REFERENCE MANUAL

|

---A-L-I-G-N---

|

Version 2.0 for Windows NT ™

April, 1998

MNEMONICS, INC.
102 Gaither Drive
Mt. Laurel, NJ 08054
Phone: 609-234-0970
Fax: 609-234-6973
E-mail: mnemonics@compuserve.com

PART I: GENERAL INFORMATION

Description

ALIGN is a high-resolution, high-speed PC-based alignment system. This powerful general-purpose tool may be applied to a wide range of industrial inspection, pattern recognition, and alignment tasks. The ALIGN package consists of an interactive version of the Alignment System providing complete functionality for convenient development of strategies and solutions.

A complete set of alignment functions is provided as a Dynamic Linked Library (DLL). These functions may be called from any programming environment that supports calling Windows NT DLLs. This library facilitates fast and efficient development of custom alignment programming.

Patterns

ALIGN is a multi-pass pattern-directed alignment system employing full gray-scale normalized correlation techniques. Patterns are rectangular in shape, and range in size from 16 to 256 pixels for both the x and y dimensions. Patterns are specified by the coordinates of the upper left corner and by the x and y spans. Patterns may be located anywhere within the video image. Selection of the best pattern size and shape is image and task dependent. An AutoTeach capability is provided as an aid in selecting and teaching patterns for alignment.

The Alignment System can maintain up to 16 patterns in the internal Pattern Data Base. An indefinite number of patterns may be stored on disk. Patterns may be stored on disk and retrieved either as Single Pattern files or as complete Pattern Data Base files of 16 patterns.

Selecting a Pattern

Selection of a good pattern is paramount to providing fast and accurate alignment performance. When a pattern is taught, three measurements are reported to the user: Teach Score, Teach Search Depth, and Next-Best Quality. These three values are based on a series of analyses and alignments run on the teach image using the combination of the selected pattern and the selected Alignment Window.

Teach Score is a measure of the gray level distribution of pixels within the pattern modified by the image content of the surround in conjunction with the positional information of the pattern. Teach Scores range from 0 to 100 and indicate the suitability of the pattern gray level distribution for successful alignment. Higher scores indicate better suitability. Teach Scores of 0 and 1 are artificial scores used to flag particularly poor patterns. A Teach Score of 0 is used to indicate a pattern that will not align on the teach image with the selected Alignment Window. A Teach Score of 1

indicates a potentially problematic pattern that may provide poor performance in future alignment scenarios. Low Teach Scores that are greater than 1 may provide perfectly adequate alignment performance, especially when coupled with a low Teach Search Depth and/or a low Next-Best Quality. Larger patterns and smaller Alignment Windows will typically provide patterns with higher Teach Scores. The Teach Score is not related to normalized correlation quality values.

Search Depth is a measure of the ease with which the alignment process is able to find a specified pattern within a particular Alignment Window. The Teach Search Depth indicates the Search Depth that is required for successful alignment of the selected pattern within the specified Alignment Window on the teach image. Search Depth values range from 1 to 16. Low Teach Search Depths indicate that the pattern is relatively easy to find within its associated Alignment Window, and high Teach Search Depths indicate that the pattern is relatively difficult to find. Patterns with lower Teach Search Depths generally produce better alignment performance and faster alignment times. Larger patterns and smaller Alignment Windows will generally provide patterns with lower Teach Search Depths.

Next-Best Quality is a measure of the uniqueness of the specified pattern within the Alignment Window. This measurement is generated by performing a series of alignments within the Alignment Window on the teach image while avoiding the area of the pattern itself. The Next-Best Quality is the highest alignment quality value reported during this series of alignments. A high Next-Best Quality indicates that there is something else in the Alignment Window that is similar to the selected pattern in the gray-scale normalized correlation sense. A low Next-Best Quality indicates that the selected pattern is relatively unique in the correlation sense within the teach image Alignment Window. Next-Best Quality ranges in value from -100 to +100. The teach pattern itself has a quality value of 100 when an alignment is performed on the teach image. A Next-Best Quality value of 100 would indicate that there is another area within the Alignment Window that is exactly the same as the selected pattern in the correlation sense. Larger patterns and smaller Alignment Windows will generally provide patterns with lower Next-Best Quality values.

In general, the best patterns will be those which have a high Teach Score combined with a low Teach Search Depth and low Next-Best Quality. In certain cases, the user may be faced with selection of a single pattern from a series of patterns that force tradeoffs between Teach Score, Teach Search Depth, and Next-Best Quality. This may occur, for instance, in the list of patterns provided by the AutoTeach function. In these cases it is not always obvious from an examination of the teach values which patterns represent the better candidates for alignment. The selection process may be aided by performing actual alignments on images that represent true align-time imagery using each of the patterns and comparing results with respect to alignment accuracy and alignment time.

AutoTeaching

The AutoTeaching process provides automatic selection and teaching of patterns by sampling a large number of patterns from the portion of the video image within the specified AutoTeach Window. The user specifies the size, shape, and number of patterns to be taught. Each sampled pattern is aligned over the specified Alignment Window and a Teach Score, Teach Search Depth, and Next-Best Quality value are generated for that pattern. Sampled patterns are then sorted based on Teach Score first and Teach Search Depth second. Selected patterns are placed in the Pattern Data Base for subsequent use in pattern alignment.

Operator Recognition Points

An Operator Recognition Point (ORP) is an optional single-point fiducial that is tightly coupled to a particular pattern at a specified horizontal and vertical offset. The ORP is useful for visual confirmation of Alignment System performance. The Alignment System supports one ORP for each pattern in the Pattern Data Base. The Operator Recognition Point should be taught immediately after the teaching of its associated pattern.

The Alignment Process

The ALIGN alignment process is a highly optimized search strategy that provides true gray-scale normalized correlation pattern matching in very fast execution times.

The first pass of the alignment process involves a cursory high-speed low-resolution qualitative analysis of the image in order to locate sites that have a high potential for good pattern alignment. The number of these potential sites is proportional to the square of the Search Depth specified for alignment. A Search Depth of 16 would indicate that up to 256 sites may be selected by the first pass and saved for additional, more detailed analysis by the second or third pass.

The second pass of the alignment process performs a more detailed, refined analysis of each site on a site-by-site basis to determine the statistical likelihood of the site being a good candidate for a high quality pattern match. If this measure of statistical likelihood is high enough, the site will be subject to immediate third pass processing, otherwise the site will be saved for possible third pass processing at a later time.

The third pass of the alignment process examines the neighborhood of the site in ultimate detail. The gray-scale normalized correlation quality value of the best pattern match is determined and the contrast ratio of the alignment is computed. If the quality value of the pattern match for the site exceeds the Early-Stop Threshold then the alignment process is terminated and the location, quality, and contrast ratio for the site are reported to the user with an indication that the Early-Stop condition occurred. If the quality value for the site does not exceed the Early-Stop Threshold, then the third pass

processing continues on a site-by-site basis until either the Early-Stop Threshold is exceeded or all sites have been processed. If third pass processing is completed on all sites, and the Early-Stop Threshold has not been exceeded, then the site having the best quality value is reported to the user with an indication that the alignment process did not Early-Stop.

Selecting Search Depth for Alignment

The Search Depth specifies the number of potential sites to be selected by the first pass of the alignment process. The maximum number of potential sites is equivalent to the square of the Search Depth, though the actual number of sites available may be less than that for any given pattern/Alignment Window combination.

The Teach Search Depth, reported when the pattern was taught, is an indication of the Search Depth required to align the pattern within the Alignment Window on the teach image. In general, if the Alignment Window used for alignment images is the same size as the Alignment Window used in teaching the pattern, the minimum Search Depth required for successful alignment will be equal to the Teach Search Depth. However, if the Alignment Window specified for alignment images is larger than the Alignment Window specified for the teach image, or if the alignment imagery is substantially different then the teach imagery, a Search Depth larger than the Teach Search Depth may be required.

Since the Search Depth dictates the number of potential sites analyzed by the alignment process, a larger Search Depth will generally result in a longer alignment execution time. This effect will be particularly noticeable if the Early-Stop Threshold is not exceeded during the alignment process, since all the sites will be analyzed in that case. If minimum alignment time is a major criteria, the smallest Search Depth that results in accurate, dependable alignment should be selected. If minimizing the chance of missing a pattern match is the most important factor, then a Search Depth of 16 should be selected.

As a rule-of-thumb, add 3 to the Teach Search Depth to provide a reasonable starting point for determining the run Search Depth. That is: Alignment Search Depth = Teach Search Depth + 3, but not to exceed 16.

Selecting Early-Stop Threshold for Alignment

The Early-Stop Threshold specifies the normalized correlation quality value that is used to control early termination of the alignment process. If the Early-Stop Threshold is set too high, the third pass of the alignment process will be unable to find a pattern match which exceeds the threshold and the execution speed advantage of early termination will be lost. If the Early-Stop Threshold is set too low, the alignment process may find a pattern match at an incorrect site whose quality value qualifies it for early termination of the alignment process. In this case, the alignment process will appear to produce an

incorrect result, even though a true occurrence of the pattern is contained in the image and would be located accurately with the proper selection of Early-Stop Threshold.

In general, the Early-Stop Threshold should be set to a value greater than the pattern's Next-Best Quality value at teach time and less than 100. The threshold should be adjusted high enough to prevent selection of a pattern match which is not a true occurrence of the pattern and low enough to cause early termination by a true occurrence of the pattern when present.

As a rule-of-thumb, an Early-Stop Threshold value halfway between the Next-Best Quality and 100 provides a reasonable starting point for determining the run time Early-Stop Threshold. That is - Early-Stop Threshold = (Next-Best Quality + 100) / 2.

Alignment Results - Location, Quality, and Contrast Ratio

The alignment process reports four values - horizontal and vertical location of the pattern, quality and contrast ratio. Quality is a measure of the normalized correlation between the pattern at teach time and the occurrence of the pattern in the alignment image. Quality values range from -100 to +100. A quality value of 100 indicates a match with an exact copy of the teach pattern. A quality value of 0 indicates a match with an area of the alignment image that is totally uncorrelated with the pattern. A quality value of -100 indicates a match with a perfect negative of the teach pattern. Contrast ratio is a measure of the ratio of the standard deviation of the gray values in the alignment image pattern and the standard deviation of the gray values in the teach pattern. Contrast ratio indicates changes in brightness and contrast between the teach image and the alignment image to which the normalized quality value may prove insensitive. Contrast ratios greater than 1 indicate an alignment image with more contrast than the teach image. Contrast ratios of less than 1 indicate an alignment image with less contrast than the teach image.

Fast Alignment

Fast Alignment is a technique that provides considerably faster alignment execution times in certain instances. This tool may be applied to any alignment application in which different patterns are to be aligned on the same image. The image is preprocessed once using the library function ali_setup_fast_align. Subsequent alignments on the pre-processed image are relieved from the pre-processing overhead, resulting in faster alignment times. A Fast Alignment will generally execute twice as fast as a comparable normal alignment. Use the function ali_fast_align to perform Fast Alignment.

MultiAlignment

MultiAlignment refers to the ability to align on more than one occurrence of the pattern in the Alignment Window. All alignment library functions support MultiAlignment.

Implementation is through two parameters which specify the number of alignments requested by the calling function and the number of alignments actually found by the alignment system. In the case of MultiAlignment the pointers to x location, y location, quality, and contrast ratio actually point to arrays. The number of valid elements in these arrays is equal to the number of alignments found by the alignment system. It is the responsibility of the calling function to provide arrays large enough to provide storage for the number of alignments requested.

Subpixel Alignment

Subpixel alignment provides the capability to align a pattern to a fractional pixel location. Subpixel alignment resolution is operator selectable and may range from 1/2 pixel to 1/32 pixel. The Alignment Window for subpixel alignment is fixed to a rectangular area plus-or-minus 1 pixel from a specified location. This location would typically be defined by a prior whole-pixel alignment.

Software Protection

The ALIGN library is protected by the use of a Sentinel software protection device. This device must be installed on the computer's first parallel port, LPT1. If any ALIGN library function is called without this device installed, the function will not execute and an error code will be returned.

Trademarks

The following trademarks are used in this manual:

- Microsoft, Windows NT, Visual Basic, and Visual C / C++ are registered trademarks of Microsoft Corporation.
- IBM is a registered trademark of International Business Machines.
- MV-1000 is a registered trademark of MuTech Corporation.
- Rainbow Sentinel is a registered trademark of Rainbow Technologies Inc.

PART II: ALIGNMENT SYSTEM GEOMETRY

The Alignment Image

The alignment image is the portion of the vision system frame buffer that is displayed on the video monitor. This is the image that is scanned by a standard 60Hz RS-170 or 50Hz CCIR (European) camera. This image is digitized by the vision system and is then displayed from the vision system frame buffer after the image acquire operation.

Alignment Image Coordinate System

The coordinate system origin of the alignment image is located at the upper left corner of the image. X coordinates increase from left to right and Y coordinates increase from top to bottom.

The origin of the alignment image is located at frame buffer coordinates (0, 0). The image size and coordinate range is dependent on the size of the digitizer as follows:

For a 512 pixel RS-170 digitizer -
- The size of the alignment image is 512 x 480.
- The range of the X coordinates is 0 to 511.
- The range of the Y coordinates is 0 to 479.

For a 640 pixel RS-170 digitizer -
- The size of the alignment image is 640 x 480.
- The range of the X coordinates is 0 to 639.
- The range of the Y coordinates is 0 to 479.

For a 512 pixel CCIR digitizer -
- The size of the alignment image is 512 x 512.
- The range of the X coordinates is 0 to 511.
- The range of the Y coordinates is 0 to 511.

For a 768 pixel CCIR digitizer -
- The size of the alignment image is 768 x 512.
- The range of the X coordinates is 0 to 767.
- The range of the Y coordinates is 0 to 511.

Figure 1 shows an alignment image for a 512 pixel RS-170 digitizer.
Figure 2 shows an alignment image for a 640 pixel RS-170 digitizer.
Figure 3 shows an alignment image for a 512 pixel CCIR digitizer.
Figure 4 shows an alignment image for a 768 pixel CCIR digitizer.

Alignment System Windows

The alignment system uses three different rectangular windows to specify portions of the alignment image for processing - the Pattern Window, Alignment Window, and AutoTeach Window. These windows are specified by two attributes - location and size.

The window location refers to the (x,y) coordinates, in pixels, of the upper left corner of the window. The window size refers to the horizontal span of the window in x pixels and the vertical span of the window in y pixels.

Pattern Window

The Pattern Window is that portion of the alignment image that is used to specify a pattern to be stored in the Pattern Data Base and subsequently used for alignment. The Pattern Window ranges in size from 16 x 16 pixels up to 256 x 256 pixels.

A Pattern Window must be specified for teaching, and must lie completely within the Alignment Window specified for teaching.

A Pattern Window size must be specified for AutoTeaching. The specified size must be equal to or smaller than the AutoTeach Window specified for AutoTeaching.

Alignment Window

The Alignment Window is that portion of the alignment image that is searched for the best pattern match. The Alignment Window ranges in size from 16 x 16 pixels up to 512 x 480 pixels.

An Alignment Window must be specified for teaching, AutoTeaching, and alignment. The Alignment Window must lie completely within the alignment image.

AutoTeach Window

The AutoTeach Window is that portion of the alignment image that is sampled for good alignment patterns. The AutoTeach Window ranges in size from 64 x 64 pixels up to 512 x 480 pixels.

An AutoTeach Window must be specified for AutoTeaching. The AutoTeach Window must lie completely within the Alignment Window specified for AutoTeaching.

Window Relationships for Teaching

The teach process stores that portion of the alignment image specified by the Pattern Window in the Pattern Data Base for use as a pattern for subsequent alignment. The pattern Teach Score, Teach Search Depth, and Next-Best Quality are generated by aligning the pattern over the specified Alignment Window on the teach image.

Figure 5 shows the relationship between the Pattern Window and the Alignment Window for the teaching process. Note the following:

- The Alignment Window must lie completely within the alignment image.
- The Pattern Window must lie completely within the Alignment Window.

Window Relationships for Alignment

The alignment process searches that portion of the image within the Alignment Window for the best match to the specified pattern.

Figure 6 shows the relationship between the Alignment Window and the alignment image for the alignment process. Note the following:

- The Alignment Window must lie completely within the alignment image.
- The Alignment Window must completely encompass the occurrence of the pattern match upon which the alignment process is operating.

Window Relationships for AutoTeaching

The AutoTeach process samples a large number of patterns from within the AutoTeach Window. A Teach Score, Teach Search Depth, and Next-Best Quality value is generated for each pattern by aligning the pattern over the portion of the image within the Alignment Window. The highest Teach Scores are used to select the patterns from the AutoTeach Window that will be stored in the Pattern Data Base.

Figure 7 shows the relationship between the Alignment Window, AutoTeach Window and Pattern Window for the AutoTeach process. Note the following:

- The Alignment Window must lie completely within the alignment image.
- The AutoTeach Window must lie completely within the Alignment Window.
- The size of the Pattern Window must be equal to or smaller than the size of the AutoTeach Window.

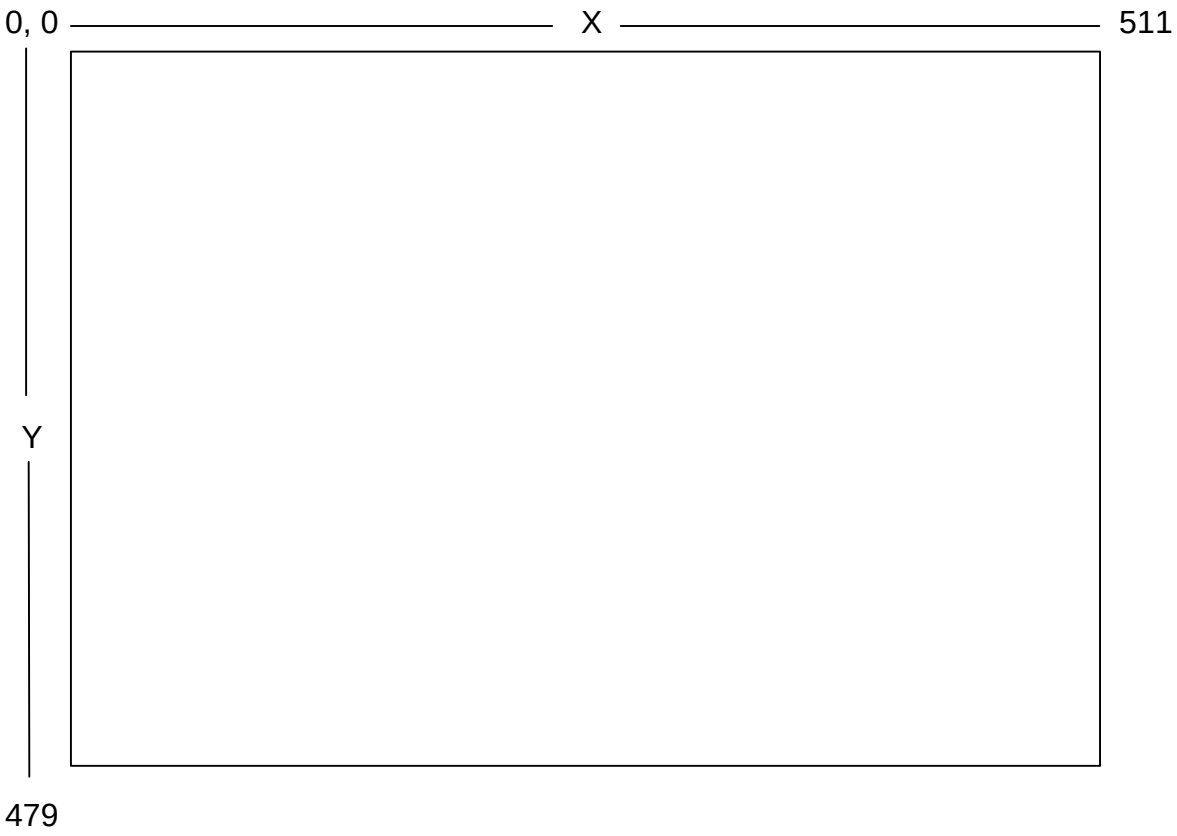

Figure 1

Alignment Image for RS-170,
512 Pixel Digitizer.

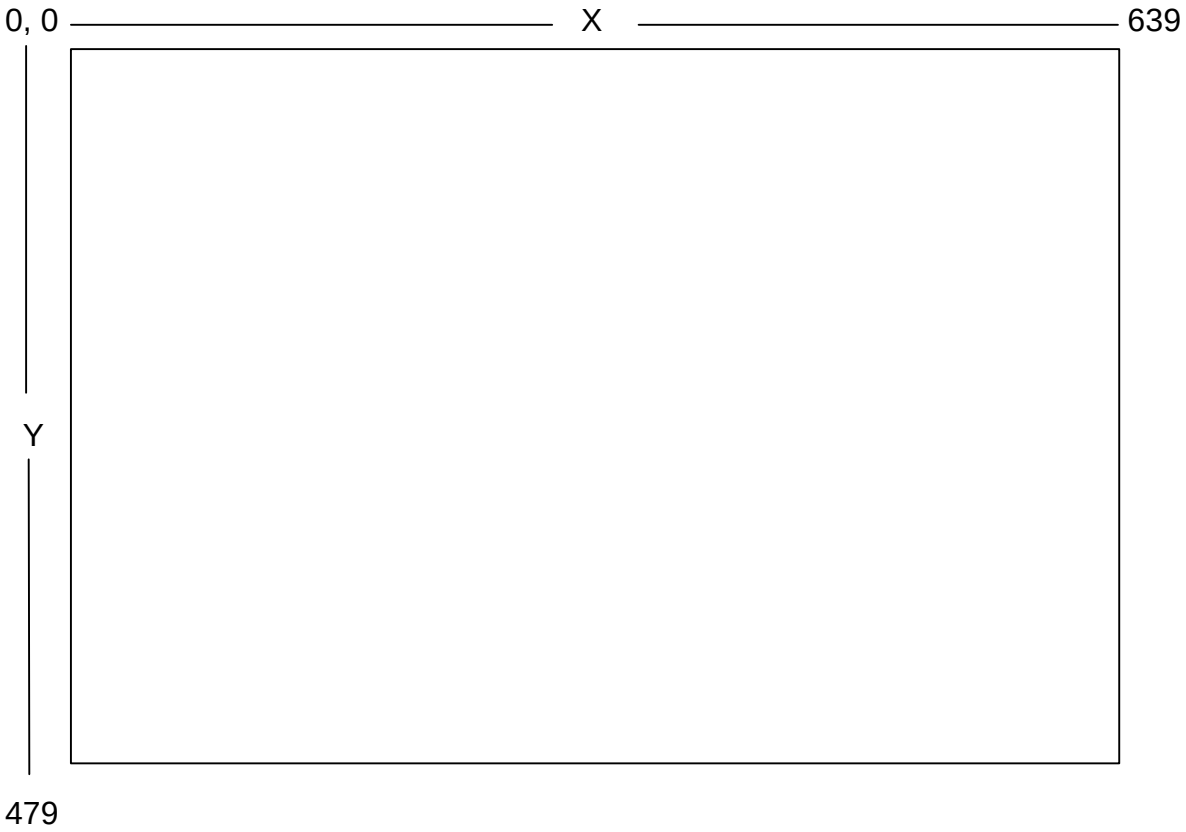

Figure 2

Alignment Image for RS-170,
640 Pixel Digitizer.

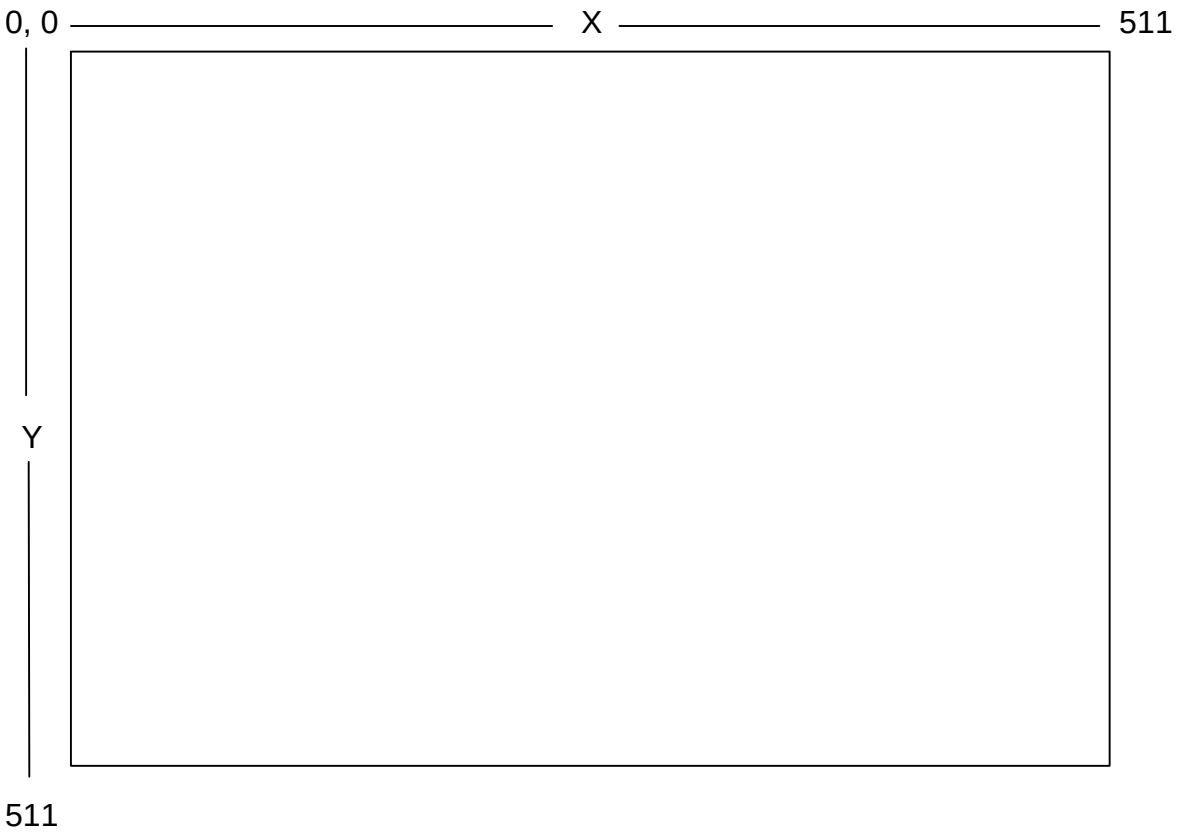

Figure 3

Alignment Image for CCIR
512 Pixel Digitizer.

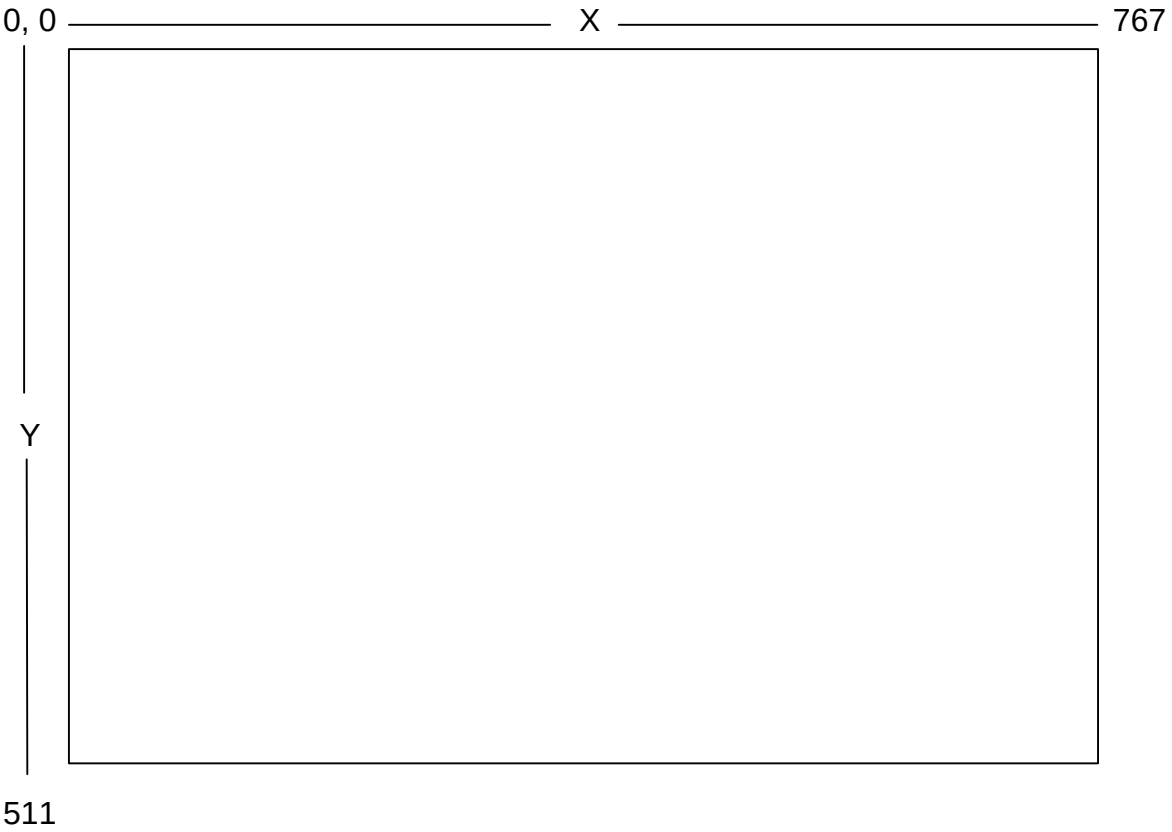

Figure 4

Alignment Image for CCIR
768 Pixel Digitizer.

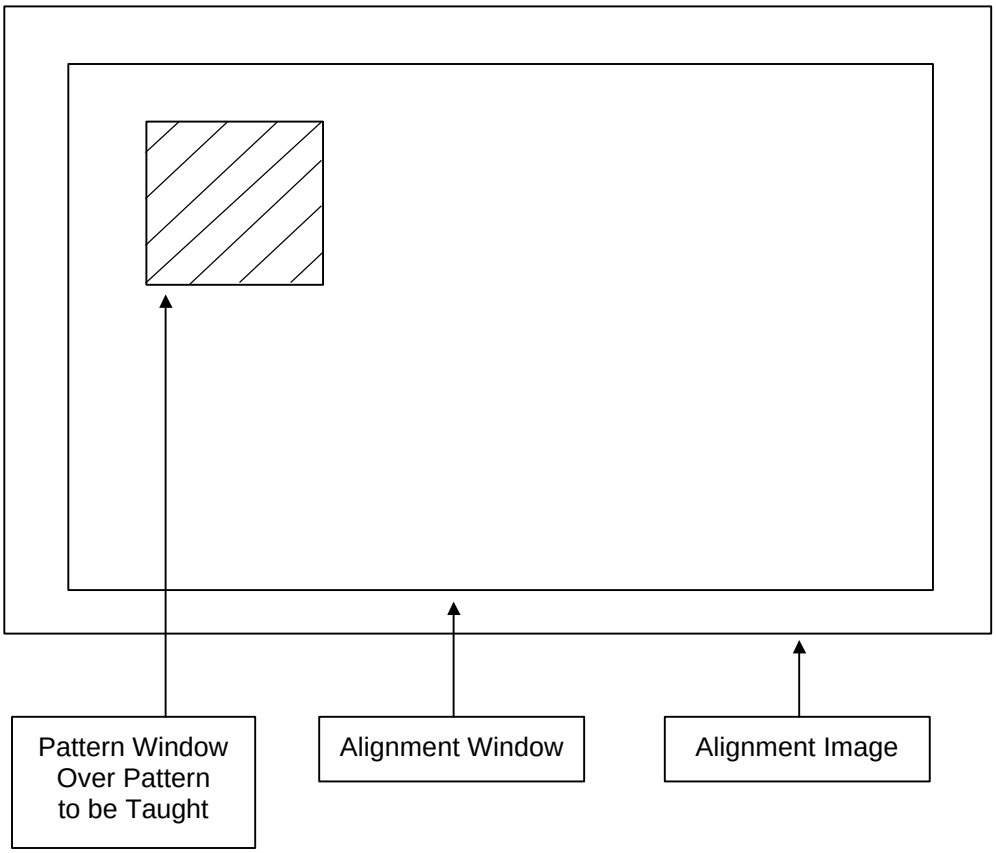

Figure 5

Window Relationships for Teaching.

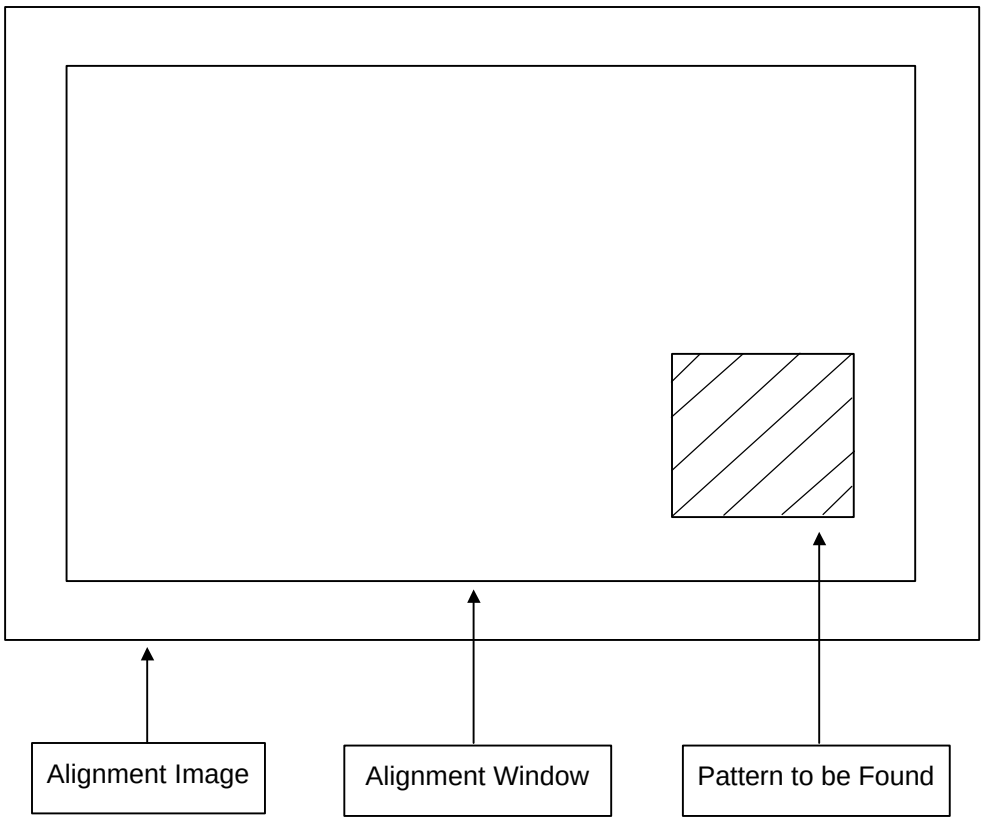

Figure 6
Window Relationships for Alignment.

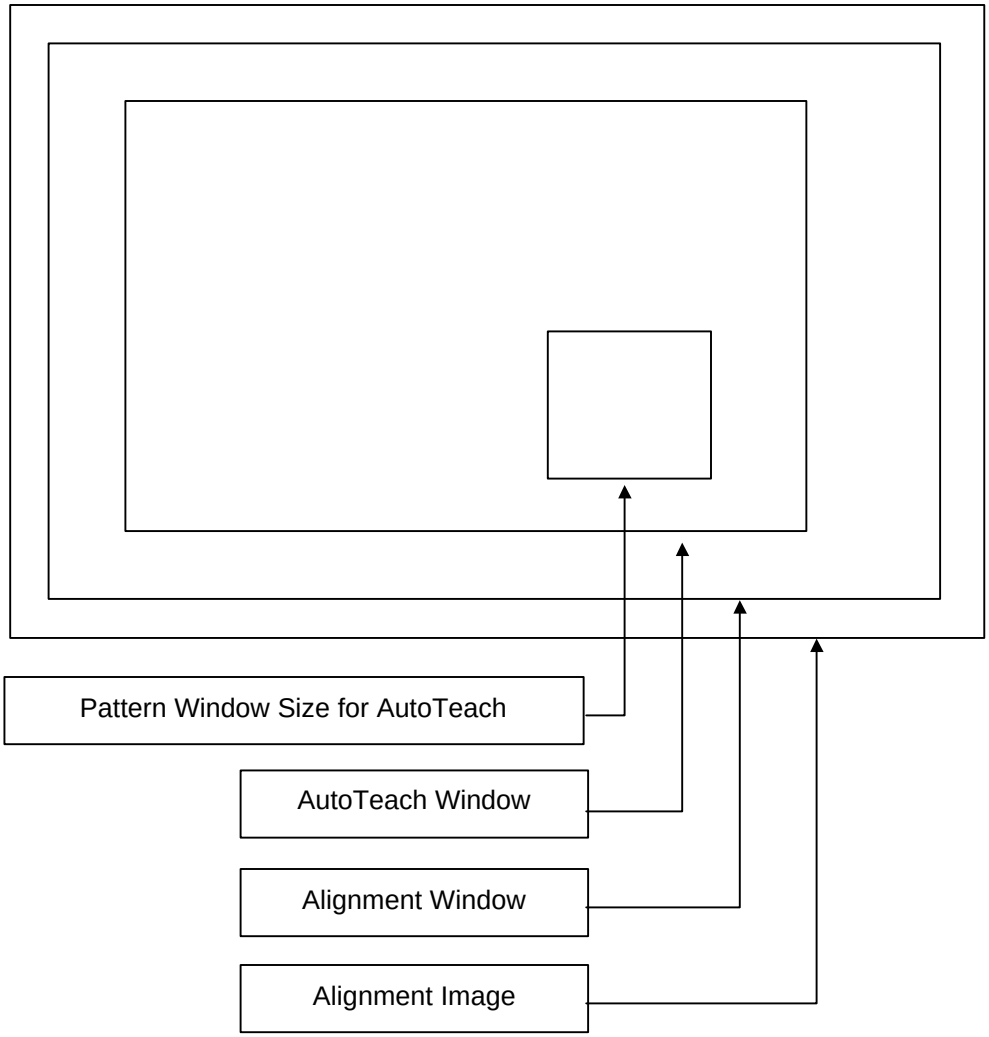

Figure 7
Window Relationships for AutoTeaching.

PART III: INSTALLATION AND CONFIGURATION

Operating Environment

ALIGN is written in C and 80x86 compatible assembly language. All code was developed on an IBM PC compatible computer using the following software environment:

- Microsoft Windows NT Version 4.0
- Microsoft Visual C/C++ Version 4.0.

The Alignment System is designed to operate in the following hardware environment:

- IBM PC or 100% compatible
- 640k RAM
- 16 Meg additional RAM
- A monitor and graphics card capable of displaying a screen resolution of 800 x 600 x 256
- MuTech MV-1000 frame grabber.

The Alignment System is designed to operate in the following software environment:

- Microsoft Windows NT V 3.51 or V 4.0
- Any software development environment that supports calling DLLs.

Supported Frame Grabber

The following frame grabber is supported:

Vision Hardware	Standard Image Size	Frame Buffer Size	Bus
MuTech MV-1000	640x480	hardware option dependent	PCI

Windows NT Installation for the MV-1000

The distribution disk, labeled **Disk 1,** contains the file README.WRI which gives complete instructions for installation as well as any important information not in this manual due to changes made after publication. **It is strongly urged that README.WRI be reviewed before proceeding to install the ALIGN software.** However, the instructions below should serve to help with installing the ALIGN software.

1. Run Windows NT.
2. Insert the distribution disk, labeled **Disk 1,** into the floppy drive.
3. For
    a. NT V 3.51 - From the Program Manager menu select File | Run.
    b. NT V 4.0 - From the Start menu choose Run.
4. Type "A:\SETUP" or "B:\SETUP" depending on which floppy drive contains the distribution disk.
5. Setup will display a Dialog box with four options. Select the first option to install the Interactive ALIGN software only. The Align DLLs, other Visual Basic™ required DLLs and OCXs will also be installed because they are required to run the Interactive Align program. Select the second option to install only the ALIGN DLLs and libraries. Select the third option to install only the ALIGN sample Visual Basic project. Select the fourth option for a complete installation.

Because installation is being performed in a Windows NT environment, installation of all DLLs and OCXs are handled by the install program and are “properly” registered with the Registry.

There is an initialization file, ALINT.INI, that needs to be placed in the Windows NT directory. Setup will copy the file to this directory. The contents of ALINT.INI are described in the Configuration section of this manual.

Windows NT Configuration for the MV-1000

Configuration for the Windows NT version of ALIGN is performed via the initialization file ALINT.INI. The initialization file has the following lines:

```
[FrameGrabber]
name=mv-1000
baseAddress=0x230
overlayPage=0xd000
frequency=MN_RS170
mode=SLAVE
iniFile=c:\mv-1000\camcfg\rs170.ini
```

Each line is explained below. The [FrameGrabber] line must be the 1st line of the file. The other lines can appear in any order.

[FrameGrabber]
This indicates that the lines below refer to the frame grabber hardware.

name=mv-1000
The name that MNEMONICS, INC. assigns to a particular frame grabber. Support for additional frame grabbers may be added in a future release.

baseAddress=0x230
The base I/O address of the frame grabber. This value is not used in Windows NT.

overlayPage=0xd000
The base segment address of the frame grabber overlay page. This value is not used in Windows NT.

frequency=MN_RS170
This entry indicates the camera compatibility of the frame grabber. Use MN_RS170 for an RS 170 camera. Use MN_CCIR for a CCIR camera. If the value MN_CAMERA is used then a camera initialization file is used to set up the frame grabber. The camera initialization file is specified in the iniFile line of the initialization file. See below.

Possible values are:
MN_RS170
MN_CCIR
MN_CAMERA

mode=SLAVE

This entry indicates the PCI bus master mode for PCI based frame grabbers.

Possible values are:

SLAVE
MASTER

iniFile=[path]name.ext; camera initialization file

This entry indicates the path and filename of the file to be used to initialize the frame grabber. This entry is used only when the mode=MN_CAMERA line is found in the configuration file and ignored if mode=MN_CCIR or mode=MN_RS170 lines are found.

PART IV: INTERACTIVE ALIGNMENT FOR WINDOWS NT

Description

The ALIGN Interactive Program provides a convenient environment for familiarizing the user with the capabilities of the functions in the ALIGN DLL. The program can also be used as a fast prototyping tool for defining alignment strategies and testing solutions to alignment problems.

Windows NT Considerations

The ALIGN Interactive Program is designed to be run under a Windows NT environment with a screen resolution of 800 x 600 and 256 colors. Higher screen resolutions are acceptable but the program will not run at the standard Windows NT resolution of 640 x 480. Use the Windows NT driver supplied with the VGA board to set the screen resolution to 800 x 600 and the number of colors to 256.

Required Files

The ALIGN Interactive Program is written in Microsoft Visual Basic V 4.0 and requires the following files:

ALIGNNT.EXE	:	a Visual Basic Version 4.0 based executable
ALINT.INI	:	the initialization file for ALIGNNT.EXE. This file must be located in the \WINDOWS NT directory.
MNSHELL.DLL	:	DLL providing low-level functions for accessing the frame grabber hardware
MNALIGNT.DLL	:	DLL containing ALIGN functions
MNAOISET.DLL	:	DLL containing support functions for setting rectangular AOIs interactively.

The program also requires a DLL and a driver for the frame grabber, DLLs and driver for the Sentinel software protection device, and additional DLLs and OCXs which support the implementation of the ALIGN Interactive Program in Visual Basic. It is expected that all drivers and DLLs for the frame grabber and for the software protection device are installed prior to running ALIGNNT.EXE. A separate disk is provided with ALIGN NT for installation of the software protection drivers. Driver installation hardware is included with the MV-1000 frame grabber.

Note: ALIGN NT will run without the Sentinel software protection device and its associated drivers. However, it will run in Demonstration Mode only.

ALIGN Windows or Areas of Interest (AOIs)

Most alignment operations require the specification of one or more of the three ALIGN windows or AOIs - the Pattern Window, the Alignment Window or the AutoTeach Window. All of these windows are rectangular and are specified by four values representing the location and size of the AOI. The first two values are the x and y coordinates of the upper left hand corner of the AOI. The second two values are the x size and y size of the AOI. All values are in pixels. AOI location and size will be represented on the screen in the format - (x_location, y_location) x_size by y_size. The ALIGN convention is to display these windows in consistent colors that provide a visual cue to the user as to the type of window being manipulated. The Pattern Window is displayed in green, the Alignment Window in red, and the AutoTeach Window in yellow.

The alignment operations of teaching, aligning, and AutoTeaching all require the AOIs to be set interactively on the Image Display Area of the screen. The AOI is displayed as a colored rectangular overlay on the image. The mouse is used to set the size and the location of the AOI. To set the location of the AOI, depress and hold the left mouse button. Use the mouse to move the entire AOI and release the left button when the AOI is in the desired position. To change the size of the AOI use the sizing handles on the perimeter of the AOI. Place the cursor on a sizing handle and depress and hold the left mouse button. Set the size of the AOI by moving the mouse. Release the left mouse button when the AOI is set to the desired size. The sizing handles located in the center of each side of the AOI are used to adjust the sides individually. The sizing handles on the corners of the AOI are used to adjust the two adjacent sides in tandem. When the location and size of the AOI have been set, click the right mouse button to signal the completion of the operation.

Screen Overview

The ALIGN Interactive Screen displays the image from the video source and provides menu items and control groups for performing alignment functions. The dominant screen element is the Image Display Area located in the left central portion of the screen. Just below the Image Display Area is a black rectangular window that is used as a Message Pad for instructions and results of alignment operations. The menu bar at the top of the screen provides access to a variety of alignment and support functions. The vertical column of controls on the right side of the screen provides the interface for performing the most common alignment operations. The uppermost group in this column is the Image Control Group which controls acquisition and display of the video image. Below this, the Align Control Group controls the most commonly used alignment functions such as teaching and aligning a pattern. Below the Align Group, the Parameter Control Group provides access to the parameters that are used to tailor alignment performance. The Windows Group is the lowest group in the control column and provides the location and size of the Alignment Window and the current Pattern Window. Each screen element will be described individually and in detail below.

Image Display Area

The Image Display Area is located in the left central portion on the screen. This area is used to display the images which are digitized from the video source. The Image Control Group provides for control of the digitization process.

Message Pad

The Message Pad is a rectangular window located below the Image Display Area. This window is used for display of instructions, status, data, or results related to the various alignment operations. The text displayed on the Message Pad is color keyed to provide a visual cue as to the type of information presented. Instructions are displayed in yellow. Status information and results are displayed in green. Red is used to display cautionary messages, such as error conditions, or to indicate results that are outside of the normal range.

Image Control Group

The Image Control Group consists of two command buttons which provide video image acquisition and display. A status line below the two command buttons indicates the current state of image acquisition. The Snap Command Button digitizes a single frame from the video source and displays it in the Image Display Area. The Live Command Button simulates a live video display by continuously digitizing and displaying frames from the video source. The live display may be terminated by any mouse click, any key press, or by clicking the Snap Command Button. When live display is terminated the Image Display Area will continue to display the last frame acquired from the video source. Both the Image Control Group status line and the Message Pad indicate the current status of video acquisition - either "Live" or "Snapped".

Align Control Group

The Align Control Group provides access to the most frequently used alignment operations. This group consists of three command buttons - the Teach Command Button, the Align Command Button, and the Align Win Command Button.

The Teach Command Button is used to teach a single alignment pattern. Prior to teaching a pattern, the pattern number may be selected using the Parameters Control Group and the Alignment Window may be set using the Align Win Command Button. To teach a pattern, click on the Teach Command Button and follow the instructions on the Message Pad to specify the Pattern Window. Results of the teach operation are displayed on the Message Pad. Results for acceptable patterns are shown in green. Results for patterns with Teach Scores of 0 or 1 are shown in red. These are patterns that have the potential to generate problematic alignment performance. If the teach operation generates a Teach Score of 0 or 1, try reteaching the pattern using a

different size or location of the Pattern Window or the Alignment Window. At the completion of the teaching operation, the Pattern Window size and location are displayed in the Windows Group at the bottom of the Control Column. Any prior pattern with the same pattern number will be overwritten by the pattern teaching operation.

The ALIGN Command Button performs a single alignment. Prior to performing a pattern alignment, the pattern number may be selected using the Parameters Control Group and the Alignment Window may be set using the ALIGN Win Command Button. To perform a single pattern alignment on the current image, click on the ALIGN Command Button. At the completion of the pattern alignment the position of best alignment is shown as a rectangular overlay on the Image Display Area. The results of the alignment operation are written to the Message Pad. The results are shown in green if the Alignment Quality exceeds the Early-Stop Threshold. Otherwise, the results are indicated in red.

The ALIGN Win Command Button is used for setting the location and size of the Alignment Window. All subsequent pattern teaching, aligning, and AutoTeaching operations use the specified Alignment Window. To set the Alignment Window, click on the ALIGN Win Command Button and follow the instructions on the Message Pad. The size and location of the Alignment Window is displayed on the Message Pad and in the Windows Group at the completion of the operation.

Parameters Control Group

The Parameters Control Group provides three horizontal scroll bars (HSB) for setting the pattern number and two alignment parameters - Search Depth and Early-Stop Threshold. In general, any HSB can be used in three different ways to select a value.

1. Use the left arrow to decrease the value and the right arrow to increase the value. A single click on the arrow will decrement or increment the value by one. Depressing and holding the mouse button will continually decrement or increment the value until it reaches its minimum or maximum limit.

2. Use the bar area to the left of the thumb to decrease the value and the area to the right of the thumb to increase the value. A single click will cause the value to be decreased or increased a single time. Depressing and holding the mouse button will cause the value to decrease or increase continually until the minimum or maximum limit is reached. In the case of the Pattern Number HSB and the Search Depth HSB, the value is changed by 1. The bar on the Early-Stop HSB will change the value by 5.

3. Use the thumb to set the value. Depress and hold the mouse button and move the thumb to the desired position. The value will be set corresponding to the position of the thumb on the horizontal scale.

The Pattern Number HSB is used for setting the current pattern number. The current pattern number will be used in all subsequent teaching and alignment operations. Pattern numbers range from 1 to 16. When a pattern number is selected, the Message Pad will display that pattern's teach values, or indicate that the pattern has not yet been taught. The location and size of the selected pattern is also displayed in the Windows Group.

The Search Depth HSB is used for setting the alignment Search Depth. The selected Search Depth value will be used for all subsequent alignments. Set the Search Depth high enough to generate robust alignment performance, but no higher. As a general rule, for any given pattern, take the Search Depth reported by that pattern's teach results and add 3. This provides a reasonable starting point for an alignment Search Depth. For alignment operations which do not Early-Stop, the alignment time will be a function of the Search Depth. Search Depth values range from 1 to 16. Higher numbers indicate that the alignment process works harder or searches more locations to find the best pattern match. A Search Depth of 16 invokes ancillary levels of processing that require significant additional execution time. Use a Search Depth of 16 only for required patterns that can not be retaught and cannot be found using lower Search Depth values.

The Early-Stop HSB is used to set the alignment Early-Stop Threshold. The Early-Stop Threshold is an alignment Quality value and ranges from -100 to +100. During the alignment process, any pattern match with a Quality value above the Early-Stop Threshold will cause the alignment process to terminate. The alignment results will report the location of the first pattern match above the Early-Stop Threshold. Appropriate use of the Early-Stop Threshold can radically reduce alignment times. The Early-Stop Threshold should be set low enough to allow the alignment operation to Early-Stop. At the same time, the Early-Stop Threshold must be set high enough to prevent alignment on similar but incorrect patterns. The range of acceptable Early-Stop Threshold values becomes smaller as the pattern becomes less unique. As a general rule, for any given pattern, set the Early-Stop Threshold halfway between the Next-Best Quality value and 100. Raise this value if the alignment process is finding patterns that are similar but incorrect.

Windows Group

The Windows Group displays the location and size of the Alignment Window and the currently selected pattern. The Pattern Window values are updated after any use of the Pattern Number HSB or the Teach Command Button. The Alignment Window values are updated after any use of the Align Win Command Button.

Menu Selections

The menu bar at the top of the ALIGN screen provides access to alignment support functions and less frequently used alignment commands. Each of the menu and submenu selections is described below.

File

This menu selection provides access to ALIGN file operations. ALIGN supports five file types as shown below:

1. Single Pattern File - a file containing a single ALIGN pattern
2. All Patterns File - a file containing all 16 patterns in the internal ALIGN Pattern Data Base
3. ALIGN Image File - a file containing a frame buffer image in ALIGN format
4. BMP Image File - a file containing a frame buffer image in BMP format
5. TIFF Image File - a file containing a frame buffer image in TIFF format

The File Submenu selections provide the capability to save any type of file to disk and to restore any type of file from disk. The File Submenu selections are -

Save a Single Pattern
Save All Patterns

Save an ALIGN Image
Save a BMP Image
Save a TIFF Image

Restore a Single Pattern
Restore All Patterns

Restore an ALIGN Image
Restore a BMP Image
Restore a TIFF Image

Exit

All File Submenu selections present a window that allows the specification of the disk drive, the directory path name, and the file name of the file to be saved or restored. When saving files, the user will be warned if the file already exists, and prompted for overwrite. File operations can be canceled at any time during the file selection process

by clicking on the Cancel button. Results of the file operation are shown on the Message Pad in green. Each File Submenu selection is described below.

File | Save a Single Pattern

This menu selection saves a single ALIGN pattern to a Single Pattern File on disk. Use the Pattern Number HSB to select the pattern number prior to saving the file. Any pattern may be saved in a Single Pattern File. The suggested file extension for Single Pattern Files is **.PAT**.

File | Save All Patterns

This menu selection saves all 16 patterns in the ALIGN internal Pattern Data Base to an All Patterns File on disk. The complete Pattern Data Base, as it currently exists, is copied to the disk file. The suggested file extension for All Pattern Files is **.ALL**.

File | Save an ALIGN Image

This menu selection saves the current frame buffer image to an Align Image File on disk. The suggested file extension for ALIGN Image Files is **.IMG**.

File | Save a BMP Image

This menu selection saves the current frame buffer to a BMP Image File on disk. The suggested file extension for BMP Image Files is **.BMP**.

File | Save a TIFF Image

This menu selection saves the current frame buffer to a TIFF Image File on disk. The suggested file extension for TIFF Image Files is **.TIF**.

File | Restore a Single Pattern

This menu selection restores an ALIGN pattern from a Single Pattern File on disk. Use the Pattern Number HSB to select the pattern number prior to restoring the file. Single Pattern Files are not linked to any particular pattern number. Patterns do not have to be restored to the same pattern number under which they were saved. For example, a pattern could be moved from Pattern 1 to Pattern 3 in the internal data base by the following method. First use the Pattern Number HSB to select Pattern 1 and then save a Single Pattern File. Next, use the Pattern Number HSB to select Pattern 3 and restore the Single Pattern File. The net result of these operations is that Pattern 1 has been copied to Pattern 3 in the ALIGN internal Pattern Data Base. Note, if the pattern selected by the Pattern Number HSB already exists, it will be overwritten by the pattern restore operation. Only Single Pattern Files may be restored by this menu selection.

File | Restore All Patterns

This menu selection restores the complete ALIGN internal Pattern Data Base of 16 patterns. The Pattern Data Base will be restored to a state that is identical to the state under which it was saved. That is, if a particular pattern was not taught at the time the All Pattern File was saved, then that pattern will not be taught when the file is restored. Note, all currently taught patterns will be overwritten and replaced by patterns from the All Patterns File. Only All Pattern Files may be restored by this menu selection.

File | Restore an ALIGN Image

This menu selection restores an ALIGN Image File. The current contents of the frame buffer are replaced with the contents of the image file. Only ALIGN Image Files may be restored by this menu selection.

File | Restore a BMP Image

This menu selection restores a BMP Image File. The current contents of the frame buffer are replaced with the contents of the image file. Only BMP Image Files may be restored by this menu selection.

File | Restore a TIFF Image

This menu selection restores a TIFF Image File. The current contents of the frame buffer are replaced with the contents of the image file. Only TIFF Image Files may be restored by this menu selection.

File | Exit

This menu selection exits the ALIGN program. All patterns will be lost.

AutoTeach

This menu selection teaches patterns automatically, selecting the best patterns for alignment. The AutoTeach process consists of three steps and may be canceled at any time by clicking on the appropriate Cancel button. Note that all patterns currently residing in the range of pattern numbers selected by AutoTeach will be overwritten.

The first step of the AutoTeach process displays a Starting Pattern HSB and a Number of Patterns HSB. Use these two HSBs to set the first pattern to teach and the number of patterns to teach. If more than one pattern is to be taught, the patterns will be stored in numerical order beginning with the starting pattern. For example, setting the starting pattern to 4 and the number of patterns to 3 will cause the AutoTeach process to teach Pattern 4, Pattern 5 and Pattern 6.

The second step involves setting, in order, the Alignment Window, the AutoTeach Window, and the Pattern Window. The Alignment Window represents the area to be searched, the AutoTeach Window represents the area from which the patterns are to be selected, and the Pattern Window specifies the size of the pattern to be selected. Note that when setting the Pattern Window, only the size of the pattern is used by AutoTeach.

In the third step, the patterns are automatically selected by AutoTeach. A pattern of the specified size is stepped through the AutoTeach Window. AutoTeach selects the best patterns for alignment and stores those patterns in the selected locations in the ALIGN Pattern Data Base. At the conclusion of the selection process, the teach results for the patterns are displayed. The patterns in the display are ranked by Teach Score. This is not meant to indicate that patterns with higher Teach Scores are better patterns. The selection of the best pattern for any alignment scenario is image dependent and task specific. Often, the Teach Search Depth and the Next-Best Quality are stronger measures of pattern performance than is Teach Score.

Patterns

This menu selection provides two mechanisms for displaying pattern information. The first displays the values resulting from the teach process. The second displays the location and size of the taught pattern. The Patterns Submenu selections are -

Information
Display

Each Patterns Submenu selection is described below.

Patterns | Information

This menu selection displays the size, Teach Search Depth, Next-Best Quality, Teach Score, and the location of the patterns at teach time. The pattern currently selected by the Pattern Number HSB will be highlighted.

Patterns | Display

This menu selection displays the size and location of the Pattern Window on the image in the Image Display Area. The display corresponds to the pattern currently selected by the Pattern Number HSB. The image information inside of the Pattern Window will be irrelevant unless the image currently in the Image Display Area is identical to the image that was used to teach the pattern. If documentation and tracking of the image information associated with taught patterns is required, this can be accomplished by saving the teach image as a disk file and then restoring the image prior to displaying the pattern.

Continuous Align

This menu selection is used to start or stop Continuous Align. Continuous Align is a mode in which the system acquires an image from the video source, performs an alignment, displays the results of the alignment, and then repeats the sequence continuously. Continuous Align can be used to investigate the tracking ability of a particular pattern or to investigate the alignment performance of a pattern with respect to degradation in the image such as changes in lighting or focus. Start Continuous Align and then move the image or adjust image quality while observing the alignment results. Continuous Align can also be utilized as an aide in determining a reasonable Search Depth and Early-Stop Threshold. The Pattern Number, Search Depth, Early-Stop Threshold, and the Alignment Window can all be changed while Continuous Align is running. The effect of changes in these parameters on alignment performance can be seen by monitoring the alignment results. The Continuous Align Submenu selections are -

Start
Stop

Each Continuous Align Submenu selection is described below.

Continuous Align | Start

This menu selection starts Continuous Align. A check mark on the submenu indicates the currently selected entry.

Continuous Align | Stop

This menu selection stops Continuous Align. A check mark on the submenu indicates the currently selected entry.

Sublign

This menu selection controls the sub-pixel alignment capabilities of ALIGN. Submenu selections are used to enable or disable Sublign and to select the sub-pixel resolution. The Sublign Submenu selections are -

On
Off

1/2 pixel
1/4 pixel
1/8 pixel
1/16 pixel
1/32 pixel

Each Sublign Submenu selection is described below.

Sublign | On

This menu selection is used to enable sub-pixel alignment. All subsequent single aligns and continuous aligns will perform sub-pixel alignment to the specified resolution. Sublign can be turned on and off while ALIGN is in Continuous Align mode. A check mark on the menu indicates if Sublign is currently on or off.

Sublign | Off

This menu selection is used to disable sub-pixel alignment. Sublign can be turned on and off while ALIGN is in Continuous Align mode. A check mark on the menu indicates if Sublign is currently on or off.

Sublign | 1/2 Pixel

This menu selection is used to set the Sublign sub-pixel resolution to 1/2 pixel. This means that ALIGN will find the best pattern alignment to the nearest 1/2 pixel. The Sublign resolution can be changed while ALIGN is in Continuous Align mode. A check mark on the menu indicates the currently selected resolution. Higher resolutions require additional alignment time.

Sublign | 1/4 Pixel

This menu selection is used to set the Sublign sub-pixel resolution to 1/4 pixel.

Sublign | 1/8 Pixel

This menu selection is used to set the Sublign sub-pixel resolution to 1/8 pixel.

Sublign | 1/16 Pixel

This menu selection is used to set the Sublign sub-pixel resolution to 1/16 pixel.

Sublign | 1/32 Pixel

This menu selection is used to set the Sublign sub-pixel resolution to 1/32 pixel.

Version

This menu selection displays the current version number of the ALIGN DLL.

Exit

This menu selection exits the ALIGN program. All patterns will be lost.

Quick Reference for ALIGN Interactive

The Quick Reference provides step-by-step instructions for accomplishing the most frequently used alignment operations. All Quick Reference headings have a QR prefix. The following Quick Reference instructions are available:

- Displaying a Live Image
- Digitizing an Image
- Setting the Alignment Window
- Teaching a Pattern
- AutoTeaching a Pattern
- Aligning a Pattern
- Continuously Aligning a Pattern
- Saving a Single Pattern
- Restoring a Single Pattern.

QR - Displaying a Live Image

Click on the Live Command Button in the Image Control Group. Any subsequent mouse click or key press will digitize the last frame and display it in the Image Display Area.

QR - Digitizing an Image

Click on the Snap Command Button in the Image Control Group. The image will be displayed in the Image Display Area.

QR - Setting the Alignment Window

Click on the Align Win Command Button in the Align Control Group and follow the instructions to interactively set the Alignment Window. Results will be displayed on the Message Pad.

QR - Teaching a Pattern

First, use the Pattern Number HSB in the Parameters Control Group to set the pattern number. Then, click on the Teach Command Button in the Align Control Group and follow the instructions to interactively set the Pattern Window and teach the pattern. Results will be displayed on the Message Pad.

QR - AutoTeaching a Pattern

Click on the AutoTeach | Start menu selection. Specify the starting pattern number and the number of patterns to be taught. Then, follow the instructions to interactively set the Alignment Window, AutoTeach Window, and the Pattern Window. Results of AutoTeach will be displayed.

QR - Aligning a Pattern

Click on the Align Command Button in the Align Control Group. The alignment will be performed using the current pattern number, Alignment Window, Search Depth, and Early-Stop Threshold. Results will be displayed on the Message Pad.

QR - Continuously Aligning a Pattern

Click on the Continuous Align | Start menu selection. Continuous alignment will be performed using the current pattern number, Alignment Window, Search Depth, and Early-Stop Threshold. Any of these parameters can be modified while Continuous Align is running. Results will be displayed on the Message Pad.

QR - Saving a Single Pattern

First, select the pattern to be saved using the Pattern Number HSB in the Parameters Control Group. Then, click on the File | Save a Single Pattern menu selection and select a file name. The pattern will be saved to the specified file on disk.

QR - Restoring a Single Pattern

First, select the pattern to be restored using the Pattern Number HSB in the Parameters Control Group. Then, click on the File | Restore a Single Pattern menu selection and select a file name. The pattern will be restored from the specified file on disk.

PART V: APPLICATION PROGRAMMING

There are two different application programming environments for which sample code is provided:

- Microsoft Visual Basic Version 4.0
- Microsoft Visual C/C++ Version 4.0

Each of these environments has different considerations for creating ALIGN applications. They will be described separately. It is assumed that the user of ALIGN has familiarity with one of the software development environments listed above.

General Considerations When Creating ALIGN Applications

ALIGN, in the Windows NT environment, is a set of Dynamic Linked Libraries (DLLs). As such, ALIGN application development can take place in *any* application development environment if the application environment provides for access to Windows 32 Bit DLLs. MNEMONICS, INC. provides, in text and library form, the information necessary for accessing the DLLs.

There are two DLLs required to run ALIGN applications; they are MNALIGNT.DLL and MNSHELL.DLL. MNALIGNT.DLL encapsulates the alignment functions as well as providing image acquisition functions. MNSHELL.DLL encapsulates the frame grabber hardware. MNALIGNT.DLL calls functions in MNSHELL.DLL to control and access the frame grabber. Developers should not have to directly call functions in MNSHELL.DLL.

ALIGN functions are provided for interactively setting the Alignment Window, the Pattern Window or the AutoTeach Window. These functions are provided in a separate library, MNAOISET.DLL.

There are ALIGN functions available for displaying the contents of the frame buffer. These functions will display an 8 bit gray scale image in a window with scale of 1 to 1. Depending on the frame grabber and the screen resolution, this may cause images to be distorted, for example, a circle might appear as an ellipse. It is the responsibility of the user to handle these display considerations.

The ALIGN DLLs need to be in the proper directory to be accessed. This is a function of how Windows NT is set up or how the application is to be executed. It is the developer's responsibility to make sure that the DLLs are in the proper location, though the setup program tries to place the DLLs in the correct directory.

An initialization file, ALINT.INI, must be used by all applications to set up the hardware and software options for the target frame grabber. See Part III Setup and Configuration for more information about ALINT.INI.

Note that when designing an application program, the functions **ali_configure** and **ali_initialize** MUST be called before any other ALIGN functions. Also, the function **ali_terminate** MUST be called prior to the end of the program.

Microsoft Visual C/C++

The files required for the C/C++ environment are:

ALI.H	-	the file that contains the ALIGN function prototypes as well as defines for error codes, set up, etc.
MNALIGNT.LIB	-	the export library for the ALIGN DLL, MNALIGNT.DLL
MNSHELL.LIB	-	the export library for the frame grabber interface, MNSHELL.DLL

Include ALI.H into source files that call ALIGN functions. When including ali.h in the project make sure that WIN32 is defined. Add MNALIGNT.LIB and MNSHELL.LIB to the project.

Microsoft Visual Basic

The file required for the Visual Basic environment is:

DECLARE.TXT	-	the function and subroutine declarations for Visual Basic applications.

This file should be added to the Project.

Sample Program in Visual Basic

A complete sample program in Visual Basic is provided. This program can be used as a design guide and teaching aid for writing an alignment application.

Description

This program demonstrates the design of an application that embodies the most frequently used alignment operations. Images can be digitized from the video source and displayed in a window. Patterns and Alignment Windows can be taught using the interactive AOI capabilities. The pattern that is taught can then be aligned on subsequent images. Results and instructions for all operations are displayed in a window that is used as a Message Pad. The Sample Program Screen contains a Picture Box for displaying the image, a Text Box for displaying instructions and results,

and six Command Buttons providing the functions - Live, Snap, Set Align Win, Teach, Align, and Exit. Complete source code is provided for the Visual Basic implementation.

Files

The following files are required. These files are automatically installed by the Setup Program provided with ALIGN.

MAINS.FRM	-	a source file containing all the controls and procedures in the sample program
DECLARES.TXT	-	a source file containing the declarations for all alignment functions
WIN32APIS.BAS	-	a file which contains definitions of the Windows NT 32 bit API constants used in the sample program source code
ALIGNNTS.EXE	-	the Visual Basic Sample Program in executable form
MNALIGNT.DLL	-	the DLL containing the alignment functions
MNAOISET.DLL	-	the DLL containing the AOI functions
MNSHELL.DLL	-	the DLL containing frame grabber support functions
ALINT.INI	-	the ALIGN initialization file, which must be located in the \WINDOWS NT directory.

There will be other Visual Basic files required to support the implementation, such as the .OCX files needed for the controls used by MAINS.FRM. These files are available when Visual Basic Version 4.0 has been installed. In addition, the program requires standard Windows NT DLLs such as WINMM.DLL and COMDLG32.DLL.

To begin using the sample program at the source code level, open a new project in Visual Basic and add the files MAINS.FRM, DECLARES.TXT, and WIN32APIS.BAS.

Controls

The following Visual Basic Controls are used:

1. **picVideo** (Picture Box Control) - this control is used to display the image from the frame buffer

2. **lblMPad** (Label Control) - this control is used as a Message Pad for instructions and results

3. **cmdLive** (Command Button Control) - this control is used to display a simulated live video image in the Picture Box Control

4. **cmdSnap** (Command Button Control) - this control is used to acquire a single frame from the video source and display it in the Picture Box Control

5. **cmdSetAlignW** (Command Button Control) - this control is used to display the Alignment Window on the image and to set it interactively

6. **cmdTeach** (Command Button Control) - this control is used to teach a pattern by interactively setting a Pattern Window on the image

7. **cmdAlign** (Command Button Control) - this control is used to align the taught pattern on the current image

All controls are specified in the file MAINS.FRM.

Event Procedures

The following event procedures are used by the sample program. All event procedures are located in the file MAINS.FRM.

1. Sub **cmdAlign_Click** () - the click event procedure for the Align Command Button. This procedure performs an alignment and displays the results on the image and the Message Pad.

2. Sub **cmdExit_Click** () - the click event procedure for the Exit Command Button. This procedure terminates the sample program.

3. Sub **cmdLive_Click** () - the click event procedure for the Live Command Button. This procedure displays simulated live video in the Picture Box.

4. Sub **cmdSetAlignWin_Click** () - the click event procedure for the Set Align Win Command Button. This procedure allows the Alignment Window to be set interactively using an AOI on the image display.

5. Sub **cmdSnap_Click** () - the click event procedure for the Snap Command Button. This procedure digitizes a single frame from the video source and displays the image in the Picture Box.

6. Sub **cmdTeach_Click** () - the click event procedure for the Teach Command Button. This procedure teaches a pattern interactively using an AOI on the image display.

7. Sub **Form_Load** () - the load event procedure for the main form. This procedure configures and initializes the alignment system.

8. Sub **Form_Resize** () - the resize event procedure for the main form. This procedure sets the size and location of the Picture Box Control used to display the frame buffer image.

9. Sub **Form_Unload** () - the unload event procedure for the main form. This procedure terminates the sample program.

10. Sub **picVIdeo_Paint** () - the paint event procedure for the Picture Box Control. This procedure displays the frame buffer image in the Picture Box.

General Procedures

The following general procedures are used by the sample program. All general procedures are located in the file MAINS.FRM.

1. Sub **Align_AOI** () - this procedure interactively sets the Alignment Window.

2. Sub **Align_cursor** (x As Integer, y As Integer, color As Long) - this procedure draws a cursor on the Picture Box image at location (x, y). A color may be specified in RGB format.

3. Sub **Align_markaw** (color As Long) - this procedure marks the Alignment Window by drawing a rectangle on the Picture Box image. The color may be specified in RGB format. In the sample program, the Alignment Window is always displayed in red.

4. Sub **Align_markpat** (x As Integer, y As Integer, color As Long) - this procedure marks the pattern on the Picture Box image at location (x, y) by displaying a rectangle with a center cursor. The color may be specified in RGB format. In the sample program, the Pattern Window is always displayed in green.

5. Sub **Align_rectangle** (x As Integer, y As Integer, dx As Integer, dy As Integer, color As Long) - this procedure draws the specified rectangle on the Picture Box image. The color is specified in RGB format.

6. Sub **Initialize** () - this procedure configures and initializes the alignment system.

7. Sub **MPad_align_error** () - this procedure displays an alignment error message on the Message Pad Text Box.

8. Sub **MPad_align_pnt** () - this procedure displays a pattern not taught message on the Message Pad Text Box.

9. Sub **MPad_align_results** () - this procedure displays the alignment results on the Message Pad Text Box.

10. Sub **MPad_aw_instruct** () - this procedure displays instructions for setting the Alignment Window on the Message Pad Text Box.

11. Sub **MPad_aw_results** () - this procedure displays Alignment Window results on the Message Pad Text Box.

12. Sub **MPad_clear** () - this procedure clears the text on the Message Pad Text Box.

13. Sub **MPad_live** () - this procedure displays a live video status on the Message Pad Text Box.

14. Sub **MPad_pw_instruct** () - this procedure displays instructions for setting the Pattern Window on the Message Pad Text Box.

15. Sub **MPad_ready** () - this procedure displays a ready message on the Message Pad Text Box.

16. Sub **MPad_snapped** () - this procedure displays snapped status on the Message Pad Text Box.

17. Sub **MPad_teach_error** () - this procedure displays a teach error on the Message Pad Text Box.

18. Sub **MPad_teach_results** () - this procedure displays the teach results on the Message Pad Text Box

19. Sub **Novideo** () - this procedure displays a Message Box indicating no video input to the frame buffer.

20. Sub **Pattern_AOI** () - this procedure sets the Pattern Window interactively.

21. Sub **Terminate** () - this procedure terminates the sample program.

22. Sub **VideoDisplay** () - this procedure displays the frame buffer image in the Picture Box.

23. Sub **VideoGrab** () - this procedure displays simulated live video in the Picture Box by continuously acquiring frames from the video source.

24. Sub **VideoSnap** () - this procedure acquires a single frame from the video source and displays it in the Picture Box.

Globals

In the sample program, global variables are used to specify the Alignment Window, the Pattern Window, and the results of teaching and alignment operations. These global variables are defined in the Declarations section of MAINS.FRM.

PART VI: ALIGNMENT LIBRARY

Introduction to the Alignment Library

This section contains a detailed description of each Alignment Library function. The information on each function is presented in three sections: FUNCTION, DESCRIPTION and EXAMPLE.

FUNCTION

This section defines the function call, the argument list, and the return values. Valid argument ranges are shown wherever possible. Extensive error checking is performed and every effort is made to detect invalid parameter values. Errors in parameter range will manifest themselves as a negative return value and an error message to the computer display. However, due to the interrelated nature of the parameter set, 100% error checking is impossible. It is the responsibility of the calling routine to insure that the parameter range and the number of parameters is valid. A C prototype and a Visual Basic declaration are provided for each function.

Note that the argument list for C variables of type int maps to the type long in Visual Basic Version 4.0. This is because Visual Basic Version 4.0 preserves the int type as a two byte value for backwards compatibility. In NT the int type is a 4 byte value.

RETURN VALUES

The return value of the function is used to indicate completion status or any error condition encountered in execution of the function. Return values are listed in terms of constants defined in the file ali.h. In general, positive return values indicate completion of the function without error and negative return values indicate that an error condition was encountered.

DESCRIPTION

This section contains a verbal description of the action of the function.

EXAMPLE

This section presents an example of the function in common usage.

PARAMETER RANGE

The allowable range of parameter values is given wherever possible. Many of these ranges are in terms of the maximum X and Y coordinates of the video image.

XDIM = x size of the frame buffer image
YDIM = y size of the frame buffer image

MuTech MV-1000

RS-170	XDIM = 640	YDIM = 480

ali_align : alignment

FUNCTION

C Prototype

int WINAPI **ali_align** (int patnum, int awx, int awy, int awdx, int awdy, int sd,
int est, int nr, int *nfptr, int *xptr, int *yptr,
int *qptr, double *cptr);

Visual Basic Declaration

Declare Function **ali_align** Lib "mnalignt.dll" (ByVal patnum As Long,
ByVal awx As Long, ByVal awy As Long, ByVal awdx As Long,
ByVal awdy As Long, ByVal sd As Long, ByVal est As Long,
ByVal nr As Long, nfptr As Long, xptr As Long, yptr As Long,
qptr As Long, cptr As Double) As Long

where

patnum	=	Pattern number 1 to 16
awx	=	X location of Alignment Window 0 to (XDIM - awdx)
awy	=	Y location of Alignment Window 0 to (YDIM - awdy)
awdx	=	X span of Alignment Window 16 to 512
awdy	=	Y span of Alignment Window 16 to 480
sd	=	Search Depth 1 to 16
est	=	Early-Stop Quality threshold -100 to +100
nr	=	number of alignment locations requested 1 to 64
nfptr	=	pointer to number of alignment locations found
xptr	=	pointer to X location of best alignment
yptr	=	pointer to Y location of best alignment
qptr	=	pointer to quality of best alignment
cptr	=	pointer to contrast ratio of best alignment

Return Values:

NO_ERROR	-	Alignment completed without Early-Stop
1	-	Alignment completed with Early-Stop invoked
PAT_NUM_ERR	-	Pattern Number out of range
PAT_NOT_TAUGHT	-	Pattern not taught
PAT_TOO_BIG	-	Pattern too large for Alignment Window
SRCH_DPTH_ERR	-	Search Depth out of range
EARLY_STP_Q_ERR	-	Early-Stop Quality threshold out of range
WIN_X_SPAN_ERR	-	Alignment Window x span out of range
WIN_Y_SPAN_ERR	-	Alignment Window y span out of range
WIN_X_ERR	-	Alignment Window x location out of range
WIN_Y_ERR	-	Alignment Window y location out of range
PROT_ERR	-	Protection device not installed

DESCRIPTION

This function performs the alignment task by searching the portion of the image within the specified Alignment Window for the best match to the specified pattern. The location, quality, and contrast ratio of the best alignment are available to the caller via four pointers. Note that the alignment process will terminate as soon as an alignment is found whose quality equals or exceeds the Early-Stop quality threshold.

EXAMPLES

Align Pattern Number 7 over an Alignment Window located at (60, 70) of x size 130 and y size 110. Use a Search Depth of 3 and an Early-Stop Quality threshold of 70. Check the return value for an error condition.

C/C++ EXAMPLE

```
int rv;         // alignment return value
int nf;         // number of alignment locations found
int x, y;       // alignment location
int q;          // alignment quality
double c;       // alignment contrast ratio
char str[80];   // message box string

...
```

```
rv = ali_align (7, 60, 70, 130, 110, 3, 70, 1, &nf, &x, &y, &q, &c);
if (rv < 0)
  {
  sprintf (str, "Alignment Error %d.", rv);
  ::MessageBox(NULL, str, "ali_align", MB_OK | MB_ICONSTOP);
  }
else
  {
  sprintf (str,
          "Best Alignment at (%d, %d).\r\n Quality = %d  Contrast Ratio = %lf ",
          x, y, q, c);
  ::MessageBox(NULL, str, "ali_align", MB_OK | MB_ICONINFORMATION);
}
```

VISUAL BASIC EXAMPLE

```
Dim rv As Long              ' alignment return value
Dim nf As Long              ' number of alignment locations found
Dim x As Long               ' x alignment location
Dim y As Long               ' y alignment location
Dim q As Long               ' alignment quality
Dim c As Double             ' alignment contrast ratio
Dim message As String       ' message box string

...

rv = ali_align(7, 60, 70, 130, 110, 3, 70, 1, nf, x, y, q, c)
If rv < 0 Then
  message = "Alignment Error."
  MsgBox message, MB_OK, "ali_align"
Else
  message = "Best Alignment at (" + Format(x) + ", " + Format(y) + ")"
  MsgBox message, MB_OK, "ali_align"
  message = "Quality = " + Format(q) + " Contrast Ratio = " + Format(c)
  MsgBox message, MB_OK, "ali_align"
End If
```

ali_autoteach : teach patterns automatically

FUNCTION

C Prototype

int WINAPI **ali_autoteach** (int numreq, int patnum, int pattype, int awx, int awy,
int awdx, int awdy, int dx, int dy, int atx, int aty,
int atdx, int atdy);

Visual Basic Declaration

Declare Function **ali_autoteach** Lib "mnalignt.dll" (ByVal numreq As Long,
ByVal patnum As Long, ByVal pattype As Long, ByVal awx As Long,
ByVal awy As Long, ByVal awdx As Long, ByVal awdy As Long,
ByVal dx As Long, ByVal dy As Long, ByVal atx As Long,
ByVal aty As Long, ByVal atdx As Long, ByVal atdy As Long) As Long

where

numreq	=	number of patterns requested to teach. 1 to 16
patnum	=	number of first pattern to teach. 1 to (17 - numreq)
pattype	=	Pattern Type. Must be 0.
awx	=	x location of Alignment Window 0 to (XDIM - awdx)
awy	=	y location of Alignment Window 0 to (YDIM - awdy)
awdx	=	x span of Alignment Window 16 to 512
awdy	=	y span of Alignment Window 16 to 480
dx	=	x span of pattern. 16 to 256
dy	=	y span of pattern. 16 to 256
atx	=	x location of AutoTeach Window awx to (awx + awdx - atdx)
aty	=	y location of AutoTeach Window. awy to (awy + awdy - atdy))

atdx	=	x span of AutoTeach Window. 64 to 512
atdy	=	y span of AutoTeach window. 64 to 480

Return Values:

1 to 16	-	Number of patterns taught
PAT_NUM_ERR	-	Pattern Number out of range
PAT_TOO_BIG	-	Pattern too large for AutoTeach Window
PAT_TYP_ERR	-	Pattern type out of range
PAT_REQ_ERR	-	Number of patterns requested out of range
PAT_X_SPAN_ERR	-	Pattern x span out of range
PAT_Y_SPAN_ERR	-	Pattern y span out of range
NO_VALID_PAT	-	No valid pattern found in AutoTeach Window
WIN_X_ERR	-	Alignment Window x location out of range
WIN_Y_ERR	-	Alignment Window y location out of range
WIN_X_SPAN_ERR	-	Alignment Window x span out of range
WIN_Y_SPAN_ERR	-	Alignment Window y span out of range
PROT_ERR	-	Protection device not installed

DESCRIPTION

This function provides automatic teaching by sampling a large number of patterns from the portion of the video image within the specified AutoTeach Window. The user specifies the size, shape, and number of patterns to be taught. Each sampled pattern is aligned over the specified Alignment Window and a Teach Score, Teach Search Depth, and Next-Best Quality value is generated for that pattern. Sampled patterns are then sorted based on Teach Score first and Teach Search Depth second. Selected patterns are placed in the Pattern Data Base starting with the pattern number specified. If the Alignment System is unable to find a valid number of patterns equal to the number of patterns requested, then the valid patterns available are placed in the Pattern Data Base and the number of patterns actually taught is returned by the function. Attributes of the taught patterns may be examined using the function **ali_pat_attributes**. **ali_autoteach** initializes the associated ORPs to the center of the taught patterns. Note that the AutoTeach Window must lie completely within the Alignment Window and that the pattern size must be smaller than the size of the AutoTeach Window.

EXAMPLES

Automatically teach the four best patterns. Pattern size is to be 91 x 101. The AutoTeach Window is located at (100, 105) and is 200 x 200. Sample patterns will be selected from the area of the AutoTeach Window. The Alignment Window is located at (10, 10) and is 400 x 400. Teach Score, Teach Search Depth, and Next-Best Quality of the sampled patterns will be based on an alignment of the patterns over the Alignment Window. Patterns will placed in the Pattern Data Base starting with Pattern Number 7.

C/C++ EXAMPLE

```
int nl;                         /* number of patterns taught */
int type;                       /* pattern type */
int awx, awy, awdx, awdy;       /* search window */
int x, y, dx, dy;               /* pattern location */
int s;                          /* Teach Score */
int sd;                         /* Teach Search Depth */
int nb;                         /* Next-Best Quality */
char str[80];                   /* string */

...

nl = ali_autoteach (4, 7, 0, 10, 10, 400, 400, 91, 101, 100, 105, 200, 200);

sprintf (str, "Number of Patterns taught = %d\n", nl);
::MessageBox(NULL, str, "ali_autoteach", MB_OK);

ali_pat_attributes (7, &type, &awx, &awy, &awdx, &awdy, &x, &y, &dx, &dy, &s, &sd,
&nb);

sprintf (str, "Best Pattern :  Location = (%d, %d)  Size = %d x %d\n", x, y, dx, dy);
::MessageBox(NULL, str, "ali_autoteach", MB_OK);

sprintf (str, "Teach Score = %d  Teach Search Depth = %d  Next-Best Quality  = %d\n",
        s, sd, nb);
::MessageBox(NULL, str, "ali_autoteach", MB_OK);
```

VISUAL BASIC EXAMPLE

```
Dim nl As Long              ' number of patterns taught
Dim pattype As Long         ' pattern type
Dim awx As Long             ' x search window
Dim awy As Long             ' y search window
Dim awdx As Long            ' dx search window
Dim awdy As Long            ' dy search window
```

```
Dim x As Long              ' x pattern location
Dim y As Long              ' y pattern location
Dim dx As Long             ' dx pattern location
Dim dy As Long             ' dy pattern location
Dim s As Long              ' Teach Score
Dim sd As Long             ' Teach Search Depth
Dim nb As Long             ' Next-Best Quality
Dim message As String      ' string
Dim rv As Long             ' return value

...

nl = ali_autoteach(4, 7, 0, 10, 10, 400, 400, 91, 101, 100, 105, 200, 200)

message = "Number of Patterns taught = " + Format(nl)
MsgBox message, MB_OK, "ali_autoteach"

rv = ali_pat_attributes(7, pattype, awx, awy, awdx, awdy, x, y, dx, dy, s, sd, nb)

message = "Best Pattern :  Location = (" + Format(x) + ", " + Format(y) + ") Size = " +
Format(dx) + " x " + Format(dy)
MsgBox message, MB_OK, "ali_autoteach"

message = "Teach Score = " + Format(s) + " Teach Search Depth = " + Format(sd) + "
Next-Best Quality  =  " + Format(nb)
MsgBox message, MB_OK, "ali_autoteach"
```

ali_calib : calibration

FUNCTION

C Prototype

int WINAPI **ali_calib** (int numpoints, double u[], double v[], double x[],
double y[], double *aptr, double *bptr,
double *cptr, double *dptr, double *fomptr);

Visual Basic Declaration

Declare Function **ali_calib** Lib "mnalignt.dll" (ByVal numpoints As Long,
u As Double, v As Double, x As Double, y As Double, aptr As Double,
bptr As Double, cptr As Double, dptr As Double, fomptr As Double) As Long

where

numpoints	=	number of associated points in two coordinate reference frames (must be 3 or greater)
u[]	=	array of abscissa points in reference system source
v[]	=	array of ordinate points in reference system source
x[]	=	array of abscissa points in reference system destination
y[]	=	array of ordinate points in reference system destination
aptr	=	transformation matrix element (1,1)
bptr	=	transformation matrix element (1,2)
cptr	=	transformation matrix element (2,1)
dptr	=	transformation matrix element (2,2)
fomptr	=	figure of merit for calibration (0 to 100)

Return Values:

NO_ERROR	-	Operation completed without error
TOO_FEW_PNTS	-	Too few points for calibration
ILL_FORM_MAT	-	Ill-formed matrix
PROT_ERR	-	Protection device not installed

DESCRIPTION

This function provides a transformation between two Cartesian reference systems, given that associated points in the two reference systems are available. There are two restrictions on the points that are used to calibrate:

1) There must be at least three points used for calibration.
2) The points may not all be on a straight line. This will cause an ill-conditioned matrix operation.

Other than these restrictions, the points may be placed at any location.

The routine uses a statistical calculation to form the transformation matrix between the two reference frames. The transformation matrix elements {a, b, c, d} ("a" refers to the contents of aptr, "b" to the contents of bptr, etc.) should be used in conjunction with the routines ali_f_xform and ali_r_xform to transform coordinates between the two reference frames.

If the reference system source points are from the frame grabber, and if the reference system destination points are from the real world with a known unit measurement length, then valuable information about system calibration may be derived from the elements {a, b, c, d} as follows:

1) Pixels per unit length for the x axis of the digitizer is given by the following:

```
kx = a/cos(atan2(b,d));
```

2) Pixels per unit length for the y axis of the digitizer is given by the following:

```
ky = d/cos(atan2(b,d));
```

3) Aspect ratio of the horizontal pixel size and the vertical pixel size is given by:

```
ar = sqrt ((a*a + c*c)/(b*b + d*d));
```

4) Angle between the two reference systems is given by:

```
theta = atan2(b,d);
```

Although the above equations provide valuable information about the vision world and the real world, the calibration routine may be used to compute transformations between

any two Cartesian reference systems. The reference systems may be two vision systems, two real world systems, a set of table axes and the real world, a set of table axes and the vision system, or any other combination of reference coordinates.

EXAMPLES

C/C++ EXAMPLE

```
int             numpoints;
double          x[3], y[3];
double          u[3], v[3];
double          a, b, c, d;
double          fom;
double          x_point, y_point, u_point, v_point;
char            str[80];
...

numpoints = 3;
ali_calib (numpoints, u, v, x, y, &a, &b, &c, &d, &fom);

if (fom > 0.9)
   {
   /*  Transform center of the vision reference system to the (x,y) reference system  */
   ali_f_xform (a, b, c, d, 256.0, 240.0, &x_point, &y_point);

   /* Reverse transform the point and check it */
   ali_r_xform (a, b, c, d, x_point, y_point, &u_point, &v_point);

   sprintf (str, "Difference in u value transformation = %lf\n", 256.0 - u_point);
   ::MessageBox(NULL, str, "ali_calib", MB_OK);

   sprintf (str, "Difference in v value transformation = %lf\n", 240.0 - v_point);
   ::MessageBox(NULL, str, "ali_calib", MB_OK);
   }
 else
   {
   sprintf (str, "Figure of merit is less than 0.9\n");
   ::MessageBox(NULL, str, "ali_calib", MB_OK);
   }
```

VISUAL BASIC EXAMPLE

```
Dim numpoints     As Long
Static x(3) As Double, y(3) As Double
Static u(3) As Double, v(3) As Double
Dim a As Double, b As Double, c As Double, d As Double
Dim fom As Double
Dim x_point As Double, y_point As Double, u_point As Double, v_point As Double
Dim message As String
Dim rv As Long

...

numpoints = 3;
rv = ali_calib(numpoints, u(0), v(0), x(0), y(0), a, b, c, d, fom)

If fom > .9 Then
   ' Transform the center of the vision reference system to the (x,y) reference system
   rv = ali_f_xform(a, b, c, d, 256#, 240#, x_point, y_point)

   ' Reverse transform the point and check it
   rv = ali_r_xform(a, b, c, d, x_point, y_point, u_point, v_point)

   message = "Difference in u value transformation = " + Format(256# - u_point)
   MsgBox message, MB_OK, "ali_calib"

   message = "Difference in v value transformation = " + Format(240# - v_point)
   MsgBox message, MB_OK, "ali_calib"
Else
   message = "Figure of merit is less than 0.9"
   MsgBox message, MB_OK, "ali_calib"
End If
```

ali_camera : select video input

FUNCTION

C Prototype

int WINAPI **ali_camera** (int n);

Visual Basic Declaration

Declare Function **ali_camera** Lib "mnalignt.dll" (ByVal n As Long) As Long

where

n	=	number of camera or video source 0 to 3 for MV-1000 (4 video inputs)

Return Values:

NO_ERROR	-	Operation completed without error
PROT_ERR	-	Protection device not installed

DESCRIPTION

This function selects the specified video source as the current input to the vision hardware.

EXAMPLES

Select the camera connected to video source 0.

C/C++ EXAMPLE

```
...
ali_camera (0);
```

VISUAL BASIC EXAMPLE

```
Dim rv As Long

...
rv = ali_camera(0)
```

ali_configure : configure the system

FUNCTION

C Prototype

int WINAPI **ali_configure** (int hWnd, char *fname);

Visual Basic Declaration

Declare Function **ali_configure** Lib "mnalignt.dll" (ByVal hWnd As Long,
ByVal fname As String) As Long

where

hWnd	=	Window Handle
fname	=	name of an Alignment System configuration file

DESCRIPTION

This function configures the alignment software based on the hardware configuration of the frame grabber. The hardware configuration of the system must be specified in a configuration file, typically ALINT.INI. See Part III of this manual for a detailed description of the configuration file.

Note that the two functions **ali_configure** and **ali_initialize** MUST be executed before any other Alignment functions are called.

EXAMPLES

Configure and Initialize an Alignment system. Terminate at the end of the program.

C/C++ EXAMPLE

```
int     iRet;          /* an integer */
char    szStr [80];    /* a string */
...

iRet = ali_configure (hWnd, "alint.ini");
if (iRet == NO_ERROR) {
        ali_initialize ();
} else {
```

```
        sprintf (szStr, “Align error %d”, iRet);
        ::MessageBox ((NULL, str, "ali_configure", MB_OK | MB_ICONSTOP);
}

...

ali_terminate ();
```

VISUAL BASIC EXAMPLE

```
Dim rv As Long
Dim message as string

...

rv = ali_configure(frmMain.hWnd, "alint.ini")
If rv = NO_ERROR Then
        ali_initialize
Else
        message = “Align error “ & str$(rv)
        MsgBox message, MB_OK + MB_ICONSTOP, "ali_configure"
End If


...

rv = ali_terminate()
```

ali_display_initialize : initialize window display of frame buffer

FUNCTION

C Prototype

int WINAPI **ali_display_initialize** (int hWnd);

Visual Basic Declaration

Declare Function **ali_display_initialize** Lib "mnalignt.dll" (ByVal hWnd As Long)
As Long

where

hWnd = Window Handle

The function, **ali_display_initialize**, requires a window handle to create and initialize a palette so that the gray scale image in the frame buffer will be correctly displayed. To insure proper initialization, hWnd should be the handle of the window to receive the frame buffer image. Later, when **ali_display** is called, the frame buffer gray scale will be properly represented.

In Visual Basic, hWnd should be the hWnd property of a Picture Box Control.
Modify the paint event of the Picture Box Control to call **ali_display**.

In C, hWnd can be a handle to any window in which the image display is controlled by the user's code and not left to Windows NT only, i.e., the WM_PAINT and possibly other message handlers need to call **ali_display** in order to display the frame buffer image.

Return Values:

NO_ERROR	-	Operation completed without error
Any other value	-	Error in initialization

DESCRIPTION

This function initializes the system for display of the frame buffer image in a user window. Note that this function needs to be called only once, but must be called prior to the first call to **ali_display**. The user can then call **ali_display** as many times as required.

EXAMPLES

see **ali_display.**

ali_display : display the frame buffer in a window

FUNCTION

C Prototype

int WINAPI **ali_display** (int hWnd);

Visual Basic Declaration

Declare Function **ali_display** Lib "mnalignt.dll" (ByVal hWnd As Long)
As Long

where

hWnd = Window Handle

For a detailed description of the window handle see **ali_display_initialize**.

Return Values:

NO_ERROR	-	Operation completed without error
Any other value	-	Error in display

DESCRIPTION

This function displays the frame buffer in a user window. Note that there is no scaling involved in the display. That is, a 640 x 480 pixel frame buffer will always require a 640 x 480 pixel window for display of the complete image.

Certain combinations of frame buffer resolution and Windows NT screen resolution will cause an aspect ratio distortion in the displayed image. For example, a circle in the frame buffer may appear as an ellipse in the display. In general, 512 x 480 frame buffers will suffer from this distortion while 640 x 480 frame buffers will be displayed without distortion. Specifically -

640 x 480 frame buffer with 640 x 480 Windows NT resolution - no distortion
640 x 480 frame buffer with 800 x 600 Windows NT resolution - no distortion
640 x 480 frame buffer with 1024 x 768 Windows NT resolution - no distortion
512 x 480 frame buffer with 640 x 480 Windows NT resolution - distortion
512 x 480 frame buffer with 800 x 600 Windows NT resolution - distortion
512 x 480 frame buffer with 1024 x 768 Windows NT resolution - distortion

The function **ali_display_initialize** must be called prior to the first call to **ali_display**.

EXAMPLES

Initialize the display. Snap an image into the frame buffer and display it in a user window. Terminate the display.

C/C++ EXAMPLE

```
// The variable m_hWnd is the handle of window which displays the video image.
// It is of type HWND.

ali_display_initialize (m_hWnd);

...

ali_snap();
ali_display(m_hWnd);              /* Display the image just acquired with ali_snap */

...

ali_display_terminate();
```

VISUAL BASIC EXAMPLE

```
' picVideo is the name of the Picture Box Control for frame buffer display

Dim rv As Long  ' return value

...

rv = ali_display_initialize(picVideo.hWnd)

...

rv = ali_snap()
‘ Display the image just acquired with ali_snap
rv = ali_display(picVideo.hWnd)


...

rv = ali_display_terminate()
```

ali_display_terminate : terminate window display of frame buffer

FUNCTION

C Prototype

int WINAPI **ali_display_terminate** (void);

Visual Basic Declaration

Declare Function **ali_display_terminate** Lib "mnalignt.dll" () As Long

Return Values:

NO_ERROR	-	Operation completed without error
PROT_ERR	-	Protection device not installed

DESCRIPTION

This function terminates frame buffer Window display. Note that this function needs to be called only once, but must be called prior to the end of any program in which **ali_display_initialize** is called. This function releases Windows NT resources associated with the display of an image.

EXAMPLE

see **ali_display.**

ali_err_level : set error reporting level

FUNCTION

C Prototype

int WINAPI **ali_err_level** (int level);

Visual Basic Declaration

Declare Function **ali_err_level** Lib "mnalignt.dll" (ByVal level As Long) As Long

where

level = Error reporting level:
- 0 Disable error reporting.
- 2 Enable error reporting (default).
- -1 Return current error level.

DESCRIPTION

This function turns ALIGN error reporting on or off. With reporting enabled, runtime errors are routed to a Message Box. The following information is reported to the Message Box.

The name of the function in which the error occurred
The return code of the error (refer to ali.h)
A message describing the nature of the error

Generally, for ALIGN functions, any error message indicates a condition that could result in unexpected or incorrect alignment results. It is recommended that error reporting always remain enabled.

Return Values:

The current error level.

EXAMPLES

Turn error reporting on.

C/C++ EXAMPLE

```
int iRet;
...

iRet = ali_err_level (2);
```

VISUAL BASIC EXAMPLE

```
Dim rv As Long

...

rv = ali_err_level(2)
```

ali_f_xform : forward transform

FUNCTION

C Prototype

int WINAPI **ali_f_xform** (double a, double b, double c, double d, double u,
double v, double *xptr, double *yptr);

Visual Basic Declaration

Declare Function **ali_f_xform** Lib "mnalignt.dll" (ByVal a As Double,
ByVal b As Double, ByVal c As Double, ByVal d As Double,
ByVal u As Double, ByVal v As Double,
xptr As Double, yptr As Double) As Long

where

u	=	abscissa point in reference system source
v	=	ordinate point in reference system source
xptr	=	abscissa point in reference system destination
yptr	=	ordinate point in reference system destination
a	=	transformation matrix element (1,1)
b	=	transformation matrix element (1,2)
c	=	transformation matrix element (2,1)
d	=	transformation matrix element (2,2)

Return Values:

NO_ERROR	-	Operation completed without error
PROT_ERR	-	Protection device not installed

DESCRIPTION

This function transforms a point between two Cartesian reference systems, given that the transformation matrix elements {a, b, c, d} between the two reference systems are available.

For example, if the reference system source points are from the frame grabber, and if the reference system destination points are from the real world with a known unit measurement length, then the transformation performed is as follows:

```
x = a*u + b*v;
y = c*u + d*v;
```

This is equivalent to the matrix operation:

```
| x |       | a   b |   | u |
|   |   =   |       |   |   |
| y |       |c   d |   | v |
```

EXAMPLES

See **ali_calib.**

ali_fast_align : fast alignment

FUNCTION

C Prototype

int WINAPI **ali_fast_align** (int patnum, int awx, int awy, int awdx, int awdy,
int sd, int est, int nr, int *nfptr, int *xptr,
int *yptr, int *qptr, double *cptr);

Visual Basic Declaration

Declare Function **ali_fast_align** Lib "mnalignt.dll" (ByVal patnum As Long,
ByVal awx As Long, ByVal awy As Long, ByVal awdx As Long,
ByVal awdy As Long, ByVal sd As Long, ByVal est As Long,
ByVal nr As Long, nfptr As Long, xptr As Long, yptr As Long,
qptr As Long, cptr As Double) As Long

where

patnum	=	Pattern number 1 to 16
awx	=	X location of Alignment Window 0 to (XDIM - awdx)
awy	=	Y location of Alignment Window 0 to (YDIM - awdy)
awdx	=	X span of Alignment Window 16 to 512
awdy	=	Y span of Alignment Window 16 to 480
sd	=	Teach Search Depth 1 to 16
est	=	Early-Stop Quality threshold -100 to +100
nr	=	number of alignment locations requested 1 to 64
nfptr	=	pointer to number of alignment locations found
xptr	=	pointer to X location of best alignment
yptr	=	pointer to Y location of best alignment
qptr	=	pointer to quality of best alignment
cptr	=	pointer to contrast ratio of best alignment

Return Values:

NO_ERROR	-	Alignment completed without Early-Stop
1	-	Alignment completed with Early-Stop invoked
PAT_TYP_ERR	-	Pattern type incorrect
PAT_NUM_ERR	-	Pattern Number out of range
PAT_NOT_TAUGHT	-	Pattern not taught
PAT_TOO_BIG	-	Pattern too large for Alignment Window
SRCH_DPTH_ERR	-	Search Depth out of range
EARLY_STP_Q_ERR	-	Early-Stop Quality threshold out of range
WIN_X_SPAN_ERR	-	Alignment Window x span out of range
WIN_Y_SPAN_ERR	-	Alignment Window y span out of range
WIN_X_ERR	-	Alignment Window x location out of range
WIN_Y_ERR	-	Alignment Window y location out of range
PROT_ERR	-	Protection device not installed

NOTE: This function is not available for the MV-1000.

DESCRIPTION

Fast Alignment is appropriate for applications where different patterns are to be aligned on the same image. Fast Alignment will execute about twice as fast as a comparable normal alignment. Use of the function **ali_fast_align** is subject to the following constraints:

- The alignment image must have been preprocessed by the function **ali_setup_fast_align**.
- The pattern type must be the same as that specified by **ali_setup_fast_align**.
- The Alignment Window must lie completely within the Alignment Window specified by **ali_setup_fast_align**.
- The image used by ali_fast_align must be the identical image processed by **ali_setup_fast_align**. (No new grabs or snaps.)

The parameter sequence for calling **ali_fast_align** is identical to that for **ali_align**. From the user's perspective, the two functions perform identical tasks with the exception of execution speed. Use **ali_fast_align** for Fast Alignment of multiple patterns on the same image.

The location, quality, and contrast ratio of the best alignment are available to the caller via four pointers.

EXAMPLES

Setup for Fast Alignment. Perform 3 Fast Alignments using Patterns 1, 2, and 3. Note that the Alignment Windows for the 3 Fast Alignments can be different, but that all lie completely within the Alignment Window specified by **ali_setup_fast_align**. This example does not check return codes. Error checking is left out to better illustrate the **ali_fast_align** functions use.

C/C++ EXAMPLE

```
int x1, y1, q1;          /* pattern 1 alignment results */
int x2, y2, q2;          /* pattern 2 alignment results */
int x3, y3, q3;          /* pattern 3 alignment results */
double c1, c2, c3;       /* contrast ratios */
int sd = 3;              /* Search Depth */
int est = 70;            /* Early-Stop quality threshold */
int nr = 1;              /* number of alignments requested */
int nf;                  /* number of alignments found */
char str[80];            /* string */
...

ali_snap ();
ali_setup_fast_align (0, 40, 40, 400, 400);                    /* setup */
ali_fast_align    (1, 40, 40, 200, 200, sd, est, nr,
                   &nf, &x1, &y1, &q1, &c1);                   /* fast align 1 */
ali_fast_align    (2, 100, 100, 200, 200, sd, est, nr,
                   &nf, &x2, &y2, &q2, &c2);                   /* fast align 2 */
ali_fast_align    (3, 200, 200, 90, 90, sd, est, nr,
                   &nf, &x3, &y3, &q3, &c3);                   /* fast align 3 */

sprintf (str, "Pattern 1 Fast Align at (%d, %d).  Q1 = %d  C1 = %lf\n", x1, y1, q1, c1);
::MessageBox(NULL, str, "ali_fast_align", MB_OK);

sprintf (str, "Pattern 2 Fast Align at (%d, %d).  Q2 = %d  C2 = %lf\n", x2, y2, q2, c2);
::MessageBox(NULL, str, "ali_fast_align", MB_OK);

sprintf (str, "Pattern 3 Fast Align at (%d, %d).  Q3 = %d  C3 = %lf\n", x3, y3, q3, c3);
::MessageBox(NULL, str, "ali_fast_align", MB_OK);
```

VISUAL BASIC EXAMPLE

```
Dim x1 As Long, y1 As Long, q1 As Long        ' pattern 1 alignment results
Dim x2 As Long, y2 As Long, q2 As Long        ' pattern 2 alignment results
Dim x3 As Long, y3 As Long, q3 As Long        ' pattern 3 alignment results
Dim c1 As Double                              ' pattern 1 contrast  ratio
Dim c2 As Double                              ' pattern 2 contrast  ratio
Dim c3 As Double                              ' pattern 3 contrast  ratio
Dim sd As Long                                ' Search Depth
Dim est As Long                               ' Early-Stop quality threshold
Dim nr As Long                                ' number of alignments requested
Dim nf As Long                                ' number of alignments found
Dim message As String                         ' string
Dim rv As Long                                ' return value

...

sd = 3
est = 70
nr = 1

rv = ali_snap()
rv = ali_setup_fast_align(40, 40, 400, 400)                                  ' setup
rv = ali_fast_align(1, 40, 40, 200, 200, sd, est, nr, nf, x1, y1, q1, c1)    ' fast align 1
rv = ali_fast_align(2, 100, 100, 200, 200, sd, est, nr, nf, x2, y2, q2, c2)  ' fast align 2
rv = ali_fast_align(3, 200, 200, 90, 90, sd, est, nr, nf, x3, y3, q3, c3)    ' fast align 3

message = "Pattern 1 Fast Align at (" + Format(x1) + ", " + Format(y1) + "). Q1 = " +
Format(q1) + " C1 = " + Format(c1, "0.00")
MsgBox message, MB_OK, "ali_fast_align"

message = "Pattern 2 Fast Align at (" + Format(x2) + ", " + Format(y2) + "). Q2 = " +
Format(q2) + " C2 = " + Format(c2, " 0.00")
MsgBox message, MB_OK, "ali_fast_align"

message = "Pattern 3 Fast Align at (" + Format(x3) + ", " + Format(y3) + "). Q3 = " +
Format(q3) + " C3 = " + Format(c3, "0.00")
MsgBox message, MB_OK, "ali_fast_align"
```

ali_fast_teach : fast teach a pattern

FUNCTION

C Prototype

int WINAPI **ali_fast_teach** (int patnum, int pattype, int awx, int awy, int awdx,
int awdy, int x, int y, int dx, int dy, int *sptr, int *sdptr, int *nbqptr);

Visual Basic Prototype

Declare Function **ali_fast_teach** Lib "mnalignt.dll" (ByVal patnum As Long,
ByVal pattype As Long, ByVal axw As Long, ByVal awy As Long,
ByVal awdx As Long, ByVal awdy As Long, ByVal x As Long,
ByVal y As Long, ByVal dx As Long, ByVal dy As Long,
sptr As Long, sdptr As Long, nbqptr As Long) As Long

where

patnum	=	Pattern Number 1 to 16
pattype	=	Pattern Type Must be 0.
awx	=	x location of Alignment Window 0 to (XDIM -awdx)
awy	=	y location of Alignment Window 0 to (YDIM - awdy)
awdx	=	x span of Alignment Window 16 to 512
awdy	=	y span of Alignment Window 16 to 480
x	=	x location of pattern upper left corner awx to (awx + awdx - dx)
y	=	y location of pattern upper left corner awy to (awy + awdy - dy)
dx	=	x span of pattern 16 to awdx
dy	=	y span of pattern 16 to awdy
sptr	=	pointer to pattern Teach Score
sdptr	=	pointer to pattern Teach Search Depth
nbqptr	=	pointer to pattern Next-Best Quality

Return Values:

NO_ERROR	-	Operation completed without error
PAT_NUM_ERR	-	Pattern Number out of range
PAT_TYP_ERR	-	Pattern type out of range
PAT_X_ERR	-	Pattern x location out of range
PAT_Y_ERR	-	Pattern y location out of range
PAT_X_SPAN_ERR	-	Pattern x span out of range
PAT_Y_SPAN_ERR	-	Pattern y span out of range
PAT_TOO_BIG	-	Pattern too large for Alignment Window
WIN_X_SPAN_ERR	-	Alignment Window x span out of range
WIN_Y_SPAN_ERR	-	Alignment Window y span out of range
WIN_X_ERR	-	Alignment Window x location out of range
WIN_Y_ERR	-	Alignment Window y location out of range
PROT_ERR	-	Protection device not installed

DESCRIPTION

This function teaches a pattern, without computing Teach Score, Teach Search Depth, and Next-Best Quality. The calling sequence is identical to function **ali_teach**, but **ali_fast_teach** executes much faster. The user specifies the pattern type, location, and size and the location and size of the Alignment Window. Pointers for Teach Score, Teach Search Depth, and Next-Best Quality must be provided but the values for these three parameters are always returned as 0. This function may be useful in cases where the application requires a specific pattern regardless of the Teach Score, or in cases where the user must teach a new pattern for every part cycle.

The following information is placed in the Pattern Data Base:

Pattern type.
Pattern Alignment Window location.
Pattern Alignment Window size.
Pattern location.
Pattern size.
Pattern ORP location.
Pattern Teach Score. Always 0.
Pattern Teach Search Depth. Always 0.
Pattern Next-Best Quality. Always 0.

Note that the pattern must lie completely within the specified Alignment Window. **ali_fast_teach** initializes the associated ORP to the center of the pattern (xorp = *xptr + *dxptr/2, yorp = *yptr + *dyptr/2). All pattern attributes may be read from the Pattern Data Base at any time using the function **ali_pat_attributes** or **ali_getorp**.

EXAMPLES

Fast Teach Pattern 1. Pattern location = (100,100) size = 120 x 120. Alignment Window location = (16,16) size = 400 x 400.

C/C++ EXAMPLE

```
int iRet;       /* integer */
int s;          /* pattern Teach Score */
int sd;         /* pattern Teach Search Depth */
int nbq;        /* pattern Next-Best Quality */
char str[80];   /* string */
...

iRet = ali_fast_teach (1, 0, 16, 16, 400, 400, 100, 100, 120, 120, &s, &sd, &nbq);
if (iRet < NO_ERROR) {
        sprintf (str, “Align error %d”, iRet);
        MessageBox (NULL, str, “ali_fast_teach”, MB_OK | MB_ICONSTOP);
}
```

VISUAL BASIC EXAMPLE

```
Dim s As Long               ' pattern Teach Score
Dim sd As Long              ' pattern Teach Search Depth
Dim nbq As Long             ' pattern Next-Best Quality
Dim message As String       ' string
Dim rv As Long              ' return value

...

rv = ali_fast_teach(1, 0, 16, 16, 400, 400, 100, 100, 120, 120, s, sd, nbq)

If rv < 0 Then
        message = “Align Error = “ + Format (rv)
        MsgBox message, MB_OK, “ali_fast_teach”
End If
```

ali_getorp : get an Operator Recognition Point

FUNCTION

C Prototype

int WINAPI **ali_getorp** (int patnum, int *xoptr, int *yoptr);

Visual Basic Declaration

Declare Function **ali_getorp** Lib "mnalignt.dll" (ByVal patnum As Long,
xoptr As Long, yoptr As Long) As Long

where

patnum	=	Pattern Number 1 to 16
xoptr	=	pointer to x location of ORP
yoptr	=	pointer to y location of ORP

Return Values:

NO_ERROR	-	Operation completed without error
PAT_NUM_ERR	-	Pattern Number out of range
PAT_NOT_TAUGHT	-	Pattern not taught
PROT_ERR	-	Protection device not installed

DESCRIPTION

This function reads the ORP location for the specified pattern from the Pattern Data Base. The x and y coordinates of the ORP are returned to the caller via two pointers.

EXAMPLES

Get the current ORP for Pattern 1.

C/C++ EXAMPLE

```
int iRet;       /* function return value */
int xo, yo;     /* ORP location */
char str[80];   /* string */
...
```

```
iRet = ali_getorp (1, &xo, &yo);
if (iRet == NO_ERROR)
        sprintf(str, "Pattern 1 ORP location = (%d, %d)", xo, yo);
 else
        sprintf(str, "Align Error %d", iRet);
::MessageBox(NULL, str, "ali_getorp", MB_OK);
```

VISUAL BASIC EXAMPLE

```
Dim xo As Long              ' ORP x location
Dim yo As Long              ' ORP y location
Dim message As String       ' string
Dim rv As Long              ' return value

...

rv = ali_getorp(1, xo, yo)

If rv = NO_ERROR Then
        message = "Pattern 1 ORP location = (" + Format(xo) + ", " + Format(yo) + ")"
Else
        message = "Align Error " + Format(rv)
End If
MsgBox message, MB_OK, "ali_getorp"
```

ali_initialize : initialize the alignment system

FUNCTION

C Prototype

void WINAPI **ali_initialize** (void);

Visual Basic Declaration

Declare Sub **ali_initialize** Lib "mnalignt.dll" ()

DESCRIPTION

This function initializes the vision system hardware for alignment. It performs certain register and memory mapped initialization based on the particular vision board and the information contained in the configuration file. (At this time, ALIGN for Windows NT supports only the MV-1000.) Note that the two functions **ali_configure** and **ali_initialize** MUST be executed before any other Alignment functions are called.

EXAMPLES

see **ali_configure**

ali_markpat : mark pattern or alignment location

FUNCTION

C Prototype

int WINAPI **ali_markpat** (int patnum, int x, int y);

Visual Basic Declaration

Declare Function **ali_markpat** Lib "mnalignt.dll" (ByVal patnum As Long,
ByVal x As Long, ByVal y As Long) As Long

where

patnum	=	Pattern Number (1 to 16)
x	=	x coordinate of upper left
y	=	y coordinate of upper left

Return Values:

NO_ERROR	-	Operation completed without error
PAT_NUM_ERR	-	Pattern Number out of range
PAT_NOT_TAUGHT	-	Pattern not taught
PAT_X_ERR	-	Pattern x location out of range
PAT_Y_ERR	-	Pattern y location out of range
PROT_ERR	-	Protection device not installed

DESCRIPTION

This function writes a rectangular overlay onto the video image stored in the frame buffer corresponding to the size and shape of the specified pattern at the specified location. It also writes a small crosshair at the ORP location for the specified pattern. This is useful for showing the location of an alignment. Note that calling this function modifies the stored image. This function is only useful if the **ali_display** function is used to redisplay the acquired image after **ali_markpat** is called.

EXAMPLES

Perform an alignment using Pattern 1. Mark the alignment location in the frame buffer image.

C/C++ EXAMPLE

```
int x, y;       /* alignment location */
int nf, q;      /* alignment results */
double c;       /* contrast ratio */
...

ali_align (1, 10, 10, 200, 200, 3, 70, 1, &nf, &x, &y, &q, &c);
ali_markpat (1, x, y);
```

VISUAL BASIC EXAMPLE

```
Dim x As Long       ' alignment x location
Dim y As Long       ' alignment y location
Dim nf As Long      ' alignment results
Dim q As Long       ' alignment results
Dim c As Double     ' contrast ratio
Dim rv As Long      ' return value

...

rv = ali_align(1, 10, 10, 200, 200, 3, 70, 1, nf, x, y, q, c)
rv = ali_markpat(1, x, y)
```

ali_pat_attributes : get pattern attributes

FUNCTION

C Prototype

```
int WINAPI ali_pat_attributes (int patnum, int  *pattypeptr, int  *awxptr, int  *awyptr,
                               int  *awdxptr, int  *awdyptr,
                               int  *xptr, int  *yptr, int  *dxptr,
                               int  *dyptr, int  *sptr, int  *sdptr,
                               int  *nbqptr);
```

Visual Basic Declaration

```
Declare Function ali_pat_attributes Lib "mnalignt.dll" (ByVal patnum As Long,
        pattypeptr As Long, awxptr As Long, awyptr As Long, awdxptr As Long,
        awdyptr As Long, xptr As Long, yptr As Long, dxptr As Long,
        dyptr As Long, sptr As Long, sdptr As Long, nbqptr As Long) As Long
```

where

patnum	=	Pattern Number 1 to 16
pattypeptr	=	pointer to Pattern Type
awxptr	=	pointer to x location of Alignment Window used for teaching the pattern
awyptr	=	pointer to y location of Alignment Window used for teaching the pattern
awdxptr	=	pointer to x span of the Alignment Window used for teaching the pattern
awdyptr	=	pointer to y span of the Alignment Window used for teaching the pattern
xptr	=	pointer to x coordinate of pattern upper left corner
yptr	=	pointer to y coordinate of pattern upper left corner
dxptr	=	pointer to x span of pattern
dyptr	=	pointer to y span of pattern
sptr	=	pointer to pattern Teach Score
sdptr	=	pointer to pattern Teach Search Depth
nbqptr	=	pointer to pattern Next-Best Quality

Return Values:

NO_ERROR	-	Operation completed without error
PAT_NUM_ERR	-	Pattern number out of range
PAT_NOT_TAUGHT	-	Pattern not taught
PROT_ERR	-	Protection device not installed

DESCRIPTION

This function retrieves the following attributes of a previously taught pattern from the Pattern Data Base.

Pattern type
Location and size of Alignment Window used to teach the pattern.
Pattern location and size.
Pattern Teach Score.
Pattern Teach Search Depth.
Pattern Next-Best Quality.

The information is available to the caller via twelve pointers. Use the function **ali_getorp ()** to read the Pattern ORP location from the Pattern Data Base.

EXAMPLES

Read attributes for Pattern 7 from the Pattern Data Base.

C/C++ EXAMPLE

```
int iRet;                    /* function return value */
int type;                    /* Pattern type */
int awx, awy, awdx, awdy;    /* Pattern Alignment Window */
int x, y, dx, dy;            /* Pattern window */
int s;                       /* Pattern Teach Score */
int sd;                      /* Pattern Teach Search Depth */
int nbq;                     /* Pattern Next-Best Quality */
char str[80];                /* string */
...
```

iRet = **ali_pat_attributes** (7, &type, &awx, &awy, &awdx, &awdy, &x, &y, &dx, &dy, &s, &sd, &nbq);

```
if (iRet == NO_ERROR) {
        sprintf (str, "Pattern 7 : Type %d  Location = (%d, %d)  Size = %d x %d\n", type,
                x, y, dx, dy);
        ::MessageBox(NULL, str, "ali_pat_attributes", MB_OK);
        sprintf (str, "Alignment Window.  Location = (%d, %d)  Size = %d x %d\n", awx,
                awy, awdx, awdy);
        ::MessageBox(NULL, str, "ali_pat_attributes", MB_OK);

        sprintf (str, "Pattern Teach Score = %d\n", s);
        ::MessageBox(NULL, str, "ali_pat_attributes", MB_OK);

        sprintf (str, "Pattern Teach Search Depth = %d\n", sd);
        ::MessageBox(NULL, str, "ali_pat_attributes", MB_OK);

        sprintf (str, "Pattern Next-Best Quality = %d\n", nbq);
        ::MessageBox(NULL, str, "ali_pat_attributes", MB_OK);
} else {
        sprintf (str, “Align Error %d”, iRet);
        ::MessageBox(NULL, str, "ali_pat_attributes", MB_OK | MB_ICONSTOP);
}
```

VISUAL BASIC EXAMPLE

```
Dim type As Long          ' Pattern type
Dim awx As Long           ' Pattern x Alignment Window
Dim awy As Long           ' Pattern y Alignment Window
Dim awdx As Long          ' Pattern dx Alignment Window
Dim awdy As Long          ' Pattern dy Alignment Window
Dim x As Long             ' Pattern x window
Dim y As Long             ' Pattern y window
Dim dx As Long            ' Pattern dx window
Dim dy As Long            ' Pattern dy window
Dim s As Long             ' Pattern Teach Score
Dim sd As Long            ' Pattern Teach Search Depth
Dim nbq As Long           ' Pattern Next-Best Quality
Dim message As String     ' string
Dim rv As Long            ' return value

...
```

```
rv = ali_pat_attributes(7, type, awx, awy, awdx, awdy, x, y, dx, dy, s, sd, nbq)

If rv = NO_ERROR Then
        message = "Pattern 7 : Type " + Format(type) + " Location = (" + Format(x) + ", "
         + Format(y) + ") Size = " + Format(dx) + " x " + Format(dy)
        MsgBox message, MB_OK, "ali_pat_attributes"

        message = "Alignment Window.  Location = (" + Format(awx) + ", " +
         Format(awy) + ") Size = " + Format(awdx) + " x " + Format(awdy)
        MsgBox message, MB_OK, "ali_pat_attributes"

        message = "Pattern Teach Score = " + Format(s)
        MsgBox message, MB_OK, "ali_pat_attributes"

        message = "Pattern Teach Search Depth = " + Format(sd)
        MsgBox message, MB_OK, "ali_pat_attributes"

        message = "Pattern Next-Best Quality = " + Format(nbq)
        MsgBox message, MB_OK, "ali_pat_attributes"
Else
        message = “Align Error “ + str$ (rv)
        MsgBox message, MB_OK + MB_ICONSTOP, "ali_pat_attributes"
End If
```

ali_r_xform : reverse transform

FUNCTION

C Prototype

int WINAPI **ali_r_xform** (double a, double b, double c, double d,
double x, double y, double *uptr, double *vptr);

Visual Basic Declaration

Declare Function **ali_r_xform** Lib "mnalignt.dll" (ByVal a As Double,
ByVal b As Double, ByVal c As Double, ByVal d As Double,
ByVal x As Double, ByVal y As Double,
uptr As Double, vptr As Double) As Long

where

uptr	=	abscissa point in reference system source
vptr	=	ordinate point in reference system source
x	=	abscissa point in reference system destination
y	=	ordinate point in reference system destination
a	=	transformation matrix element (1,1)
b	=	transformation matrix element (1,2)
c	=	transformation matrix element (2,1)
d	=	transformation matrix element (2,2)

Return Values:

NO_ERROR	-	Operation completed without error
ILL_FORM_MAT	-	Ill-formed matrix
PROT_ERR	-	Protection device not installed

DESCRIPTION

This function reverse transforms a point between two Cartesian reference systems, given that the transformation matrix elements {a, b, c, d} between the two reference systems are available.

For example, if the reference system source points are from the frame grabber, and if the reference system destination points are from the real world with a known unit measurement length, then the transformation performed is as follows:

```
u     =     ( d*x - b*y)/det;

v     =     (-c*x + a*y)/det;

det   =     a*d - c*b;
```

This is equivalent to the matrix operation:

```
| u |            | a   b |       | x |
|   |  =  inv{   |       |   }   |   |
| v |            | c   d |       | y |
```

This reverse transformation does not take into account the offset between the two coordinate systems. These offsets must be added in by the programmer.

EXAMPLES

See **ali_calib**.

ali_read_bmp_image : read BMP image from disk

FUNCTION

C Prototype

int WINAPI **ali_read_bmp_image** (char *fname);

Visual Basic Declaration

Declare Function **ali_read_bmp_image** Lib "mnalignt.dll" (ByVal fname As String) As Long

where

fname = pointer to string containing disk file name

Return Values:

NO_ERROR	-	Operation completed without error
FILE_ERROR	-	Unable to access file
FORMAT_ERROR	-	Incompatible file type

DESCRIPTION

This function restores a frame buffer image from the specified disk file. The image to be restored must be in BMP format. Note that this function will overwrite and destroy the current contents of the frame buffer.

EXAMPLES

Read a BMP format image from disk file, "IMAGE1.BMP" and display.

C/C++ EXAMPLE

```
...

ali_read_bmp_image ("IMAGE1.BMP");
```

VISUAL BASIC EXAMPLE

```
Dim rv As Long

...

rv = ali_read_bmp_image("IMAGE1.BMP")
```

ali_read_image : read an ALIGN image from disk

FUNCTION

C Prototype

int WINAPI **ali_read_image** (char *fname);

Visual Basic Declaration

Declare Function **ali_read_image** Lib "mnalignt.dll" (ByVal fname As String) As Long

where

fname = pointer to string containing disk file name

Return Values:

NO_ERROR	-	Operation completed without error
FILE_ERROR	-	Unable to access file
FORMAT_ERROR	-	Incompatible file type

DESCRIPTION

This function restores a frame buffer image from the specified disk file. The image to be restored must be in ALIGN format. Note that this function will overwrite and destroy the current contents of the frame buffer.

EXAMPLES

Read an ALIGN format image from disk file, "IMAGE.IMG" and display.

C/C++ EXAMPLE

```
...

ali_read_image ("IMAGE.IMG");
```

VISUAL BASIC EXAMPLE

```
Dim rv As Long

...

rv = ali_read_image("IMAGE. IMG")
```

ali_read_pat : read a pattern file from disk

FUNCTION

C Prototype

int WINAPI **ali_read_pat** (int pn, char *fname);

Visual Basic Declaration

Declare Function **ali_read_pat** Lib "mnalignt.dll" (ByVal patnum As Long, ByVal fname As String) As Long

where

pn	=	Pattern Number 0 = read a file containing 16 patterns into the Pattern Data Base 1 to 16 = read a file containing a single pattern into the specified Pattern Data Base slot.
fname	=	pointer to string containing disk file name

Return Values:

NO_ERROR	-	Operation completed without error
FILE_ERROR	-	File error
FORMAT_ERROR	-	File of incompatible type
PAT_NUM_ERR	-	Pattern Number out of range
NOT_ALL_FILE	-	Not an "all patterns" file
NOT_ONE_FILE	-	Not a "one pattern" file
PROT_ERR	-	Protection device not installed

DESCRIPTION

This function provides a mechanism for reading a full set of patterns or a single pattern from a disk file and placing the patterns or pattern into the Pattern Data Base. Files read by **ali_read_pat** must have been written by **ali_write_pat**.

EXAMPLES

Read a pattern originally taught as Pattern 2 from Single Pattern File "PAT2.PAT" and restore to the Pattern Data Base as Pattern 4.

C/C++ EXAMPLE

```
int      rv;            /* return value */
char     str[80];       /* string */
...

rv = ali_read_pat (4, "PAT2.PAT");
if (rv == NO_ERROR)
  {
  sprintf(str, "File read successfully.\n");
  ::MessageBox(NULL, str, "ali_read_pat", MB_OK);
  }
```

VISUAL BASIC EXAMPLE

```
Dim rv As Long              ' return value
Dim message As String       ' string

...

rv = ali_read_pat(4, "PAT2.PAT")
If rv = NO_ERROR Then
    message = "File read successfully."
    MsgBox message, MB_OK, "ali_read_pat"
End If
```

ali_read_tiff_image : read a TIFF image from disk

FUNCTION

C Prototype

int WINAPI **ali_read_tiff_image** (char *fname);

Visual Basic Declaration

Declare Function **ali_read_tiff_image** Lib "mnalignt.dll" (ByVal fname As String)
As Long

where

fname = pointer to string containing disk file name

Return Values:

NO_ERROR	-	Operation completed without error
FILE_ERROR	-	Unable to access file
FORMAT_ERROR	-	Incompatible file type

DESCRIPTION

This function restores a frame buffer image from the specified disk file. The image to be restored must be in TIFF format. Note that this function will overwrite and destroy the current contents of the frame buffer.

EXAMPLES

Read a TIFF format image from disk file, "IMAGE1.TIF" and display.

C/C++ EXAMPLE

```
...

ali_read_tiff_image ("IMAGE1.TIF");
```

VISUAL BASIC EXAMPLE

```
Dim rv As Long

...

rv = ali_read_tiff_image("IMAGE1.TIF")
```

ali_set_aoi : set an AOI interactively

FUNCTION

C Prototype

int WINAPI **ali_set_aoi** (int hWnd, double xscale, double yscale, long color,
RECT *lpAoi, RECT *lpBoundary,
RECT *lpMin, RECT *lpMax);

Visual Basic Declaration

Declare Function **ali_set_aoi** Lib "mnaoiset.dll" (ByVal hWnd As Long,
ByVal xscale As Double, ByVal yscale As Double, ByVal color As Long,
lpAoi As RECT, lpBoundary As RECT, lpMin As RECT, lpMax As RECT)
As Long

where

hWnd	=	handle to the window where the AOI is to be displayed
xscale	=	the x scale factor. MUST be set to 1.0
yscale	=	the y scale factor. MUST be set to 1.0
color	=	the color of the AOI in RGB format
lpAOI	=	a rectangular area which defines the AOI to be set. This parameter may be used to initialize the AOI when the function is called and to read the location and size of the AOI when the function returns.
lpBoundary	=	a rectangular area which limits the excursion of the AOI to be set. The AOI to be set is restricted to lie inside of this area and may not move outside of its boundaries.
lpMin	=	a rectangular area which defines the minimum size of the AOI to be set. The location of this area is ignored. Only the size is used.
lpMax	=	a rectangular area which defines the maximum size of the AOI to be set. The location of the area is ignored. Only the size is used.

The hWnd parameter is a handle to the window used to display the AOI. This would generally be the same window that is being used to display the image from the frame buffer. For a detailed description of this window handle see **ali_display_initialize**.

RECT is a structure defining a rectangular area. The structure contains 4 integer elements - left, top, right, and bottom. In Visual Basic, this data type is defined in WIN32API.BAS. In C/C++ this structure is defined in WINDOWS.H.

Return Values:

NO_ERROR - Operation completed without error

DESCRIPTION

This function provides the capability to set an AOI by interactively adjusting the size and location of a rectangular overlay on the frame buffer image. It will typically be used to set one or more of the three ALIGN Windows - the Alignment Window, the Pattern Window, or the AutoTeach Window. The AOI will generally be displayed as a colored rectangular overlay, appearing in the same window as that used by **ali_display** to display the frame buffer image.

To set the location of the AOI, point to the interior of the AOI and depress and hold the left mouse button. Use the mouse to adjust the position of the AOI and release the button when the AOI is in the desired location.

To set the size of the AOI, use one of the sizing handles on the perimeter of the rectangle. The sizing handles in the center of each side are used to adjust that side only. The sizing handles at the 4 corners are used to adjust the two adjacent sides simultaneously. Point to a sizing handle and depress and hold the left mouse button. Use the mouse to adjust the size and release the button when the proper size has been achieved. Note that the location and size of the AOI can be monitored during the adjustment process by using the function, **ali_set_aoi_text**.

The structure lpBoundary can be used to limit the excursion of the AOI. The function will not allow the AOI to be moved outside of the boundary specified by this rectangular area. The structures lpMin and lpMax can be used to limit the minimum and maximum size of the AOI. The function will not allow the AOI to be set smaller than the size specified by lpMin, or larger than the size specified by lpMax.

Note that this function is located in MNAOISET.DLL.

EXAMPLES

Set the Pattern Window interactively, prior to teaching a pattern. Confine the Pattern Window to lie inside of the Alignment Window. Limit the minimum size of the Pattern Window to 16 x 16 and the maximum size to 256 x 256. Use the Pattern Window to teach Pattern 1.

In Visual Basic, use the Text Box Control, txtAOI to display the AOI text. Use the Picture Box Control, picVideo, to display the AOI.

In C/C++, txtAOI.m_hWnd is the handle to the window used to display the AOI text. The handle to the window used to display the AOI is picVideo.m_hWnd.

C/C++ EXAMPLE

```
int         rv;         // return value
double      xscale;     // x scale factor. Must be 1.
double      yscale;     // y scale factor. Must be 1.
long        color;      // color of the AOI
RECT        lpAoi;      // AOI location and size
RECT        lpBoundary; // Boundary location and size
RECT        lpMin;      // Minimum size of AOI
RECT        lpMax;      // Maximum size of AOI
int         aw_x;       // x location of Alignment Window
int         aw_y;       // y location of Alignment Window
int         aw_dx;      // x size of Alignment Window
int         aw_dy;      // y size of Alignment Window
int         pw_x;       // x location of Pattern Window
int         pw_y;       // y location of Pattern Window
int         pw_dx;      // x size of Pattern Window
int         pw_dy;      // y size of Pattern Window
int         ts;         // Teach Score
int         tsd;        // Teach Search Depth
int         tnb;        // Next-Best Quality

...

// Set scale to required values
xscale = 1.0;
yscale = 1.0;

// Set the color of the AOI to green
color = RGB(0, 255, 0);

// Set the Alignment Window to size 300 x 300  located at (64, 64)
aw_x = 64;
aw_y = 64;
aw_dx = 300;
aw_dy = 300;

// Initialize the Pattern Window to size 128 x 128.
pw_dx = 128;
pw_dy = 128;

// Initialize the Pattern Window to be centered in the Alignment Window
pw_x = aw_x + (aw_dx - pw_dx) / 2;
pw_y = aw_y + (aw_dy - pw_dy) / 2;
```

```
// Set the initial Pattern Window size and location
lpAoi.left = pw_x;
lpAoi.top = pw_y;
lpAoi.right = pw_x + pw_dx - 1;
lpAoi.bottom = pw_y + pw_dy - 1;

// Set the boundary to be the Alignment Window
lpBoundary.left = aw_x;
lpBoundary.top = aw_y;
lpBoundary.right = aw_x + aw_dx - 1;
lpBoundary.bottom =aw_y + aw_dy - 1;

// Set the maximum size of the Pattern Window to 256 x 256
lpMax.left = 0;
lpMax.top = 0;
lpMax.right = 255;
lpMax.bottom = 255;

// Set the minimum size of the Pattern window to 16 x 16
lpMin.left = 0;
lpMin.top = 0;
lpMin.right = 15;
lpMin.bottom = 15;

// Set the text window
// Assume txtAOI is the name of the Text Box Control used to display the AOI text
// Use black background and green text.
ali_set_aoi_text(txtAOI.m_hWnd, RGB(0, 0, 0), RGB(0, 255, 0));

// Set the Pattern Window
// Assume picVideo is the name of the Picture Box Control for image display
rv = ali_set_aoi(picVideo.m_hWnd, xscale, yscale, color, &lpAoi, &lpBoundary, &lpMin,
&lpMax);

// Convert from RECT to ALIGN format
pw_x = lpAoi.left;
pw_y = lpAoi.top;
pw_dx = lpAoi.right - lpAoi.left + 1;
pw_dy = lpAoi.bottom - lpAoi.top + 1;

// Teach the pattern defined by the Pattern Window
rv = ali_teach(1, 0, aw_x, aw_y, aw_dx, aw_dy, pw_x, pw_y, pw_dx, pw_dy, &ts, &tsd,
&tnb);
```

VISUAL BASIC EXAMPLE

```
Dim rv As Long                  ' return value
Dim xscale As Double            ' x scale factor. Must be 1.
Dim yscale As Double            ' y scale factor. Must be 1.
Dim color As Long               ' color of the AOI
Dim lpAoi As RECT               ' AOI location and size
Dim lpBoundary As RECT          ' Boundary location and size
Dim lpMin As RECT               ' Minimum size of AOI
Dim lpMax As RECT               ' Maximum size of AOI
Dim aw_x  As Long               ' x location of Alignment Window
Dim aw_y As Long                ' y location of Alignment Window
Dim aw_dx As Long               ' x size of Alignment Window
Dim aw_dy As Long               ' y size of Alignment Window
Dim pw_x  As Long               ' x location of Pattern Window
Dim pw_y As Long                ' y location of Pattern Window
Dim pw_dx As Long               ' x size of Pattern Window
Dim pw_dy As Long               ' y size of Pattern Window
Dim ts As Long                  ' Teach Score
Dim tsd As Long                 ' Teach Search Depth
Dim tnb As Long                 ' Next-Best Quality

' Set scale to required values
xscale = 1#
yscale = 1#

' Set the color of the AOI to green
color = RGB(0, 255, 0)

' Set the Alignment Window to size 300 x 300  located at (64, 64)
aw_x = 64
aw_y = 64
aw_dx = 300
aw_dy = 300

' Initialize the Pattern Window to size 128 x 128.
pw_dx = 128
pw_dy = 128

' Initialize the Pattern Window to be centered in the Alignment Window
pw_x = aw_x + (aw_dx - pw_dx) / 2
pw_y = aw_y + (aw_dy - pw_dy) / 2
```

```
' Set the initial Pattern Window size and location
lpAoi.left = pw_x
lpAoi.top = pw_y
lpAoi.right = pw_x + pw_dx - 1
lpAoi.bottom = pw_y + pw_dy - 1

' Set the boundary to be the Alignment Window
lpBoundary.left = aw_x
lpBoundary.top = aw_y
lpBoundary.right = aw_x + aw_dx - 1
lpBoundary.bottom = aw_y + aw_dy - 1

' Set the maximum size of the Pattern Window to 256 x 256
lpMax.left = 0
lpMax.top = 0
lpMax.right = 255
lpMax.bottom = 255

' Set the minimum size of the Pattern window to 16 x 16
lpMin.left = 0
lpMin.top = 0
lpMin.right = 15
lpMin.bottom = 15

' Set the text window
' Assume txtAOI is the name of the Text Box Control used to display AOI text
' Use black background and green text
rv = ali_set_aoi_text(txtAOI.hWnd, RGB(0, 0, 0), RGB(0, 255, 0))

' Set the Pattern Window
' Assume picVideo is the name of the Picture Box Control for image display
rv = ali_set_aoi(picVideo.hWnd, xscale, yscale, color, lpAoi, lpBoundary, lpMin, lpMax)

' Convert from RECT to ALIGN format
pw_x = lpAoi.left
pw_y = lpAoi.top
pw_dx = lpAoi.right - lpAoi.left + 1
pw_dy = lpAoi.bottom - lpAoi.top + 1

' Teach the pattern defined by the Pattern Window
rv = ali_teach(1, 0, aw_x, aw_y, aw_dx, aw_dy, pw_x, pw_y, pw_dx, pw_dy, ts, tsd, tnb)
```

ali_set_aoi_text : enable display of AOI text

FUNCTION

C Prototype

int WINAPI **ali_set_aoi_text** (int hWnd, long backcolor, long forecolor);

Visual Basic Declaration

Declare Function **ali_set_aoi_text** Lib "mnaoiset.dll" (ByVal hWnd As Long,
ByVal backcolor As Long, ByVal forecolor As Long) As Long

where

hWnd	=	handle to window where the AOI text is to be displayed (to enable text) or NULL (to disable the text display)
backcolor	=	the background color of the text window in RGB format
forecolor	=	the text color in RGB format

The hWnd parameter is a handle to the window used to display the AOI text. In Visual Basic this would generally be the handle property of a Text Box Control.

Return Values:

NO_ERROR - Operation completed without error

DESCRIPTION

This function is used in combination with the function **ali_set_aoi**. It enables the display of text indicating the location and size of the AOI while the AOI is being adjusted by **ali_set_aoi**. This allows the user to set the AOI to a precise location and size. The function **ali_set_aoi_text** must be called prior to **aoi_set_aoi** in order to enable text display. The text will be centered in the specified window. The background color of the window and the color of the text may be specified. The size and font of the text are not adjustable. To disable the AOI text display, call **ali_set_aoi_text** with a NULL window handle.

Note that this function is located in MNAOISET.DLL.

EXAMPLES

see **ali_set_aoi.**

ali_setorp : set an Operator Recognition Point

FUNCTION

C Prototype

int WINAPI **ali_setorp** (int patnum, int x, int y);

Visual Basic Declaration

Declare Function **ali_setorp** Lib "mnalignt.dll" (ByVal patnum As Long,
ByVal x As Long, ByVal y As Long) As Long

where

patnum	=	Pattern Number 1 to 16
x	=	x location of ORP 0 to (XDIM - 1)
y	=	y location of ORP 0 to (YDIM - 1)

Return Values:

NO_ERROR	-	Operation completed without error
PAT_NUM_ERR	-	Pattern Number out of range
PAT_NOT_TAUGHT	-	Pattern not taught
ORP_X_ERR	-	ORP x location out of range
ORP_Y_ERR	-	ORP y location out of range
PROT_ERR	-	Protection device not installed

DESCRIPTION

This function sets the Operator Recognition Point for the specified pattern to the specified location. The ORP should be taught on the same image that was used to teach the associated pattern. Note that teaching of the associated pattern initializes the ORP to the center of the pattern.

EXAMPLES

Set the ORP for Pattern 1 to (120, 130).

C/C++ EXAMPLE

```
...

ali_setorp (1, 120, 130);
```

VISUAL BASIC EXAMPLE

```
Dim rv As Long        ' return value

...

rv = ali_setorp(1, 120, 130)
```

ali_setup_fast_align : setup for Fast Alignment

FUNCTION

C Prototype

int WINAPI **ali_setup_fast_align** (int pattype, int awx, int awy, int awdx, int awdy);

Visual Basic Declaration

Declare Function **ali_setup_fast_align** Lib "mnalignt.dll" (ByVal pattype As Long,
ByVal awx As Long, ByVal awy As Long, ByVal awdx As Long,
ByVal awdy As Long) As Long

where

pattype	=	Pattern type Must be 0.
awx	=	X location of Alignment Window 0 to (XDIM - awdx)
awy	=	Y location of Alignment Window 0 to (YDIM - awdy)
awdx	=	X span of Alignment Window 16 to 512
awdy	=	Y span of Alignment Window 16 to 480

Return Values:

NO_ERROR	-	Operation completed without error
PAT_TYP_ERR	-	Pattern type out of range
WIN_X_ERR	-	X location of Alignment Window out of range
WIN_Y_ERR	-	Y location of Alignment Window out of range
WIN_X_SPAN_ERR	-	X span of Alignment Window out of range
WIN_Y_SPAN_ERR	-	Y span of Alignment Window out of range

NOTE: This function is not available for the MV-1000.

DESCRIPTION

This function prepares for Fast Alignment by preprocessing the image within the specified Alignment Window. Fast Alignment may be used in any alignment application

where different patterns are to be aligned on the same image. Use the function **ali_fast_align** to perform the actual Fast Alignment.

Subsequent Fast Alignments will be subject to the following constraints:

- The pattern selected for Fast Alignment must be of the same type as that specified by **ali_setup_fast_align**.
- The Alignment Window specified for Fast Alignment must lie completely within the Alignment Window specified by **ali_setup_fast_align**.
- The image used for Fast Alignment must be the same as the image used for **ali_setup_fast_align**. (No new grabs or snaps.)

After the image has been preprocessed by **ali_setup_fast_align**, any number of Fast Alignments may be performed as long as they satisfy the specified constraints. Any image acquisition or image modification will require another call to **ali_setup_fast_align** before Fast Alignments can be performed on the new image.

EXAMPLES

See **ali_fast_align**.

ali_snap : snap an image

FUNCTION

C Prototype

int WINAPI **ali_snap (**void**);**

Visual Basic Declaration

Declare Function **ali_snap** Lib "mnalignt.dll" () As Long

Return Values:

NO_ERROR	-	Operation completed without error
PROT_ERR	-	Protection device not installed

DESCRIPTION

This function digitizes an image from the video source in a form compatible with the Alignment System.

EXAMPLES

Snap an image and perform a Pattern 1 alignment.

C/C++ EXAMPLE

```
int nf;        /* number of alignment locations found */
int x, y;      /* alignment location */
int q;         /* alignment quality */
double c;      /* alignment contrast ratio */
...

ali_snap ();
ali_align (1, 10, 10, 400, 400, 3, 80, 1, &nf, &x, &y, &q, &c);
```

VISUAL BASIC EXAMPLE

```
Dim nf As Long      ' number of alignment locations found
Dim x As Long       ' x alignment location
Dim y As Long       ' y alignment location
Dim q As Long       ' alignment quality
Dim c As Double     ' alignment contrast ratio
Dim rv As Long      ' return value

...

rv = ali_snap()
rv = ali_align(1, 10, 10, 400, 400, 3, 80, 1, nf, x, y, q, c)
```

ali_sublign : subpixel alignment

FUNCTION

C Prototype

int WINAPI **ali_sublign** (int patnum, int x, int y, int res,
double *xsptr, double *ysptr, double *qsptr);

Visual Basic Declaration

Declare Function **ali_sublign** Lib "mnalignt.dll" (ByVal patnum As Long,
ByVal x As Long, ByVal y As Long, ByVal res As Long,
xsptr As Double, ysptr As Double, csptr As Double) As Long

where

patnum	=	Pattern number 1 to 16
x	=	X location of Alignment 1 to (XDIM - pattern x span)
y	=	Y location of Alignment 1 to (YDIM - pattern y span)
res	=	subpixel resolution 2 = 1/2 pixel 4 = 1/4 pixel 8 = 1/8 pixel 16 = 1/16 pixel 32 = 1/32 pixel
xsptr	=	pointer to X location of best subpixel alignment
ysptr	=	pointer to Y location of best subpixel alignment
qsptr	=	pointer to quality of best subpixel alignment

Return Values:

NO_ERROR	-	Operation completed without error
PAT_NUM_ERR	-	Pattern Number out of range
PAT_NOT_TAUGHT	-	Pattern not taught
PAT_X_ERR	-	Alignment x location out of range
PAT_Y_ERR	-	Alignment y location out of range
RESOLUTION_ERR	-	Subpixel resolution out of range
PROT_ERR	-	Protection device not installed

DESCRIPTION

This function performs subpixel alignment by searching a plus-or-minus one pixel area of the image centered on the specified alignment location. The best subpixel alignment is determined to the specified subpixel alignment resolution. Subpixel alignment resolution may range from 1/2 to 1/32 pixel. Higher subpixel resolutions require longer function execution times. The location and quality of the best subpixel alignment are available to the calling function via three pointers.

EXAMPLES

Align Pattern Number 7 over an Alignment Window located at (60, 70) of x size 130 and y size 110. Use a Search Depth of 3 and an Early-Stop Quality Threshold of 70. Check return value. Perform a subpixel alignment of Pattern 7 to a resolution of 1/16 pixel.

C/C++ EXAMPLE

```
int rv;             /* alignment return value */
int nf;             /* number of alignment locations found */
int x, y;           /* alignment location */
int q;              /* alignment quality */
double c;           /* alignment contrast ratio */
double xs, ys;      /* subpixel alignment location */
double qs;          /* subpixel alignment quality */
char str[80];       /* string */
...

/* Perform alignment. */
rv = ali_align (7, 60, 70, 130, 110, 3, 70, 1, &nf, &x, &y, &q, &c);

sprintf (str, "Best Alignment at (%d, %d).\n", x, y);
::MessageBox(NULL, str, "ali_sublign", MB_OK);

sprintf (str, "Quality = %d  Contrast Ratio = %lf\n", q, c);
::MessageBox(NULL, str, "ali_sublign", MB_OK);

/* Perform subpixel alignment */
rv = ali_sublign (7, x, y, 16, &xs, &ys, &qs);

sprintf (str, "Best Subpixel Alignment at (%lf, %lf).\n", xs, ys);
::MessageBox(NULL, str, "ali_sublign", MB_OK);
```

```
sprintf (str, "Subpixel Quality = %lf\n", qs);
::MessageBox(NULL, str, "ali_sublign", MB_OK);
```

VISUAL BASIC EXAMPLE

```
Dim rv As Long              ' alignment return value
Dim nf As Long              ' number of alignment locations found
Dim x As Long               ' alignment x location
Dim y As Long               ' alignment y location
Dim q As Long               ' alignment quality
Dim c As Double             ' alignment contrast ratio
Dim xs As Double            ' subpixel alignment x location
Dim ys As Double            ' subpixel alignment y location
Dim qs As Double            ' subpixel alignment quality
Dim message As String       ' string

...

' Perform alignment.
rv = ali_align(7, 60, 70, 130, 110, 3, 70, 1, nf, x, y, q, c)

message = "Best Alignment at (" + Format(x) + ", " + Format(y) + ")"
MsgBox message, MB_OK, "ali_sublign"

message = "Quality = " + Format(q) + " Contrast Ratio = " + Format(c, "0.00")
MsgBox message, MB_OK, "ali_sublign"

' Perform subpixel alignment
rv = ali_sublign(7, x, y, 16, xs, ys, qs)

message = "Best Subpixel Alignment at (" + Format(xs, "0.00") + ", " + Format(ys,
"0.00") + ")."
MsgBox message, MB_OK, "ali_sublign"

message = "Subpixel Quality = " + Format(qs, "0.00")
MsgBox message, MB_OK, "ali_sublign"
```

ali_teach : teach a pattern

FUNCTION

C Prototype

int WINAPI **ali_teach** (int patnum, int pattype, int awx, int awy, int awdx, int awdy,
int x, int y, int dx, int dy,
int *sptr, int *sdptr, int *nbqptr);

Visual Basic Declaration

Declare Function **ali_teach** Lib "mnalignt.dll" (ByVal patnum As Long,
ByVal pattype As Long, ByVal awx As Long, ByVal awy As Long,
ByVal awdx As Long, ByVal awdy As Long, ByVal x As Long,
ByVal y As Long, ByVal dx As Long, ByVal dy As Long,
sptr As Long, sdptr As Long, nbqptr As Long) As Long

where

patnum	=	Pattern Number 1 to 16
pattype	=	Pattern Type Must be 0.
awx	=	x location of Alignment Window 0 to (XDIM -awdx)
awy	=	y location of Alignment Window 0 to (YDIM - awdy)
awdx	=	x span of Alignment Window 16 to 512
awdy	=	y span of Alignment Window 16 to 480
x	=	x location of pattern upper left corner awx to (awx + awdx - dx)
y	=	y location of pattern upper left corner awy to (awy + awdy - dy) dx = x span of pattern 16 to awdx dy = y span of pattern 16 to awdy
sptr	=	pointer to pattern Teach Score
sdptr	=	pointer to pattern Teach Search Depth
nbqptr	=	pointer to pattern Next-Best Quality

Return Values:

NO_ERROR	-	Operation completed without error
PAT_NUM_ERR	-	Pattern Number out of range
PAT_TYP_ERR	-	Pattern type out of range
PAT_X_ERR	-	Pattern x location out of range
PAT_Y_ERR	-	Pattern y location out of range
PAT_X_SPAN_ERR	-	Pattern x span out of range
PAT_Y_SPAN_ERR	-	Pattern y span out of range
PAT_TOO_BIG	-	Pattern too large for Alignment Window
WIN_X_SPAN_ERR	-	Alignment Window x span out of range
WIN_Y_SPAN_ERR	-	Alignment Window y span out of range
WIN_X_ERR	-	Alignment Window x location out of range
WIN_Y_ERR	-	Alignment Window y location out of range
PROT_ERR	-	Protection device not installed

DESCRIPTION

This function teaches a pattern. The user specifies the pattern type, location, and size and the location and size of the Alignment Window. The pattern Teach Score, Teach Search Depth, and Next-Best Quality are based on an alignment of the pattern over the specified Alignment Window. These three values are available via three pointers.

The following information is placed in the Pattern Data Base:

Pattern type.
Pattern Alignment Window location.
Pattern Alignment Window size.
Pattern location.
Pattern size.
Pattern ORP location.
Pattern Teach Score.
Pattern Teach Search Depth.
Pattern Next-Best Quality.

Note that the pattern must lie completely within the specified Alignment Window. **ali_teach** initializes the associated ORP to the center of the pattern (xorp = *xptr + *dxptr/2, yorp = *yptr + *dyptr/2). All pattern attributes may be read from the Pattern Data Base at any time using the function **ali_pat_attributes** or **ali_getorp**.

EXAMPLES

Teach Pattern 1. Pattern location = (100,100) size = 120 x 120. Alignment Window location = (16,16) size = 400 x 400.

C/C++ EXAMPLE

```
int iRet;       /* Function return value */
int s;          /* pattern Teach Score */
int sd;         /* pattern Teach Search Depth */
int nbq;        /* pattern Next-Best Quality */
char str[80];   /* string */
...

iRet = ali_teach (1, 0, 16, 16, 400, 400, 100, 100, 120, 120, &s, &sd, &nbq);

if (iRet == NO_ERROR) {
        sprintf (str, "Pattern 1 Teach Score = %d\n", s);
        ::MessageBox(NULL, str, "ali_teach", MB_OK);

        sprintf (str, "Pattern 1 Teach Search Depth = %d\n", sd);
        ::MessageBox(NULL, str, "ali_teach", MB_OK);

        sprintf (str,  "Pattern 1 Next-Best Quality = %d\n", nbq);
        ::MessageBox(NULL, str, "ali_teach", MB_OK);
} else {
        sprintf (str, “Align Error %d”, iRet);
        ::MessageBox(NULL, str, "ali_teach", MB_OK | MB_ICONSTOP);
}
```

VISUAL BASIC EXAMPLE

```
Dim s As Long               ' pattern Teach Score
Dim sd As Long              ' pattern Teach Search Depth
Dim nbq As Long             ' pattern Next-Best Quality
Dim message As String       ' string
Dim rv As Long              ' return value

...

rv = ali_teach(1, 0, 16, 16, 400, 400, 100, 100, 120, 120, s, sd, nbq)
If rv =  NO_ERROR Then
        message = "Pattern 1 Teach Score = " + Format(s)
        MsgBox message, MB_OK, "ali_teach"

        message = "Pattern 1 Teach Search Depth = " + Format(sd)
        MsgBox message, MB_OK, "ali_teach"

        message = "Pattern 1 Next-Best Quality = " + Format(nbq)
        MsgBox message, MB_OK, "ali_teach"
Else
        message = "Align Error " + Format(rv)
        MsgBox message, MB_OK + MB_ICONSTOP, "ali_teach"
End If
```

ali_terminate : terminate an alignment program

FUNCTION

C Prototype

int WINAPI **ali_terminate** (void);

Visual Basic Declaration

Declare Function **ali_terminate** Lib "mnalignt.dll" () As Long

Return Values:

NO_ERROR	-	Operation completed without error
Any other value	-	Error in termination

DESCRIPTION

This function terminates an alignment program. Note that this function needs to be called only once, but must be called prior to the end of any program in which **ali_configure** and **ali_initialize** are called. This function releases Windows NT resources associated with the alignment system.

EXAMPLE

see **ali_configure.**

ali_write_bmp_image : write a BMP image to disk

FUNCTION

C Prototype

int WINAPI **ali_write_bmp_image** (char *fname);

Visual Basic Declaration

Declare Function **ali_write_bmp_image** Lib "mnalignt.dll" (ByVal fname As String)
As Long

where

fname = pointer to string containing disk file name

Return Values:

NO_ERROR - Operation completed without error
FILE_ERROR - Unable to access file

DESCRIPTION

This function saves the current frame buffer image in the specified disk file. The image is saved in BMP format and may be restored using the function **ali_read_bmp_image**.

EXAMPLES

Write the current frame buffer image to a disk file, "IMAGE1.BMP" in BMP format.

C/C++ EXAMPLE

```
...

ali_write_bmp_image ("IMAGE1.BMP");
```

VISUAL BASIC EXAMPLE

```
Dim rv As Long                ' return value

...

rv = ali_write_bmp_image("IMAGE1.BMP")
```

ali_write_image : write an ALIGN image to disk

FUNCTION

C Prototype

int WINAPI **ali_write_image** (char *fname);

Visual Basic Declaration

Declare Function **ali_write_image** Lib "mnalignt.dll" (ByVal fname As String) As Long

where

fname = pointer to string containing disk file name

Return Values:

NO_ERROR - Operation completed without error
FILE_ERROR - Unable to access file

DESCRIPTION

This function saves the current frame buffer image to the specified disk file. The image is saved in ALIGN format and may be restored using the function **ali_read_image**.

EXAMPLES

Write the current frame buffer image to a disk file, "IMAGE1.IMG" in ALIGN format.

C/C++ EXAMPLE

```
...

ali_write_image ("IMAGE1.IMG");
```

VISUAL BASIC EXAMPLE

```
Dim rv As Long                ' return value

...

rv = ali_write_image("IMAGE1.IMG")
```

ali_write_pat : write a pattern file to disk

FUNCTION

C Prototype

int WINAPI **ali_write_pat** (int pn, char *fname);

Visual Basic Declaration

Declare Function **ali_write_pat** Lib "mnalignt.dll" (ByVal patnum As Long,
ByVal fname As String) As Long

where

pn	=	Pattern Number 0 = write the complete data base of 16 patterns to a file on disk 1 to 16 = write the specified Pattern to a file on disk
fname	=	pointer to string containing disk file name

Return Values:

NO_ERROR	-	Operation completed without error
FILE_ERROR	-	File error
PAT_NUM_ERR	-	Pattern Number out of range
PAT_NOT_TAUGHT	-	Pattern not taught
PROT_ERR	-	Protection device not installed

DESCRIPTION

This function provides a mechanism for writing a full set of patterns or a single pattern from the Alignment System Pattern Data Base to a disk file. Files written by **ali_write_pat** must be read by **ali_read_pat**.

EXAMPLES

Write Pattern 2 to Single Pattern File "pat2.pat".

C/C++ EXAMPLE

```
int rv;
char str[80];
...

rv = ali_write_pat (2, "pat2.pat");
if (rv == NO_ERROR)
  {
  sprintf (str, "File written successfully.\n");
  ::MessageBox(NULL, str, "ali_write_pat", MB_OK);
  }
```

VISUAL BASIC EXAMPLE

```
Dim rv As Long
Dim message As String

...

rv = ali_write_pat(2, "pat2.pat")

If rv = NO_ERROR Then
    message = "File written successfully."
    MsgBox message, MB_OK, "ali_write_pat"
End If
```

ali_write_tiff_image : write a TIFF image to disk

FUNCTION

C Prototype

int WINAPI **ali_write_tiff_image** (char *fname);

Visual Basic Declaration

Declare Function **ali_write_tiff_image** Lib "mnalignt.dll" (ByVal fname As String)
As Long

where

fname = pointer to string containing disk file name

Return Values:

NO_ERROR - Operation completed without error
FILE_ERROR - Unable to access file

DESCRIPTION

This function saves the current frame buffer image in the specified disk file. The image is saved in TIFF format and may be restored using the function **ali_read_tiff_image**.

EXAMPLES

Write the current frame buffer image to a disk file, "IMAGE1.TIF" in TIFF format.

C/C++ EXAMPLE

```
...

ali_write_tiff_image ("IMAGE1.TIF");
```

VISUAL BASIC EXAMPLE

```
Dim rv As Long                ' return value

...

rv = ali_write_tiff_image("IMAGE1.TIF")
```

ali_version : read the ALIGN version

FUNCTION

C Prototype

int WINAPI **ali_version** (char *version, int numchar);

Visual Basic Declaration

```
Declare Function ali_version Lib "mnalignt.dll" (ByVal version As String,
        ByVal numchar As Long) As Long
```

where

version	=	string in which the version is returned
numchar	=	number of characters to be written into the string

Return Values:

Number of characters written into the string

DESCRIPTION

This function writes a character string that describes the ALIGN version. The user specifies the number of characters to be written. It is the responsibility of the calling program to provide a string long enough to hold the number of characters specified. The maximum size of the version string is 128 characters.

EXAMPLE

Get the ALIGN version.

C/C++ EXAMPLE

```
char AlignVersion[129];
int Numchar;
int rv;

Numchar = 128;
memset (AlignVersion, 0, sizeof (AlignVersion));
rv = ali_version(AlignVersion, Numchar);
::MessageBox(NULL, AlignVersion, "ali_align", MB_OK);
```

VISUAL BASIC EXAMPLE

```
Dim AlignVersion As String * 128
Dim Numchar As Long
Dim rv As Long

Numchar = 128
rv = ali_version(AlignVersion, Numchar)
MsgBox AlignVersion, MB_OK, "ali_version"
```

Alignment System Glossary

Alignment

The process of identifying Potential Sites within an image and then processing these sites to determine the location of the best match to the specified pattern.

Alignment Image

The portion of the frame buffer that is currently displayed on the video monitor.

Alignment Window

The portion of the image that is searched for the best pattern match during an Alignment, Teach, or AutoTeach.

AutoTeach

The process of automatically selecting the best pattern or set of patterns with the highest Teach Scores, from a specified portion of the image.

AutoTeach Window

The portion of the image from which patterns are selected during the AutoTeach process.

Contrast Ratio

An alignment result which indicates the brightness of the reported alignment location relative to the taught pattern.

Early-Stop

Early termination of the Alignment process due to an alignment quality value which equals or exceeds the Early-Stop quality threshold.

Early-Stop Threshold

A Quality Value used to adjust Early-Stop sensitivity.

Next-Best Quality

A Teach result which indicates the uniqueness of the pattern in the portion of the teach image specified by the Alignment Window.

Operator Recognition Point (ORP)

A reference point that is located at a fixed offset from its associated pattern.

Pattern

A rectangular sub-image that is specified during Teach and used as a template during Alignment.

Pattern Window

A rectangular graphical overlay that is used to specify the pattern during Teach.

Potential Site

A local area within the image that is considered as a candidate for best pattern match during the Alignment process.

Quality Value

An alignment result which indicates how closely the specified pattern matches the reported Alignment location in terms of correlation.

Search Depth

For Alignment: A number which indicates the maximum number of Potential Sites to be searched during an Alignment task.
For Teach: A number which indicates the Search Depth necessary to find the pattern in the teach image.

Teach

The process of specifying a pattern for Alignment.

Teach Score

A Teach result which indicates the suitability of the pattern for Alignment.

Type Gray

A Pattern Type which uses gray scale characteristics of the image.

TABLE OF CONTENTS