Preliminary Design of an Image Quality Tester for Helmet-Mounted Displays

by

Sheng-Jen Hsieh
Texas A&M University

and

Clarence E. Rash
Aircrew Health and Performance Division

and

Thomas H. Harding
Howard H. Beasley
John S. Martin
UES, Inc.

November 1999

Approved for public release, distribution unlimited.

U.S. Army Aeromedical Research Laboratory
Fort Rucker, Alabama 36362-0577
Notice

Qualified requesters

Qualified requesters may obtain copies from the Defense Technical Information Center (DTIC), Cameron Station, Alexandria, Virginia 22314. Orders will be expedited if placed through the librarian or other person designated to request documents from DTIC.

Change of address

Organizations receiving reports from the U.S. Army Aeromedical Research Laboratory on automatic mailing lists should confirm correct address when corresponding about laboratory reports.

Disposition

Destroy this document when it is no longer needed. Do not return it to the originator.

Disclaimer

The views, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other official documentation. Citation of trade names in this report does not constitute an official Department of the Army endorsement or approval of the use of such commercial items.
**Title:** Preliminary Design of an Image Quality Tester for Helmet-Mounted Displays

**Authors:** Sheng-Gen Estee, Clarence Rash, Thomas Harding, Howard Beasley, John Marin

**Type of Report:** Final

**Date of Report:** November 1999

---

**Abstract:**

Helmet-mounted displays (HMDs) provide essential pilotage and fire control imagery information for pilots. However, image quality testers for HMD field performance validation do not currently exist. This research employed techniques from imaging analysis and interpretation, and computer-aided design/computer-aided manufacturing (CAD/CAM) to investigate a preliminary design for a portable HMD image tester.

For this study, a charged couple device (CCD) camera and lens were selected. Hardware characteristics such as viewing angles in horizontal and vertical positions, dynamic working range at day and night, pixel resolution, focal length, and aperture ratio were evaluated with regard to HMD functionality. Experiments to evaluate camera sensitivity and test pattern merits were conducted using a programmable micro positioning system, CCD camera, optical fixtures and benches. Next, the relative ratio among features within the image profile was established and an ideal image profile and evaluation criteria were established based on the experimental results. Third, image processing algorithms and techniques, such as edge detection, were developed and applied in test pattern feature...
detection. A software prototype, including modules for image capture, image analysis, interpretation, and user manuals, was developed. Finally, a concept hardware package design proposed. This design incorporates a notebook computer with a flat panel display to interface with the camera and software prototype; and incorporates fixtures for the HMD, camera, computer, and power supply. This design will allow the tester to be used in the field.
# Table of contents

Introduction ................................................................. 1

Functionality and operating process ................................................ 2

Methodology ................................................................. 3

   Image capture hardware specification ............................................. 4

   Test pattern features investigation ............................................. 6

Software prototype design ................................................. 9

   Image capture module ......................................................... 9

   Image analysis and interpretation module .................................... 11

   Algorithm design ............................................................. 11

   Testing and validation ....................................................... 13

Hardware package design ................................................. 15

Conclusions and future directions ................................................. 15

References ........................................................................ 18

Appendix A. List of manufacturers ............................................. 19

Appendix B. Software prototype program ..................................... 20

## List of figures

1. The IHU of the AH-64 IHADSS ............................................. 1

2. The IHADSS HDU ............................................................ 2

3. Display size .................................................................. 3

4. Test pattern from the IHADSS HMD ..................................... 3
### Table of contents (continued)

<table>
<thead>
<tr>
<th>List of figures (continued)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5. Flow chart for the HMD prototype tester operation</td>
</tr>
<tr>
<td>6. Experimental setup for camera sensitivity analysis</td>
</tr>
<tr>
<td>7. Sampling locations on the test pattern</td>
</tr>
<tr>
<td>8. Plot of photometer and CCD camera data</td>
</tr>
<tr>
<td>9. Setup for test pattern measurement</td>
</tr>
<tr>
<td>10. Test pattern design based on measurement results</td>
</tr>
<tr>
<td>11. Replicated test pattern image</td>
</tr>
<tr>
<td>12. Measurement of luminance of the center lines</td>
</tr>
<tr>
<td>13. Center lines measurement with varied focus</td>
</tr>
<tr>
<td>14. Designed test pattern with focus on the center lines</td>
</tr>
<tr>
<td>15. Opening screen of prototype software</td>
</tr>
<tr>
<td>16. Image capture module</td>
</tr>
<tr>
<td>17. Image capture component</td>
</tr>
<tr>
<td>18. Image processing component</td>
</tr>
<tr>
<td>19. Image analysis and interpretation module</td>
</tr>
<tr>
<td>20. Tilted test pattern binary images from image analysis module</td>
</tr>
<tr>
<td>21. Overall testing results of an HMD</td>
</tr>
<tr>
<td>22. Tilted test pattern before (left) and after (right) Sober edge detection</td>
</tr>
<tr>
<td>23. Investigation of CCD image capture arrangement</td>
</tr>
<tr>
<td>24. CAD concept of prototype hardware design</td>
</tr>
<tr>
<td>Table of contents (continued)</td>
</tr>
<tr>
<td>-------------------------------</td>
</tr>
<tr>
<td>List of tables</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measured data and correlation coefficient from photometer and CCD camera</td>
</tr>
</tbody>
</table>
**Introduction**

Helmet-mounted displays (HMDs) are a gateway to the pilot for viewing pilotage and fire control imagery. In Army aviation, the AH-64 Apache helicopter uses an HMD system known as the Integrated Helmet and Display Sighting System (IHADSS). The IHADSS consists of various electronic components and a helmet/display system called the Integrated Helmet Unit (IHU). The IHU (Figure 1) includes a helmet, visor housings with visors, miniature cathode ray tube (CRT), and helmet display unit (HDU). The HDU serves as an optical relay device which conveys the image formed on the CRT through a series of lenses, off a beamsplitter (called a combiner), and into the aviator’s right eye (Figure 2). The CRT is 1 inch in diameter and uses a P-43 phosphor. The combiner is a multilayer dichroic filter which is maximized for reflectance at the peak emission of the P-43 phosphor.

The U.S. Army is currently developing the next generation reconnaissance aircraft, the RAH-66 Comanche. This aircraft will incorporate an HMD which will be binocular in design. While its final design is still in review, it will basically consist of two image sources (either miniature CRTs or liquid crystal displays) with two sets of optics, delivering imagery to both eyes.

Currently, there is no existing image quality tester for HMD validation in the field. To maintain system integrity and readiness, and to provide pilots with validated pilotage, navigation, and fire control imagery, there is a need to design and construct an image quality testing tool for the HMD. The objective of this study is to propose and test a design concept for an image quality tester for HMD subsystems. The tester can be used as a validation tool to verify settings for regular flight missions and for preventive maintenance tasks. The first prototype tester will be designed for the AH-64's IHADSS HMD.
Functionality and operating process

The proposed tester will allow pilots and maintenance personnel to validate the image quality of an HMD. Basic required characteristics include (1) simple design, (2) ease of use, (3) robustness, and (4) accuracy for operations and maintenance. The prototype should be small enough to fit into a brief case, which would include a lap-top, image capture system, and power supply pack.

The IHADSS HMD has a monocular 30-degree vertical by 40-degree horizontal field-of-view (FOV). Future HMDs most likely will have larger FOVs and be binocular in design. HMD corner obscurations are generally permissible and symmetrical for the IHADSS, as illustrated in Figure 3. Since hardware changes to the various aircraft electronics will not be allowed, image quality validation must be performed using manufacturer built-in test patterns. The built-in test pattern of the IHADSS HMD is used as the inspection specification on which the first tester will be based. The test pattern shows strips of gray opposed along the vertical center lines. Each strip contains 8 to 10 shades of gray, depending on the contrast ratio. Adjacent shades have a square root of 2 differential of brightness. Figure 4 is a snapshot of the test pattern captured from the IHADSS HMD. For more detailed discussion of the HMD test pattern features, see the Honeywell, Inc. study guide (1985) and Harding et al. (1995). For testing this test pattern, the inspection features used by the image quality tester prototype will include (1) four vertical center lines, (2) one horizontal center line, (3) two gray shade patterns (with 8 to 10 shades), and (4) a boundary box.
Based on the design objectives and inspection procedures, the tester operation procedures are as follows: (1) the pilot adjusts the HMD settings and passes the HDU to the crew chief; (2) the crew chief inserts the HMD into a fixture; (3) the system examines the center and horizontal line features of the test pattern using a narrow-angle lens; (4) the system inspects the test pattern for image displacement and/or disorientation; (5) the system examines the number of gray-shades, the focus, luminance, and boundary lines, using a 42-degree wide-angle lens; and (6) the system generates a final report. Figure 5 shows a flow chart for the proposed operation procedures.

Methodology

This study involved designing and testing (1) the hardware specification for image capture, (2) the test pattern inspection features, (3) the software prototype, and finally (4) the hardware prototype. Experiments and statistical analysis tools were applied throughout the design process.
Image capture hardware specifications

To determine the needed camera and lens specification for test pattern image capture, experiments were conducted to verify the sensitivity of a candidate camera. The camera and a Photo Research (Appendix A) model 1980 photometer were mounted using a reconfigurable optical fixture and bench accessories and were used to capture an electronically generated gray shade test pattern. Figure 6 illustrates the experimental setup. The luminance of the test pattern image was registered by the charged couple device (CCD) camera (and image capture card) and the photometer. Figure 7 shows the locations where data were sampled from the test pattern. These data were measured from a fixed position along a horizontal line across the entire test pattern. Three measurements were taken from each region. An observation resulting from the experiment was that the luminances of the gray shades presented in the test pattern were not linearly distributed between 0 and 255. The differential of luminance for adjacent shades was greater than an approximate square root of 2. A statistical analysis was performed on these data. Results indicated that the luminance levels measured by the photometer were consistent with data from the camera and image capture card up to and including the 7th gray shade. It can be seen that the CCD saturated after the 7th gray shade area. To prevent this, the aperture of the CCD would have to be adjusted. If only the first seven gray shades are used in the analysis, correlation is 0.98. The table and Figure 8 record the data collected from both instruments and the statistical analysis results.

![Figure 6. Experimental setup for camera sensitivity analysis.](image)

![Figure 7. Sampling locations on the test pattern.](image)
Table.
Measured data and correlation coefficient from photometer and CCD camera.

<table>
<thead>
<tr>
<th>Gray shade</th>
<th>Photometer luminance readings</th>
<th>CCD gray level readings</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.25</td>
<td>3.32</td>
</tr>
<tr>
<td>2</td>
<td>7.47</td>
<td>7.51</td>
</tr>
<tr>
<td>3</td>
<td>17.07</td>
<td>16.99</td>
</tr>
<tr>
<td>4</td>
<td>30.51</td>
<td>30.54</td>
</tr>
<tr>
<td>5</td>
<td>48.28</td>
<td>48.24</td>
</tr>
<tr>
<td>6</td>
<td>71.9</td>
<td>71.86</td>
</tr>
<tr>
<td>7</td>
<td>98.35</td>
<td>98.54</td>
</tr>
<tr>
<td>8</td>
<td>127.1</td>
<td>127.2</td>
</tr>
<tr>
<td>9</td>
<td>157.9</td>
<td>158.1</td>
</tr>
<tr>
<td>10</td>
<td>187.4</td>
<td>187.4</td>
</tr>
<tr>
<td>11</td>
<td>221.2</td>
<td>221.4</td>
</tr>
<tr>
<td>12</td>
<td>200.7</td>
<td>200.6</td>
</tr>
</tbody>
</table>

Luminance vs gray level (7 shades): Correlation = 0.983886; Fisher’s z = 2.406549; Probability = 0.0006

Figure 8. Plot of photometer and CCD camera data.
In an attempt to capture the test pattern image on the IHADSS fully, several different cameras (with standard lenses) were evaluated. However, although the full test pattern could be captured, the details of the four vertical center lines could not be differentiated. Therefore, a decision was made to use a narrow angle lens to zoom in on the center area of the test pattern in order to capture the details of the center lines. HMDs are also used at night; therefore, the prototype tester--specifically the camera--should provide good sensitivity at low luminance levels. First order specifications for the required camera were summarized as follows:

1. Sensitivity: \#0.005 lux
2. Focus: To infinity
3. Resolution: > 768 x 498 pixels
4. Focal length: \(\sim\frac{1}{2}\) inch
5. Iris: Manual
6. Fields of view: >40 (H) x 30 (V) degrees and \(~5 \times 3\) degrees

Test pattern features investigation

An additional experiment was conducted to investigate various aspects of capturing the test pattern. Multiple cameras were used since a single camera that met all the desired specifications was not available at the time of this study. Aspects of interest included the size of the pattern, number of different features, relative luminance ratios among features, spatial content of each feature, and number of gray shades. The IHADSS HMD was mounted on the top of the optical post, and the post was fixed on top of a round optical table controlled by a programmable position table. The table was driven by a stepping motor with an accuracy of 1 micron (\(\mu\)m). The test pattern image was projected onto a video monitor for observation. Figure 9 shows the experimental setup. The entire test pattern image from the HMD was captured and constructed through a series of mini steps in the horizontal and vertical directions. The overall picture was approximately 38 x 29 degrees, which was close to the specification in the study guide (Honeywell, Inc., 1985). The center line occupied approximately 0.5 degree out of 38 degrees. There were two strips with 10 to 12 gray shades mirrored opposite the center lines. Figure 10 shows the structure of the IHADSS test pattern. A series of images were taken to probe the content of each gray shade in terms of luminance. Based on the observed information, a series of image files was constructed and used as an image profile for purposes of the software prototype development. Figure 11 displays this replicated test pattern image.

![Figure 9. Setup for test pattern measurement.](image-url)
Figure 10. Test pattern design based on measurement results.

Figure 11. Replicated test pattern image.

A similar experiment was conducted to detail the center lines within the test pattern. Figure 12 shows the luminance scan measurements for the center lines. The four peaks represent the four center lines which are spread out over 0.8 degree from valley to valley and 0.4 degree peak to peak. The average peak width is about 0.0969 degree and the average distance between peaks is about 0.1347 degree. Note: A measurement of 1 degree is about 485 \( m \) in the object plane.

Another experiment was conducted to probe the state of the center lines when the HMD is in focus and not in focus. Varied focus values of -1 to 1 diopter of CRT were applied. Measurements of the four vertical center lines were taken. An interesting finding was, when the
HMD was in focus, the ratio of luminances between bottom to mid-peak (B) and peak to valley (A) was close to 1. However, when the setting was not in focus, the B:A ratio was less than one. Figure 13 documents these observations and illustrates the concept. Findings from the above experiments, such as measurements, luminance ratios, and the content of each feature within the test pattern, were used to create a test pattern image using graphics software. Figure 14 shows an image of such a test pattern using a 5 x 4 degree lens to focus on the center lines of the test pattern. In addition, the ratio of the square root of 2 luminance difference was used to design gray shades ranging from 0 to 255 gray levels.

![Scan of 4 lines (0 diopter)](image)

Figure 12. Measurement of luminance of the center lines.

![Scan of 4 Lines at with CRT focus varied](image)

Figure 13. Center lines measurement with varied focus.

To emulate potential human errors in setting up the HMD, a set of parameters (including brightness, orientation, spatial adjustment, and contrast) were manipulated and the resulting images captured. These images were used as a basis for creating new image files. These designed images
were used to test the software prototype. The experiments were carried out using similar methods. For example, to measure the potential displacement of the test pattern, a camera was mounted facing the HMD. The test pattern was projected onto a video monitor by means of a personal computer (PC). Measurements were taken before and after the spatial adjustments. The maximum adjustments in the upward, downward, left and right directions were 3.57, 2.98, 4.90 and 4.90 degrees, respectively, based on an FOV of 40 x 31 degrees (Harding et al., 1995).

Software prototype design

The software prototype was designed to capture, analyze, and interpret the image against test pattern features such as the four center lines and number of gray shades. Accordingly, the prototype design will require three modules--image acquisition, image analysis and interpretation--as well as on-line user help. Figure 15 shows the modules involved in the prototype. Visual Basic (VB) was used to develop the prototype because of its flexibility in linking and embedding with other commercial software and because it was a powerful toolbox for rapidly prototyping a complicated window. In the following sections, we describe the functionality of each module and how the modules are integrated. Algorithms developed to interpret the image follow. Finally, testing and validation of the code is addressed. The source code for the program can be found in Appendix B.

Image capture module

The VB Object Linking and Embedding (OLE) capability allows integration of other programs. In this case, the image capture graphics program served as an object which was linked into the VB main program. The graphics program was launched by activating the linked object. Once the object had been activated, the VB main program allowed the user to modify, save, or open documents created by the graphic program in VB’s integrated design environment (IDE). After the user was done with the image capture graphics program, control was released to the VB environment. The graphics program itself contained three components: the driver used to activate
the image capture card and digitize the video signal into a graphics image format (e.g., bitmap or jpeg); an image processing shell which allowed image manipulation (e.g., sharpening and lightening); and an on-line user manual. Figure 16 shows the opening screen for the image capture module. Figures 17 and 18 show image capture and processing subcomponents.

Figure 15. Opening screen of prototype software.

Figure 16. Image capture module.

Figure 17. Image capture component.

Figure 18. Image processing component.
Image analysis and interpretation module

The image analysis and interpretation module (1) detects the presence of key features such as center lines within the test pattern, (2) compares selected features against the feature specification, and (3) generates findings. VB components were created to provide these functions and to interface with other modules. A subwindow titled “evaluation criteria” was created to analyze and interpret the captured image from an HMD. A few created algorithms were coded in VB to perform the analysis. Other subwindows, such as a directory box and file list boxes were created to allow retrieval of image files for analysis. Finally, an additional subwindow was designed to display the image currently being analyzed. This module also allows access to other modules via a button control. Figure 19 shows the image analysis and interpretation module.

![Image Analysis and Interpretation Module](image)

Figure 19. Image analysis and interpretation module.

Algorithm design

Algorithms were developed to detect various features within the test pattern as described earlier. These are described below:

A. Identify the number of center lines.

Step 1. Apply binary image technique to the entire image.
Step 2. Draw multiple lines across X and/or Y axes.
Step 3. Identify mask with feature of B/W . . . W/B.
Step 4. Store the intersection points in an array with multiple dimensions.
Step 5. Construct regression lines based on the points within each dimension.
Step 6. Develop regression lines to compare the parallel property.
Step 7. Average the intersection points around the array to obtain the number of estimated lines.

Note 1: B = black pixel and W = white pixel.
Note 2: Use of linear regression analysis would make the linear mode robust and insensitive to noise presence.

B. Identify the center point.
Step 1. Construct a regression line based on all the intercepted points.
Step 2. Identify the midpoint of an array as a starting point with the feature of W/B/W.
Step 3. Examine neighboring pixels to see if a W/W/W mask exists.
Step 4. If a W/W/W mask exists, stop the procedure; else next step.
Step 5. Check the distance of neighboring pixels from the regression line using a 3 x 3 area.
Step 6. Select the point with the smallest distance from the regression line as the next point.
Step 7. Go to step 3.

C. Identify test pattern orientation and displacement.

Step 1. Compute a theoretical center as point A.
Step 2. Identify the actual center point (based on part B) as point B.
Step 3. Compute the distance between point A and B as d.
Step 4. If d is equal to 0; then the displacement is zero.
Step 5. Construct lines between a given point with points A and B.
Step 6. Compute the angle between lines as orientation angle

D. Identify the number of gray shades within a test pattern.

Step 1. Use the center point as a starting point.
Step 2. Pick five points across the center line that are within the boundary of gray shades.
Step 3. Compute the average gray level of the five points.
Step 4. Store it in one dimension of the array.
Step 5. If the boundary is not reached, move up or down a given distance, and go to Step 3; else next.
Step 6. Use of square root of two differences to determine the number of gray shades.

E. Identify boundary lines.

Step 1. Use the center point and boundary ratio to determine the region of the image boundary.
Step 2. Locate a starting point white pixel to use for back tracking the rest of the white pixels for each line segment.

F. Identify the focus setting.

Step 1. Use line scan technique to record the pixels along the center lines.
Step 2. Use the B/W/B mask to identify the separation of lines.
Step 3. Compute the ratio of bottom to mid-peak and peak to valley for all four lines.
Step 4. If the ratio is approximately one, we may conclude that the focus setting is good; or else check the focus setting
Other methods for center point detection exist. However, these were deemed less appropriate for this application. For instance:

**Alternate approach #1:**

Step 1. Use of the mask of  

```
  b b b b
```

Note: If the orientation of the image is unknown, this method can be time consuming.

**Alternate approach #2:**

Step 1. Find the center point of each line.
Step 2. Use the averaging method to find the center of all the centers.

Note: This method involves more steps than the proposed one, because you must first find the center of each line and there are four lines to be examined.

**Alternate approach #3:**

Step 1. Identify the boundary of the image.
Step 2. Use the center of gravity method to find the center of the image.

Figure 20(a-d) shows screens from the image analysis module. Figure 20a shows a binary image of the test pattern after the binary image technique had been applied to the test pattern captured from the HMD. Figure 20b shows the four center lines that were identified from the binary image (Figure 20a). After the center lines had been identified, the image analysis module identified the center point of the image. Figure 20c shows the coordinates (y only shown) of the center point. The image analysis module then determined if the image was tilted or not. Figure 20d displays the tilt angle of the image. The analysis results are summarized and displayed in Figure 21. A primary feature of the image analysis module is to identify features present in the captured test pattern. The “Sober operator,” a well known edge detection technique, is used to identify the boundaries of the features and, thereby, allow the analysis module to determine whether or not the required features are present in the captured test pattern image. Figure 22 shows the same image before and after the Sober operator is applied.

**Testing and validation**

To verify the accuracy of the program, language debugging tools, and split-half and back tracking strategies were imposed throughout the coding process. The program results were compared with the simulation results. For example, to check the accuracy of the constructed regression line, the same data points also were analyzed and compared with the results obtained from a statistics package and hand calculation.
Figure 20. Tilted test pattern binary images from image analysis module.

Figure 21. Overall testing results of an HMD.
Hardware package design

A preliminary concept for the hardware package design consists of a display/output module, power supply module, and image capture module. The display/output module should be designed to display/generate inspection results of an HMD test pattern. The power supply module should be designed to provide the voltages needed for the cameras and computer. The design also should include a rechargeable battery pack which will allow the unit to operate in areas without an external power supply. The power supply would be required to provide 12- and 9-volt outputs for the cameras and computer, respectively. Finally, the image capture module must be designed to hold an HMD and two cameras in fixed and contained positions, thereby preventing potential noise that may affect the inspection accuracy. A proposed design is as follows: Two cameras arranged vertically and facing the HMD. [Figure 23 shows one method investigated for aligning the CCD image capture cameras and the HMD.] An inverted HMD fixture will be the most likely one be used in the final concept. The fixture would be mounted with spring return locks on the sides and bottom. The spring return locks will lock the HMD in a fixed position. These locks would prevent the inspection process from continuing if the HMD is not positioned correctly. Once the HMD is in the correct position, a proximity sensor will be used to trigger the image system to start the image capture and interpretation processes. The cover of the image capture module is in the shape of an inverted HMD. It is designed to cover the HMD tightly once it is in the correct position, and to eliminate any optical noise from the surrounding environment. To enhance the speed of image analysis, an Electronic Programmable Read Only Memory (EPROM) chip, custom programmed to load the executable program for image analysis, could be used. Figure 24 illustrates a preliminary computer aided design (CAD) concept of the hardware prototype design.

Conclusions and future directions

In this project, a design framework for an image quality tester was proposed and evaluated. Functionality and requirements of the tester were identified. Experiments were conducted to test
camera sensitivity and to probe aspects of an HMD test pattern using programmable micro-positioning systems and a CCD camera. Test pattern specifications were developed based on these observations. A strategy for image analysis and interpretation was formed, and algorithms were designed to verify the test pattern of a given HMD against the specifications. A prototype software package was written to inspect the test pattern and verify the effectiveness of the algorithms. Finally, a design framework for a concept hardware package was proposed.
To build a brassboard version of a tester, future work must include: (1) fabrication of the hardware design using inverse casting techniques, (2) integration of software and hardware components for a prototype design, (3) field testing of the prototype, (4) incorporation of learning algorithms to increase inspection accuracy, and (5) expansion of functionality from validation to on-line real time interactive adjusting and self-tuning based on a given environmental scenario. From the maintenance perspective, the work can be expanded to self-diagnosis and preventative maintenance (such as life-time prediction).
References


Appendix A.

List of manufacturers.

Photo Research
3000 North Hollywood Way
Burbank, CA 91505
Appendix B

Software prototype program.
Private Sub Timer1_Timer()

    Dim PauseTime, Start

    PauseTime = 2  ' Set duration.
    Start = Timer  ' Set start time.
    Do While Timer < Start + PauseTime
        DoEvents    ' Yield to other processes.
        Loop
        Unload Me
        Form2.Show
    Loop

End Sub
Form2 - 1

Private Sub cmdQUIT_Click()

Unload Form2
End

End Sub

Private Sub Command2_Click() 'Image Analysis

Unload Form2
Form4.Show

End Sub
Private Sub Continue_Click()
Unload Form3
Form4.Show
End Sub

Private Sub Quit_Click()
Unload Form3
Form2.Show
End Sub
Form 4 - 1

Public Displacement, Angle As Double
Public CenterLineSlope As Double
Public CenterLineIntercept As Double
Public Center_Point_X, Center_Point_Y As Double

Const intUpperBoundX = 320 '320 total
Const intUpperBoundY = 244 '244 total
Const N = 4 ' # of center line

Dim X, Y As Integer
Dim picObject0, picObject1 As Picture
Dim Coord_X(0 To 45, 0 To 10) As Integer
Dim Coord_Y(0 To 45, 0 To 10) As Integer
Dim pixels(0 To intUpperBoundX, 0 To intUpperBoundY) As Long
Dim ImagePixels(2, intUpperBoundX, intUpperBoundY) As Long
Private Sub cmdSelect_Click()

Dim FileName, EdgeDetection As String
Dim bytRed, bytGreen, bytBlue, bytAverage As Integer

On Error GoTo FileError
If (Right$(Dir1.Path, 1) = ") Then
    FileName = File1.Path & File1.FileName
Else
    FileName = File1.Path & "" & File1.FileName
End If

Open FileName For Input As #1
Set picObject0 = LoadPicture(FileName)
Set Picture0.Picture = picObject0
Close #1

For X = 0 To intUpperBoundX - 1
    For Y = 0 To intUpperBoundY - 1
        pixels(X, Y) = Picture0.Point(X, Y)
        bytRed = GetRed(pixels(X, Y))
        bytGreen = GetGreen(pixels(X, Y))
        bytBlue = GetBlue(pixels(X, Y))
        ImagePixels(0, X, Y) = bytRed
        ImagePixels(1, X, Y) = bytGreen
        ImagePixels(2, X, Y) = bytBlue

        'the file you have is in gray scale; therefore, you do not need to average
        Picture0.PSet (X, Y), RGB(bytRed, bytGreen, bytBlue)
    Next Y
Next X

Exit Sub
FileError: MsgBox "File Error!"

End Sub

Private Sub cmdCenter_and_Boundary_Click()

Set Picture0.Picture = picObject0
For X = 0 To intUpperBoundX - 1
    For Y = 0 To intUpperBoundY - 1
        Picture0.PSet (X, Y), Picture0.Point(X, Y)
    Next Y
Next X

End Sub
From 4 - 2

Next Y
Next X
Set picObject1 = Picture0.Picture
SavePicture picObject1, "TEST1.BMP"
LoadPicture ("TEST1.BMP")

End Sub

Private Sub cmdEdgeDetection_Click()

Dim RGBLong As Long
Dim G_X, G_Y, G_X_Y As Integer
Dim bRXm1Y, bRXm1Ym1, bRXm1Ym1 As Integer
Dim bRXp1Y, bRXp1Yp1, bRXm1Yp1, bRXm1Yp1 As Integer
Dim bytRed, bytGreen, bytBlue As Integer

Set Picture0.Picture = picObject0

For X = 0 To intUpperBoundX - 1
    For Y = 0 To intUpperBoundY - 1

        If (X = 0 Or X = intUpperBoundX - 1 Or Y = 0 Or Y = intUpperBoundY - 1) Then
            bytRed = ImagePixels(0, X, Y)
            bytBlue = ImagePixels(1, X, Y)
            bytGreen = ImagePixels(2, X, Y)
            RGBLong = RGB(bytRed, bytGreen, bytBlue)
            Picture0.PSet (X, Y), RGBLong
        Else
            G_X = 0
            G_Y = 0
            G_X_Y = 0
            bRXm1Y = ImagePixels(0, X - 1, Y)
            bRXm1Ym1 = ImagePixels(0, X - 1, Y - 1)
            bRXm1Yp1 = ImagePixels(0, X - 1, Y + 1)
            bRXm1Ym1 = ImagePixels(0, X - 1, Y - 1)
            bRXm1Yp1 = ImagePixels(0, X - 1, Y + 1)
            bRXp1Y = ImagePixels(0, X + 1, Y)
            bRXp1Ym1 = ImagePixels(0, X + 1, Y - 1)
            bRXp1Yp1 = ImagePixels(0, X + 1, Y + 1)
            G_X = bRXp1Ym1 + 2 * bRXp1Y - bRXm1Ym1 - 2 * bRXm1Y + bRXm1Yp1
            G_Y = bRXm1Yp1 + 2 * bRXp1Y + bRXp1Yp1 - bRXm1Ym1 - 2 * bRXm1Y - bRXm1Yp1
            G_X_Y = Sqr((G_X * G_X) + (G_Y * G_Y))
            bytRed = G_X_Y
            bRXp1Y = ImagePixels(1, X, Y + 1)
            bRXm1Y = ImagePixels(1, X - 1, Y - 1)
            bRXm1Ym1 = ImagePixels(1, X - 1, Y - 1)
            bRXm1Yp1 = ImagePixels(1, X - 1, Y + 1)
            bRXm1Ym1 = ImagePixels(1, X - 1, Y - 1)
            bRXm1Yp1 = ImagePixels(1, X - 1, Y + 1)
            bRXp1Ym1 = ImagePixels(1, X + 1, Y - 1)
            bRXp1Yp1 = ImagePixels(1, X + 1, Y + 1)

        End If

    Next Y

Next X

LoadPicture ("TEST1.BMP")
G_X = bRXp1Ym1 + 2 * bRXp1Y + bRXp1Yp1 - bRXm1Ym1 - 2 * bRXm1Y - bRXm1Yp1
G_Y = bRXm1Yp1 + 2 * bRXp1Yp1 + bRXm1Ym1 - 2 * bRXYm1 - bRXp1Ym1
G_X_Y = Sqr((G_X * G_X) + (G_Y * G_Y))

bytBlue = G_X_Y

bRXY = ImagePixels(2, X, Y)
bRXYp1 = ImagePixels(2, X, Y + 1)
bRXm1Y = ImagePixels(2, X - 1, Y)
bRXYm1 = ImagePixels(2, X, Y - 1)
bRXm1Yp1 = ImagePixels(2, X - 1, Y + 1)
bRXm1Ym1 = ImagePixels(2, X - 1, Y - 1)
bRXp1Y = ImagePixels(2, X + 1, Y)
bRXp1Ym1 = ImagePixels(2, X + 1, Y - 1)
bRXp1Yp1 = ImagePixels(2, X + 1, Y + 1)

G_X = bRXp1Ym1 + 2 * bRXp1Y + bRXp1Yp1 - bRXm1Ym1 - 2 * bRXm1Y - bRXm1Yp1
G_Y = bRXm1Yp1 + 2 * bRXp1Yp1 + bRXm1Ym1 - 2 * bRXYm1 - bRXp1Ym1
G_X_Y = Sqr((G_X * G_X) + (G_Y * G_Y))

bytGreen = G_X_Y

Picture0.PSet (X, Y), RGB(bytRed, bytGreen, bytBlue)

End If

Next Y
Next X

End Sub
Private Sub cmdGray_Shade__Click()

Set Picture0.Picture = picObject0
For X = 0 To intUpperBoundX - 1
  For Y = 0 To intUpperBoundY - 1
    Picture0.PSet (X, Y), Picture0.Point(X, Y) - 5
  Next Y
Next X

End Sub
Private Sub cmdFoucs_Click()

Set Picture0.Picture = picObject0
For X = 0 To intUpperBoundX - 1
  For Y = 0 To intUpperBoundY - 1
    Picture0.PSet (X, Y), Picture0.Point(X, Y) - 10
  Next Y
Next X

End Sub
Private Sub cmdDis_and_Orientation_Click()
Const interval_range = 7
Dim WhitePixel, BlackPixel As Long
Dim linescan As Integer
Dim i, j, k, L, IntX, Temp_X, Temp_Y As Integer
Dim Flag, SumTline, Dummy As Integer
Dim interval As Integer
Dim ZeroO_X, ZeroO_Y As Double
Dim L1SlopeR, L2SlopeR, L3SlopeR, L4SlopeR, L1SlopeY, _
   L2SlopeY, L3SlopeY, L4SlopeY, AvgSlope As Double
Form4 - 4

Dim UpperBound, LowerBound As Double
Dim InterceptY As Integer
Dim Count_Points(0 To 403) As Integer
Dim TempInt, Choice As Integer
Dim Dum(0 To 15) As Double
Dim TempDouble As Double
Dim Tline(0 To 50) As Integer
Dim Oripixels(0 To intUpperBoundX, 0 To intUpperBoundY) As Long
Dim Displacement, Angle, Theta As Double
Dim CenterLineSlope As Double
Dim CenterLineIntercept As Double
Dim Center_Point_X, Center_Point_Y As Double
Dim TempText As String

Open "c:\windows\desktop\InspResults.txt" For Output As #1

For X = 0 To intUpperBoundX - 1
    For Y = 0 To intUpperBoundY - 1
        Oripixels(X, Y) = pixels(X, Y)
    Next Y
    Next X

'Apply the binary image technique

For X = 0 To intUpperBoundX - 1
    For Y = 0 To intUpperBoundY - 1
        If (Oripixels(X, Y) < RGB(255, 255, 255)) Then
            Oripixels(X, Y) = 0
        Else
            Oripixels(X, Y) = RGB(255, 255, 255)
        End If
        Picture0.PSet (X, Y), Oripixels(X, Y)
    Next Y
    Next X

'Find the number of center lines
'A line is BW...WB; if there is less than four BW...WBs; then Image is tilled
'white interval should be less than 7 for the central lines
'use Black/White/Black to find a line

linescan = 0
interval = 1

For Y = 50 To intUpperBoundY - 1
    Tline(linescan) = 0
    Flag = 0
    L = 0
    For X = 0 To intUpperBoundX - 1
        If ((Oripixels(X, Y) = RGB(0, 0, 0)) And _
            (Oripixels(X + 1, Y) = RGB(255, 255, 255))) Then
            For interval = 1 To interval_range - 1
                If (Oripixels(X + 1 + interval, Y) = RGB(0, 0, 0)) Then
                    Tline(linescan) = Tline(linescan) + 1
                    Flag = 1
                    Coord_X(linescan, L) = X + 1  'of each line
                    Coord_Y(linescan, L) = Y
                    L = L + 1
                End If
            Next interval
            If Flag = 1 Then
                L = L + 1
            End If
        End If
    Next X
    linescan = linescan + 1
Form4 - 5

L = L + 1
End If

interval = interval_range 'stop the for loop
Next interval
End If

Next X
Y = Y + 10 ' 5          'to have 40 arbitrary vertical lines
If (Flag = 1) Then
linescan = linescan + 1
End If
Next Y

k = 0
SumTline = 0
For j = 0 To linescan - 1 'from prev. routine # of arb. ver. lines
If (Tline(j) > 0) Then
SumTline = SumTline + Tline(j)
k = k + 1
End If
Next j

If (3.5 <= (SumTline / k) <= 4.5) Then
MsgBox ("Number of center lines is " & N)
L1SlopeR = GetSlope(linescan, 0, 0)
L1SlopeY = GetSlope(linescan, 0, 1)
L2SlopeR = GetSlope(linescan, 1, 0)
L2SlopeY = GetSlope(linescan, 1, 1)
L3SlopeR = GetSlope(linescan, 2, 0)
L3SlopeY = GetSlope(linescan, 2, 1)
L4SlopeR = GetSlope(linescan, 3, 0)
L4SlopeY = GetSlope(linescan, 3, 1)
AvgSlope = (L1SlopeY + L2SlopeY + L3SlopeY + L4SlopeY) / 4
LowerBound = 0.025 * AvgSlope
UpperBound = 1.025 * AvgSlope

'Use the absolute value; therefore, it works on both -/+ values
If ((Abs(LowerBound) <= Abs(L1SlopeY) <= Abs(UpperBound)) And _
    (Abs(LowerBound) <= Abs(L2SlopeY) <= Abs(UpperBound)) And _
    (Abs(LowerBound) <= Abs(L3SlopeY) <= Abs(UpperBound)) And _
    (Abs(LowerBound) <= Abs(L4SlopeY) <= Abs(UpperBound))) Then
    MsgBox ("Four lines are parallel !")
Else: MsgBox ("Potential errors in finding parallel lines")
End If

Else
    MsgBox ("Number of center lines is " & SumTline / k)
End If

'The following is to find the center point of the image
'Step 1: Find the black pixel
'Step 2: Calculate the neighborhood pixels distance to the regression line
'Step 3: Locate the one that has the smallest distance
'Step 4: Check to see if the feature of w
    www
    W
Form 4 - 6

' been meet
' if not; based on current X, Y; go to Step 1

BlackPixel = RGB(0, 0, 0)
WhitePixel = RGB(255, 255, 255)

CenterLineSlope = GetSlope(linescan, 0, 2)
CenterLineIntercept = GetSlope(linescan, 0, 3)

MsgBox ("C.L.Intercept = " & CenterLineIntercept)
MsgBox ("C.L.Slope = " & CenterLineSlope)

For Y = 20 To intUpperBoundY - 1
X = (Y * CenterLineSlope) + CenterLineIntercept
IntX = X
If (Oripixels(IntX, Y) = BlackPixel) Then

L = 0
Temp_X = 0
Temp_Y = 0
For i = -1 To 1
    For j = -1 To 1
        If (Oripixels(IntX + i, Y + j) = WhitePixel) Then
            Temp_X = Temp_X + (IntX + i)
            Temp_Y = Temp_Y + (Y + j)
            L = L + 1
        End If
    End For
End If
If (L >= 3) Then
    'Neighborhood pixels are White
    Center_Point_X = Temp_X / L
    Center_Point_Y = Temp_Y / L
    MsgBox ("Center X = " & Center_Point_X)
    MsgBox ("Center Y = " & Center_Point_Y)
    Beep
    i = 1
    j = 1
    Y = intUpperBoundY
End If
Next j
Next i

L = 0
Dum(L) = 0
For i = 0 To 1
    For j = 0 To 1
        If (i <> 0 Or j <> 0) Then
            Dum(L) = Measure_Distance(CenterLineIntercept, CenterLineSlope, X + i, Y + j)
            L = L + 1
        End If
    End For
End If
Next i

For k = 0 To L - 1
    If (Dum(k) < Dum(k + 1)) Then
        TempDouble = Dum(k)
        Dum(k) = Dum(k + 1)
        Dum(k + 1) = TempDouble
    End If
Next k

For i = 0 To 1
    For j = 0 To 1
        If ((i <> 0 Or j <> 0) And (Dum(L - 1) = Measure_Distance(CenterLineIntercept, CenterLineSlope, X + i, Y + j)))
Form4 - 7

Then _

    X = X + i
    Y = Y + j - 1  'because Y auto. inc. by 1
    i = 1
    j = 1

    End If
    Next j
    Next i

End If
Picture0.PSet (IntX, Y), RGB(255, 255, 255)

Next Y

'The following section is to find the orientation and displacement
'Comparing the theoretical zero point and new zero point
'Calculate the displacement and titled angle

ZeroO_X = (intUpperBoundX - 1) / 2
ZeroO_Y = (intUpperBoundY - 1) / 2

If ((Center_Point_X - ZeroO_X = 0) And (Center_Point_Y - ZeroO_Y = 0)) Then

    Theta = 0
    Displacement = 0

Else

    Displacement = Sqr((Center_Point_X - ZeroO_X) ^ 2 + (Center_Point_Y - ZeroO_Y) ^ 2)
    TempDouble = (Center_Point_Y - ZeroO_Y) / Displacement
    Theta = Atn(TempDouble / Sqr(-TempDouble * TempDouble + 1))
    Angle = 90 - ((Theta / 3.141592654) * 180)

End If

MsgBox ("Titled angle is (clockwise):   " & Angle)
MsgBox ("Displacement is:   " & Displacement)

For X = 0 To intUpperBoundX - 1
    For Y = 0 To intUpperBoundY - 1
        Picture0.PSet (X, Y), RGB(255, 255, 255)
    Next Y
    Next X

For i = 0 To 6

    Picture0.CurrentX = 20
    Picture0.CurrentY = 20 + 15 * i
    Select Case i
    Case 0:
        Picture0.Print ("Number of center lines are:   " & N)
        TempText = "Number of center lines are:   
        Write #1, TempText, N
    Case 1:
        Picture0.Print ("C.L.Intercept = " & CenterLineIntercept)
        Write #1, "C.L.Intercept = ", CenterLineIntercept

30
Case 2:
    Picture0.Print ("C.L.Slope = " & CenterLineSlope)

Form4 - 8

    Write #1, "C.L.Slope = ", CenterLineSlope

Case 3:
    Picture0.Print ("Center X = " & Center_Point_X)
    Write #1, "Center X = ", Center_Point_X

Case 4:
    Picture0.Print ("Center Y = " & Center_Point_Y)
    Write #1, "Center Y = ", Center_Point_Y

Case 5:
    Picture0.Print ("Titled angle is (clockwise):   " & Angle)
    Write #1, "Titled angle is (clockwise):  ", Angle

Case 6:
    Picture0.Print ("Displacement is:   " & Displacement)
    Write #1, "Displacement is   ", Displacement

End Select
Next i
Close #1
End Sub

Private Sub cmdQUIT_Click()

    Unload Form4
    Exit Sub
    ' Form2.Show

End Sub
Private Sub cmdBack_Click()

    Unload Form4
    Form3.Show

End Sub
Private Sub Dir1_Change()

    File1.Path = Dir1.Path

End Sub
Private Sub Drive1_Change()

    Dir1.Path = Drive1.Drive

End Sub
Function GetRed(colorVal As Long) As Integer
    GetRed = colorVal Mod 256
End Function
Function GetGreen(colorVal As Long) As Integer
    GetGreen = ((colorVal And &HFF00FF00) / 256&)
End Function
Function GetBlue(colorVal As Long) As Integer
    GetBlue = (colorVal And &HFF0000) / (256& * 256&)
End Function
Function GetSlope(Points As Integer, LineN As Integer, Choice As Integer) As Double

    Dim SumXY, SumX, SumY As Double
    Dim SumYsq, SumXsq, FuncDumy As Double
    Dim A, Index, Position_X, Position_Y As Integer

    SumXY = 0
SumX = 0
SumY = 0
SumXsq = 0

Form4 - 9

SumYsq = 0
Position_X = 0
Position_Y = 0
Index = 0
FuncDumy = 0

'Sometimes the image is truncated; u do not have 'all the 18 points; we use the B to represent to count 'all the points
'Choice 0: Line correlation coefficient
'Choice 1: Parallel line slope
'Choice 2: Center line slope
'Choice 3: Center line intercept

If (Choice = 0 Or Choice = 1) Then
  For A = 0 To Points - 1
    Position_X = Coord_X(A, LineN)
    Position_Y = Coord_Y(A, LineN)
    If (Position_X <> 0) And (Position_Y <> 0) Then
      SumXY = SumXY + (Position_X * Position_Y)
      SumX = SumX + Position_X
      SumY = SumY + Position_Y
      SumYsq = SumYsq + Position_Y ^ 2
      SumXsq = SumXsq + Position_X ^ 2
      Index = Index + 1
    End If
  Next A
End If

If (Choice = 2 Or Choice = 3) Then
  For A = 0 To Points - 1
    For LineN = 0 To N - 1
      Position_X = Coord_X(A, LineN)
      Position_Y = Coord_Y(A, LineN)
      If (Position_X <> 0) And (Position_Y <> 0) Then
        SumXY = SumXY + (Position_X * Position_Y)
        SumX = SumX + Position_X
        SumY = SumY + Position_Y
        SumYsq = SumYsq + Position_Y ^ 2
        SumXsq = SumXsq + Position_X ^ 2
        Index = Index + 1
      End If
    Next LineN
  Next A
End If

If ((SumX = 0) Or (SumY = 0) Or (SumX * SumY = 0)) Then
  GetSlope = 0
Else
  If (Choice = 1 Or Choice = 2) Then
    GetSlope = ((SumXY) - ((SumX * SumY) / Index)) / ((SumYsq) - ((SumY * SumY) / Index))
  End If

  If (Choice = 3) Then
    FuncDumy = ((SumXY) - ((SumX * SumY) / Index)) / ((SumYsq) - ((SumY * SumY) / Index))
    GetSlope = (SumX - (FuncDumy * SumY)) / Index
  End If

  If (Choice = 0) Then
    FuncDumy = Sqr((SumXsq - (SumX ^ 2 / Index)) * (SumYsq - (SumY ^ 2 / Index)))
    GetSlope = ((SumXY) - ((SumX * SumY) / Index)) / FuncDumy
  End If

End If
End If
End If

End Function
Function dblSquare(SquareMe As Integer) As Double

Form4 - 10

dblSquare = SquareMe ^ 2 "* SquareMe

End Function
Function Measure_Distance(c1 As Double, m1 As Double, Point2_X As Integer, Point2_Y As Integer) As Double

Dim Point1_X, Point1_Y As Long
Dim c2 As Long

c2 = Point2_X - ((-1 / m1) * Point2_Y)
Point1_X = (c2 * m1 - c1 * (-1 / m1)) / (m1 - (-1 / m1))
Point1_Y = (c2 - c1) / (m1 - (-1 / m1))
Measure_Distance = Sqr((Point2_X - Point1_X)^2 + (Point2_Y - Point1_Y)^2)

End Function

End Sub
Form5 - 1

Private Sub Back_Click()
Unload Form5
Form4.Show
End Sub

Private Sub Picture2_Click()
End Sub

Private Sub Quit_Click()
Unload Form5
Exit Sub
End Sub

Private Sub ShowRes_Click()
Dim NewLine As String

On Error GoTo FileError
Open "c:\windows\desktop\InspResults.txt" For Input As #1
Do Until EOF(1)
    Line Input #1, NewLine
    TEXT1.Text = TEXT1.Text + NewLine + vbCrLf
Loop

Exit Sub

FileError:
    MsgBox "File Error!"
End Sub
Form1 - 1

Private Sub Timer1_Timer()

    Dim PauseTime, Start
    PauseTime = 2 ' Set duration.
    Start = Timer ' Set start time.
    Do While Timer < Start + PauseTime
        DoEvents ' Yield to other processes.
        Loop
    Unload Me
    Form2.Show

End Sub
VERSION 5.00
Begin VB.Form Form1
   Caption         =   "HMD TESTER"
   ClientHeight    =   4140
   ClientLeft      =   60
   ClientTop       =   345
   ClientWidth     =   7890
   LinkTopic       =   "Form1"
   ScaleHeight     =   4140
   ScaleWidth      =   7890
   StartUpPosition =   3  'Windows Default
Begin VB.Frame Frame1
   Height          =   4050
   Left            =   0
   TabIndex        =   0
   Top             =   0
   Width           =   7905
Begin VB.Timer Timer1
   Interval        =   1000
   Left            =   6960
   Top             =   3360
End
Begin VB.Label lblCompanyProduct
   AutoSize        =   -1  'True
   Caption         =   "US AARL"
   BeginProperty Font
      Name            =   "Arial"
      Size            =   18
      Charset         =   0
      Weight          =   700
      Underline       =   0  'False
      Italic          =   0  'False
      Strikethrough   =   0  'False
   EndProperty
   Height          =   435
   Left            =   3240
   TabIndex        =   8
   Top             =   600
   Width           =   1590
End
Begin VB.Label lblLicenseTo
   Alignment       =   1  'Right Justify
   Caption         =   "******"
   BeginProperty Font
      Name            =   "Arial"
      Size            =   8.25
      Charset         =   0
      Weight          =   400
      Underline       =   0  'False
      Italic          =   0  'False
      Strikethrough   =   0  'False
   EndProperty
   Height          =   255
   Left            =   3960
   TabIndex        =   7
   Top             =   360
   Width           =   3495
End
Begin VB.Label lblProductName
   AutoSize        =   -1  'True
   BackColor       =   &H80000018&
   Caption         =   "HMD TESTER"
   BeginProperty Font
      Name            =   "Arial"
      Size            =   32.25
   EndProperty
Charset = 0
Weight = 700
Underline = 0 'False
Italic = 0 'False
Strikethrough = 0 'False
EndProperty
Height = 765
Left = 3240
TabIndex = 6
Top = 1140
Width = 4245
End

Begin VB.Label lblPlatform
Alignment = 1 'Right Justify
AutoSize = -1 'True
Caption = "Platform: PC"
BeginProperty Font
Name = "Arial"
Size = 15.75
Charset = 0
Weight = 700
Underline = 0 'False
Italic = 0 'False
Strikethrough = 0 'False
EndProperty
Height = 360
Left = 4950
TabIndex = 5
Top = 2340
Width = 1905
End

Begin VB.Label lblVersion
Alignment = 1 'Right Justify
AutoSize = -1 'True
Caption = "Version: 1.0"
BeginProperty Font
Name = "Arial"
Size = 12
Charset = 0
Weight = 700
Underline = 0 'False
Italic = 0 'False
Strikethrough = 0 'False
EndProperty
Height = 285
Left = 5490
TabIndex = 4
Top = 2700
Width = 1365
End

Begin VB.Label lblWarning
Caption = "Supported by US AARL and Army Summer Faculty Research Program"
BeginProperty Font
Name = "Arial"
Size = 8.25
Charset = 0
Weight = 400
Underline = 0 'False
Italic = 0 'False
Strikethrough = 0 'False
EndProperty
Height = 195
Left = 150
TabIndex = 3
Top = 3660

Form1 - 3

    Width    =   6855
End
Begin VB.Label lblCompany
    Caption         =   "August, 1999"
    BeginProperty Font
        Name            =   "Arial"
        Size            =   8.25
        CharSet         =   0
        Weight          =   400
        Underline       =   0 'False
        Italic          =   0 'False
        Strikethrough   =   0 'False
    EndProperty
    Height          =   255
    Left            =   4560
    TabIndex        =   2
    Top             =   3270
    Width           =   2415
End
Begin VB.Label lblCopyright
    Caption         =   "Sheng-Jen (""Tony"") Hsieh, Ph.D."
    BeginProperty Font
        Name            =   "Arial"
        Size            =   8.25
        CharSet         =   0
        Weight          =   400
        Underline       =   0 'False
        Italic          =   0 'False
        Strikethrough   =   0 'False
    EndProperty
    Height          =   255
    Left            =   4560
    TabIndex        =   1
    Top             =   3060
    Width           =   2415
End
Begin VB.Image imgLogo
    Height          =   3105
    Left            =   240
    Picture         =   "820F1.frx":0000
    Stretch         =   -1 'True
    Top             =   360
    Width           =   2655
End
End
End
Form2 - 1

Private Sub cmdQUIT_Click()
Unload Form2
End
End Sub

Private Sub Command2_Click() 'Image Analysis
    Unload Form2
    Form4.Show
End Sub
Form2 -2

Begin VB.CommandButton Results
Caption = "Results"
BeginProperty Font
    Name = "Comic Sans MS"
    Size = 8.25
    CharSet = 0
    Weight = 400
    Underline = 0 'False
    Italic = 0 'False
    Strikethrough = 0 'False
EndProperty
Height = 375
Left = 2880
TabIndex = 1
Top = 3720
Width = 1215
End

Begin VB.CommandButton Command4
Caption = "Help"
BeginProperty Font
    Name = "Comic Sans MS"
    Size = 8.25
    CharSet = 0
    Weight = 400
    Underline = 0 'False
    Italic = 0 'False
    Strikethrough = 0 'False
EndProperty
Height = 375
Left = 4200
TabIndex = 0
Top = 3720
Width = 1215
End

Begin VB.Image Image1
Height = 4140
Left = 0
Picture = "820F2.frx":0000
Stretch = -1 'True
Top = 0
Width = 6240
End

End
From3 - 1

Private Sub Continue_Click()
Unload Form3
Form4.Show
End Sub

Private Sub Quit_Click()
Unload Form3
Form2.Show
End Sub
VERSION 5.00
Begin VB.Form Form3
    AutoRedraw = -1 "True"
    Caption = "Image Capture"
    ClientHeight = 3690
    ClientLeft = 60
    ClientTop = 345
    ClientWidth = 7605
    BeginProperty Font
        Name = "Comic Sans MS"
        Size = 8.25
        Charset = 0
        Weight = 400
        Underline = 0 'False
        Italic = 0 'False
        Strikethrough = 0 'False
    EndProperty
    LinkTopic = "Form3"
    Picture = "820F3.frx":0000
    ScaleHeight = 3690
    ScaleWidth = 7605
    StartUpPosition = 3 'Windows Default
    Begin VB.CommandButton Continue
        Caption = "Continue"
        Height = 375
        Left = 3600
        TabIndex = 8
        Top = 3240
        Width = 855
    End
    Begin VB.TextBox Help
        Alignment = 2 'Center
        BorderStyle = 0 'None
        DragMode = 1 'Automatic
        Height = 285
        Left = 6960
        TabIndex = 6
        Text = "Help"
        Top = 2880
        Width = 495
    End
    Begin VB.TextBox Text2
        Alignment = 2 'Center
        BorderStyle = 0 'None
        Height = 285
        Left = 5880
        TabIndex = 5
        Text = "User Manual"
        Top = 2880
        Width = 975
    End
    Begin VB.TextBox Capture
        Alignment = 2 'Center
        BorderStyle = 0 'None
        Height = 285
        Left = 4560
        TabIndex = 4
        Text = "View & Capture"
        Top = 2880
        Width = 1215
    End
    Begin VB.CommandButton Quit
End
Form3 - 2

Caption        = "Quit"
Height         = 375
Left           = 2400
TabIndex       = 0
Top            = 3240
Width          = 735
End

Begin VB.Label Label1
    BackColor   = &H00FFFFFF&
    Caption     = "Double click to activate"
    Height      = 255
    Left        = 5400
    TabIndex    = 7
    Top         = 2520
    Width       = 1815
End

Begin VB.OLE OLE3
    BackColor    = &H00C0C0C0&
    Class        = "Package"
    DisplayType  = 1 'Icon
    Height       = 375
    Left         = 6960
    OleObjectBlob = "820F3.frx":99C4
    SourceDoc    = "C:\Program Files\MRT micro\MRT VideoPort Professional\User Manuals\Edoc16.hlp"
    TabIndex     = 3
    Top          = 3240
    Width        = 615
End

Begin VB.OLE OLE2
    BackColor    = &H00C0C0C0&
    DisplayType  = 1 'Icon
    Height       = 375
    Left         = 5880
    OleObjectBlob = "820F3.frx":41FDC
    SourceDoc    = "C:\WINDOWS\twain\Camdrv80\Camdrive.hlp"
    TabIndex     = 2
    Top          = 3240
    Width        = 975
End

Begin VB.OLE OLE1
    BackColor    = &H00C0C0C0&
    Class        = "IMWizard"
    DisplayType  = 1 'Icon
    Height       = 375
    Left         = 4560
    OleObjectBlob = "820F3.frx":435F4
    SourceDoc    = "C:\Program Files\MRT micro\MRT VideoPort Professional\Image Wizard\Ripple.tif"
    TabIndex     = 1
    Top          = 3240
    Width        = 1215
End

Begin VB.Image Image1
    Height       = 3705
    Left         = 240
    Picture      = "820F3.frx":44C0C
    Top          = 0
    Width        = 8070
End

End