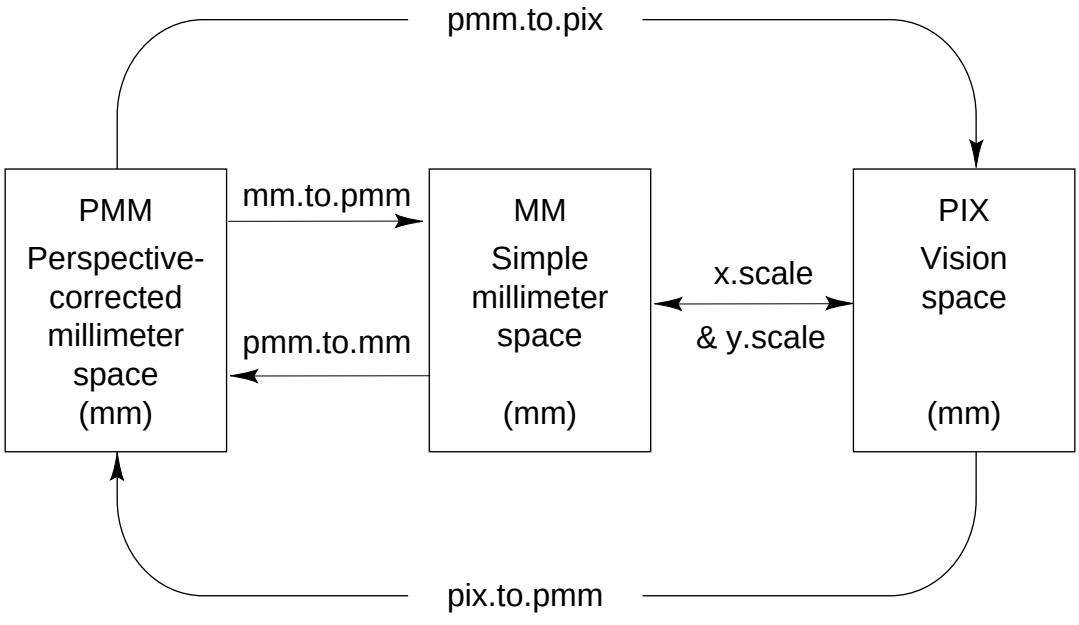


Figure C-1. Vision Coordinate Systems

The vision system uses "pmm.to.pix[,]" to place rulers (both linear and circular) and the centers of arc finders. All the other tools (including line and point finders) are placed using only the scale transformation. The results of rulers (line and arc) and finders (line, point, and arc) are transformed to millimeters using the "pix.to.pmm[,]" transformation. The results of all other vision tools are transformed using the simpler scale transformation.

If correction for perspective distortion is **disabled**, the "pmm.to.pix[,]" and "pix.to.pmm[,]" transformations that VGETCAL returns represent the following simple scale transformations:

pmm.to.pix[,] =	1/X.scale	0	0
	0	1/Y.scale	0
	0	0	1

pix.to.pmm[,] =	X.scale	0	0
	0	Y.scale	0
	0	0	1

(Both of the arrays "pmm.to.mm[,]" and "mm.to.pmm[,]" contain identity transformations when correction for perspective distortion is **disabled**.)

How to Use the Arrays "pmm.to.mm[,]" and "mm.to.pmm[,]"

How do you use "pmm.to.mm[,]" and "mm.to.pmm[,]" to correct for perspective distortion when the vision system does not automatically do it for you?

The V+ routine "tr.point" (see below) transforms a point using the transformation array provided. Your application program can use this routine as follows:

- Lines are transformed by simply transforming the two endpoints of the line
- Circles transform to an egg shape under perspective projection

> **NOTE:** The effect can be approximated by transforming the center of the circle. A radius for the transformed circle can then be computed by transforming some points on the perimeter of the circle and computing their average distance to the transformed center point.

```
.PROGRAM tr.point(trans[,], pt[], tpt[])

; ABSTRACT: Transform a 2-D point given a 3x3 homogeneous
;   transformation.
;
; INPUT PARAM: trans[,] 3x3 real array containing a 2-D homogeneous
;                        transform with pixel-to-mm scaling and
;                        compensation for perspective .
;              pt[]     3-element real array containing the X,Y,Z
;                        coordinates of the point to be transformed
;                        (the Z value should be 1 unless you are
;                        scaling)
;
; OUTPUT PARAM: tpt[]   3-element real array containing the X,Y,Z
;                        coordinates of the transformed point
;                        (normalized so that the Z value is 1)
;
; Copyright (c) 1992 by Adept Technology, Inc.

    AUTO row, col, sum

    FOR row

 0 TO 2                          ;Compute transformed point
         sum = 0
         FOR col = 0 TO 2
              sum = sum+trans[row,col]*pt[col]
         END
         tpt[row] = sum

    END

    tpt[0] = tpt[0]/tpt[2]       ;Normalize the result
    tpt[1] = tpt[1]/tpt[2]
    tpt[2] = 1
.END
```

As an example of how to use the routine “tr.point”, the following instructions apply perspective correction to the centroid of a blob. (They would be executed after the blob has been successfully VLOCATEd.)

```
pt[0] = VFEATURE(2) ;X coordinate in "simple mm space"
pt[1] = VFEATURE(3) ;Y      "      "     "    "   "
pt[2] = 1
CALL tr.point(mm.to.pmm[,], pt[], exact_pt[])
```

After these instructions are executed, the elements “exact_pt[0]” and “exact_pt[1]” contain the X,Y location of the object centroid with correction for perspective distortion.

Upgrading Program Code D

Introduction

Program code written for systems prior to version 11.0 needs to be upgraded. All version 11.0- and later-compatible program code will continue to work with version 12.1 systems.

The following section provides a compatability summary for users that need to upgrade pre-11.0 program code. This is not a summary of all changes that occurred prior to version 11.0. It is a summary of operations that have changed and may require modifications to upgrade pre-11.0 program code so it will execute correctly in version 11.0 and later systems.

Compatibility Summary

Maximum value for V.LAST.LINE is now 480 instead of 484.

The system parameters V.FIRST.COL, V.LAST.COL, V.FIRST.LINE, and V.LAST.LINE are no longer used as a clipping window for VOCR.

V.FIRST.COL, etc., no longer define the AOI for the following instructions:

VADD	VAUTOTHR	VCOPY
VCONVOLVE	VEDGE	VHISTOGRAM
VMORPH	VSUBTRACT	VTHRESHOLD

The old frame numbers 1 and 2 will no longer be accepted. 1nnn (e.g., 1001 and 1002) must be used.

When a frame number of 1nnn (e.g., 1001) is used, the entire frame is processed as before.

The default value of V.OFFSET is set to 255 instead of 127 when the system is loaded.

The maximum physical camera number is 4 instead of 8.

5x5 (and larger) convolutions can not be applied on full frame (640x480 image). They are limited to 508-pixel-wide AOIs.

Correlation templates and the AOIs of image processing operations are reduced if necessary to become a multiple of 4 pixels in width. If you need to know the exact size of a template, use VSHOW.MODEL.

V.SHOW.MODEL now shows templates in the middle of the vision window.

VDEF.CONVOLVE must be given a 7x7 array instead of a 5x5 array.

VPUTPIC strips the binary out of averaged images. Type #0 stores both grayscale and binary images.

Instructions with added arguments that make them incompatible are:

VADD	VCONVOLVE	VEDGE
VMORPH	VSUBTRACT	VTHRESHOLD

These incompatibilities are caused by standardizing on the “cam-type-dmode” argument triples.

VFIND.LINE, VFIND.ARC, and VFIND.POINT have a new “shape” argument for consistency with the other instructions.

Full-frame and field-only image acquires do not mix well. If you plan to do both with the same vision system, you experience some delay when switching back and forth. The reason for this is that full-frame is interlaced and field-only is noninterlaced, yet all cameras connected to the MUX are being driven by the same synch signals.

VWINDOW, VOCR, VTRAIN.MODEL (on a font), and VTRAIN automatically do VSELECT of the specified frame store. This should pose no backward-compatibility problems, but it was not done in previous versions.

VMORPH, VEDGE, VTHRESHOLD, VCONVOLVE, VCOPY, VADD, and VSUB operations do not need to wait for the vision processor to be idle. This should pose no compatibility problems and may allow some applications to run faster.

AdeptVision VXL has a third frame store for live video. Therefore, displaying live video will not erase a physical frame store. This should pose no compatibility problems and may provide a useful addition to some applications.

Convolutions larger than 3x3 require the AdeptVision Enhanced VXL Interface option.

The OCR capability is no longer optional: It is included with the basic system.

Index

Numerics

A

B

C

D

E

F

G

H

T

U

V

W

Adept User's Manual Comment Form

We have provided this form to allow you to make comments about this manual, to point out any mistakes you may find, or to offer suggestions about information you want to see added to the manual. We review and revise user's manuals on a regular basis, and any comments or feedback you send us will be given serious consideration. Thank you for your input.

NAME_____ DATE_____

COMPANY_____

ADDRESS _____

PHONE _____

MANUAL TITLE: *AdeptVision Reference Guide, Rev A*
PART NUMBER: 00962-01300 PUBLICATION DATE: August 1997

COMMENTS _____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

MAIL TO: Adept Technology, Inc.
Technical Publications Dept.
11133 Kenwood Rd.
Cincinnati, OH 45242