AN ABSTRACT OF THE THESIS OF

Zhao-ming Song for the degree of Master of Science in Industrial and Manufacturing Engineering Presented on January 30, 1991.

Title: Package Inspection with a Machine Vision System

Abstract approved: Redacted for Privacy

Dr. Eugene Fichter

Machine Vision has been extensively applied in industry. This thesis project, which originated with a local food processor, applies a vision system to inspection of packages for cosmetic errors. The basic elements and theory of the machine vision system are introduced, and some image processing techniques, such as histogram analysis, thresholding, and SRI algorithm, are utilized in this thesis. Computer programs written in C and Pascal are described. Hardware setup and computer interface, such as RS-232 serial interface, parallel digital I/O interface, conveyor control, and incremental shaft encoder, are described. Test results are presented and discussed.
Package Inspection with a Machine Vision System

by

Zhao-ming Song

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of
Master of Science

Completed January 30, 1991
Commencement June, 1991
Date thesis is presented  January 30, 1991
Typed by Zhao-ming Song for Zhao-ming Song
This work is dedicated to
my parents Juxing and Keqiang Song
ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to my major professor Dr. Eugene Fichter and his wife Dr. Becky Fichter for their guidance, encouragement and support throughout the course of my graduate study. Their patience and understanding as well as expertise in the field made this thesis possible.

I would also like to thank Dr. Sabah Randhawa, Dr. Logen Logendran and Dr. Clifford Gray for contributing their time and expertise while serving on my graduate committee.

It is a pleasure to acknowledge with special thanks my friends, Mr. Johan Forrer, Mr. Yimin Zeng, Mr. Tony Chou, and Ms. Yuan Zhong, for their valuable suggestions and help. I am deeply indebted to my parents for their love and support.
# TABLE OF CONTENTS

Chapter 1  Introduction ............................................. 1
  1.1  Machine Vision Application in Industry .............. 1
  1.2  Introduction to Machine Vision ..................... 2
    1.2.1 Basic Elements ................................. 2
    1.2.2 Digital Image ................................ 3
    1.2.3 Resolution .................................... 4
  1.3  Thesis Origination and Structure .................... 5

Chapter 2  Vision System .......................................... 7
  2.1  Vision System Composition ........................... 7
  2.2  Digitizer ............................................. 7
  2.3  RS-232 Serial Interface ............................ 8
  2.4  Light Source ......................................... 8
  2.5  Image Processing .................................... 10
    2.5.1  Histogram ..................................... 11
    2.5.2  Thresholding .................................. 13
    2.5.3  SRI Algorithm ................................ 14

Chapter 3  Electrical and Mechanical Construction .... 18
  3.1  Parallel Digital I/O Interface ................... 18
  3.2  Conveyor Control .................................. 20
    3.2.1  Motor Control ................................ 21
    3.2.2  Incremental Shaft Encoder ................... 21
  3.3  Defective Package Rejection ....................... 25

Chapter 4  Software .............................................. 26
  4.1  Function Initialization ............................ 26
  4.2  Function Conveyor .................................. 26
  4.3  Program Vision ..................................... 28
  4.4  Function Histogram ................................ 28
  4.5  Function Elong ..................................... 28

Chapter 5  Testing and Conclusion ......................... 29
  5.1  Testing Results ..................................... 29
  5.2  Future Expansion ................................... 32
  5.3  Conclusion ........................................... 34

Bibliography ..................................................... 35

Appendices ......................................................... 37
  Appendix A  Program INSPECT and VISION ............ 37
  Appendix B  Procedures Supplied with ImageWize
              Digitizer .................................... 48
  Appendix C  Shaft Encoder Counter Circuit .......... 66
  Appendix D  Equipment List ............................. 78
LIST OF FIGURES

1.1 Basic Elements of a Machine Vision System ... 3
1.2 Package Inspection with a Machine Vision System .... 5
2.1 RS-232 Serial Interface Cable Connecting ... 9
2.2 An Image without Filtering ... 9
2.3 An Image with Filtering ... 10
2.4 An Image the Digitizer Transfers to the Computer ... 10
2.5 Histogram of a Good Package Image ... 12
2.6 Histogram of a Bad Package Image ... 12
2.7 The Histogram from Figure 2.6 with Expanded Vertical Scale ... 13
2.8 An Image after Thresholding ... 14
3.1 Write-only Control Register ... 19
3.2 Base Address Switch Setting for 280 HEX ... 19
3.3 Address Map for P.P.I. Register ... 20
3.4 Conveyor Motor Control ... 20
3.5 Example of Optically Isolated I/O Connections ... 22
3.6 Motor Control Circuit Using Relay R2 for On/Off Control and Relay R1 to Reverse Motor ... 23
3.7 Incremental Encoder Output ... 24
3.8 Diversion Gate ... 25
4.1 Flow Chart of Program INSPECT ... 27
5.1 Package Positions in System Testing ... 31
LIST OF TABLES

Table 5.1 Testing Records ................. 30
Table 5.2 Two-by-two Contingency Table ....... 33
Chapter 1  Introduction

1.1 Machine Vision Application in Industry

The purpose of this thesis project is to use a vision system to inspect packages for cosmetic errors. Machine vision originated in the 1960's. It was mainly applied in very high altitude aerial photography and satellite images at that time. Its industrial application started in the 1970's[3], and has been developing very quickly in a short period of 20 years. Many companies in the United States offer some type of machine vision product or service, and more and more companies are using machine vision systems to carry out various tasks[2], such as inspection of products, inventory control and parts counting, sorting, safety and surveillance, character recognition, process control, and robot control.

Machine vision has many advantages compared with human eyes. It will continue working round the clock and will not tire. It can work in conditions which would be very unpleasant or impossible for a human operator. It can take dimensional measurements more accurately than a person can estimate by eye. It can give an objective measure of other variables such as color, which an inspector could only assess subjectively, and it can deal with infrared and ultraviolet that can not be sensed by human eyes. It is these advantages
that push machine vision theory and application forward.

Although machine vision is used in many industrial fields, there are three principal application areas. The first, also the most important, is inspection including not only visual examination for defects, but also measurement of dimensions or other characteristics, counting, checking of orientation, and so on. The second area is recognition of an object from its shape; this can simplify certain manufacturing operations. The third, and most rapidly growing application area, is guidance including control of robots, vehicles, orientation devices, and so on.

1.2 Introduction to Machine Vision

The definition of "machine vision" adopted by the Machine Vision Association of the Society of Manufacturing Engineers is: "the use of devices for optical non-contact sensing to automatically receive and interpret an image of a real scene, in order to obtain information and/or control machines or process."[3] According to this definition, a machine vision system at least has to carry out three tasks: 1) capture an image of a real object; 2) process the image to analyze it quantitatively; 3) make a decision to control machines.

1.2.1 Basic Elements

The basic elements of a machine vision system are shown in Figure 1.1. The camera, which serves as the sensor of the
system, captures an image of the object. The digitizer converts the analog signals of the image into an array of digital data. The computer processor analyzes the data, then makes necessary decisions. In a machine vision system, lighting is one of the most important aspects since it directly influences the captured image.

1.2.2 Digital Image

In the digitizing process, the entire image is automatically divided into a mesh of tiny squares called pixels, each of which is labeled with a pair of coordinates - one defining the row that it is in and one the column. Row numbers range from 0, at the top, to m-1 where m is the number of rows in the image. Likewise, column numbers range from 0, at the left side, to n-1 where n is the number of columns in the image.
1.2.3 Resolution

When digitizing an image, there is always the question of how good the representation is when compared with the original. Quality of the digitized image is limited by sensor resolution. In image processing, resolution may be broken into two definite types, spatial and brightness.

Spatial resolution is the number of pixels a digital image is divided into. The finer this resolution is, the closer the spatial appearance of the original image is approached. But, the size of the pixel array is limited by the speed of the process, if spatial resolution is m by n, the processor must handle mxn brightness values. A typical size in practical machine vision system is 256x256.

Brightness resolution is the number of different brightness values that can be represented. Quantization converts an analog brightness level at a sample point to a numeric value within a certain range. For instance, conversion to an 8-bit binary number allows each pixel to be represented by one of 256 brightness levels. The 256 levels of brightness comprise what is called a gray scale, or the 8-bit gray scale.

If the digital representation of an image is given by a discrete function P(i,j), where 0<=i<=m, and 0<=j<=n, then the resolution of the digital image is given by the values m and n and the number of bits by which the gray scale values of P(i,j) are represented. Once these concepts are understood,
we can start to study how to deal with all the information that is input to a computer processor.

1.3 Thesis Origination and Structure

This thesis project originated with a local food processor. Where, granular food is loaded into plastic packages by machine, and human inspectors must monitor packages for cosmetic errors. A machine vision system was assembled to perform this inspection task. The whole system is shown in Figure 1.2. The conveyor transfers a package to an appropriate position for the camera to capture an image.
The digitizer converts the image into a pixel array which is input to the computer. After data analysis, the computer controls a gate which directs defective packages out of the system.

The remainder of this thesis is divided into five chapters. Chapter 2 describes the vision system. In chapter 3, parallel digital I/O interface, conveyor control and diversion gate are discussed. Chapter 4 describes software for image processing and control. In chapter 5, test data are shown and discussed.
Chapter 2 Vision System

2.1 Vision System Composition

A RCA TV camera with a KODAK lens (f:2.7, 15mm) is used as system sensor. Two 25W lamps with filters constitute system light source. The 9" TV monitor is also made by RCA company. A digitizer made by Circuit Cellar Inc. is used to capture an image and send it to an AST Premium/286 personal computer for processing. The image background is black paper in order to enhance contrast.

2.2 Digitizer

A digitizer converts an image into a numerical representation suitable for input into a digital computer. The most commonly used input devices are TV camera digitizers. In TV cameras the image is focused directly on the surface of a photosensitive device whose response is proportional to the incident light pattern.

In this thesis project, image digitizing and transmission are carried out by ImageWise Digitizer. It is a true "frame grabber" which takes only 1/60th of a second to capture an image. It accepts the video signals from a standard television camera, and stores the picture as 244 lines of 256 pixels with 64 levels in the gray scale (244x256x6 bits). The
digitizer board converts the stored video image to RS-232 serial data which are transmitted at the rate of 28.8 Kbps (thousand bits per second) to the computer.

2.3 RS-232 Serial Interface

RS-232 serial interface is very commonly used for computer communication. The digitizer board is connected to the personal computer through the COM1 serial port. A "null modem" cable is necessary so that the data and handshaking signals are properly routed between the digitizer board and the computer. The cable has 7 wires and 25 pin connectors on both ends. Most of the 25 pins are never used even though they are defined by the standard. The cable configuration is shown on Figure 2.1. The data format parameters are set as follows.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data bits</td>
<td>8</td>
</tr>
<tr>
<td>Stop bit</td>
<td>1</td>
</tr>
<tr>
<td>Parity</td>
<td>NO</td>
</tr>
</tbody>
</table>

The transmission speed is 28.8 Kbps (thousand bits per second).

2.4 Light Source

Lighting plays a very important role in a vision system. Position, color, and luminance of light sources directly affect the resulting image. Wrapping material's outer surface of the package to be inspected is yellow and smooth. If the
<table>
<thead>
<tr>
<th>Computer</th>
<th>Digitizer</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td>Transmitted Data</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>Received Data</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Request to Send</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>Clear to Send</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>Data Set Ready</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>Data Terminal Ready</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>Signal Ground</td>
</tr>
</tbody>
</table>

Figure 2.1 RS-232 Serial Interface Cable Connecting

package is not processed well, wrapping material’s inside aluminum foil surface will appear on the upper edge. In order to make aluminum foil reflect more strongly than the other part of the package, a piece of white paper and a blue transparency are used as filters in front of the lamps. White paper diffuses the light, and blue transparency changes the light color because yellow color absorbs blue rays. Figure 2.2 and Figure 2.3 show the difference of the images with and without filters. Both pictures are of the same package.

Figure 2.2 An Image without Filtering
Without filtering, the foil line is hard to identify, but with filtering, a white line on the upper edge of the package can be found more easily. For proper system operation, ambient light in this system must be lower than 10 foot-candles.

2.5 Image Processing

The image transferred from the digitizer to the computer is shown in Figure 2.4. There are a total of 244x256 pixels each of which has a gray scale from 0 to 63. Hence, the computer must handle 62,464 brightness values. However, we are actually only interested in the upper part of the package as shown in Figure 2.3. It would obviously save a lot of time if the computer only operates on the pixels related this part. So, the image processing in this system deals only with line 20 to line 100 of the pixel array. This means that the number of brightness
values which have to be handled is reduced to 20,736. That greatly increases the process speed. Another advantage is that errors caused by reflections from the rest of the package are avoided.

### 2.5.1 Histogram

A gray-level histogram of an image is a function that gives the frequency of occurrence of each gray level in the image. Where the gray levels are quantified from 0 to $k$, the value of the histogram at a particular gray level $p$, denoted $h(p)$, is the number or fraction of pixels in the image with that gray level. Figure 2.5 and Figure 2.6 show histograms for good and bad packages respectively.

A histogram is useful in many different ways. It gives us a convenient, easy-to-read representation of the concentration of pixels versus brightness in an image. Using this graph we are able to see immediately whether the image is basically dark or light, and whether contrast is high or low. Furthermore, it gives us a first clue as to what contrast enhancements would be appropriately applied to make the image more subjectively pleasing to an observer. Comparing Figure 2.5 with Figure 2.6, we find that the package with flaws has some high gray level pixels because the foil has higher brightness value. That gives us a way to separate flawed packages from good ones.
Figure 2.5 Histogram of a Good Package Image

Figure 2.6 Histogram of a Bad Package Image
2.5.2 Thresholding

In order to detect the flaw, we must be able to distinguish it from the surface on which it lies. A basic method of doing this is thresholding, i.e., the process of distinguishing light pixels from dark ones. Thresholding converts a gray level image to a binary image — only white and black. The gray level $p$ of each pixel is compared with a "threshold" $t$; if $p \geq t$, the pixel is light; if $p < t$, it is dark. So, in thresholding, it is most important to decide the threshold value $t$.

Since we are more interested in the lighter (higher) gray levels of the histogram, scale up the vertical axis of the

![Figure 2.7](image_url)
histogram in Figure 2.6 between $h(p)=0$ and $h(p)=600$ into Figure 2.7. Three "valleys" are found around $p=3$, $p=14$, and $p=30$. Usually, the gray level at the bottom of the "valley" will be chosen as threshold value $t$. Since we already know from Figure 2.5 and Figure 2.6 that foil flaws contribute the highest gray scale level, in this system, we choose $t=30$.

After thresholding, the value of a pixel can be represented by a single bit, i.e., 1 for light; 0 for dark. Figure 2.8 shows the image in Figure 2.4 after thresholding. A foil flaw is very clearly detected.

![Figure 2.8 An Image after Thresholding](image)

### 2.5.3 SRI Algorithm

It is possible to get wrong information from the pixel array after thresholding. Sometimes light reflects from another part of the package and there is a bright area but no foil flaw. However, the shape of bright area in this case is not a narrow line as in Figure 2.8. The SRI algorithm, a simple and commonly used algorithm, can effectively distinguish different shapes.

The SRI algorithm was developed at the Stanford Research Institute in the late 1970's. Its input is a pixel array whose values have been converted from gray level to binary, that is, white and black. The output of the SRI algorithm is
a set of parameters. Some definitions we will use later are listed here.

- **Area**: number of ones in the pixel array;
- **Coordinates of the Centroid**: centroid of area of the ones in the pixel array;
- **Principal Axes**: axes about which the moments of inertia are maximum and minimum. They intersect at the centroid and are at right angles with each other.
- **Principal Moments of Inertia**: the largest and smallest moments of Inertia about lines through the centroid. A measure of the distribution about the principal axes of the area of ones in the pixel array.
- **Elongation**: a measure of how long and narrow the area of ones in the pixel array is. A circle has a small value of elongation while an ellipse of the same area has a larger value of elongation.

Each of the SRI output parameters can be determined by doing a relatively simple series of additions and multiplications on the input array. A set of preliminary calculations are done to determine area and moment histograms of the pixel array in the x and y directions. Let pixel array indices be i in the y direction and j in the x direction. Index i ranges from 0 to m, and index j ranges from 0 to n.
The elements of the area histograms $A_x$ and $A_y$ in the x and y directions respectively are calculated using the equations below.

$$A_x(j) = \sum_{i=0}^{m} P(i,j) \quad (j=0,1,...,n)$$  \hspace{1cm} (1)

$$A_y(i) = \sum_{j=0}^{n} P(i,j) \quad (i=0,1,...,m)$$  \hspace{1cm} (2)

Here the term $P(i,j)$ is the element of the pixel array in the $j$th column and the $i$th row.

The elements of $M_{xx}$, one of the moment histograms, are calculated below.

$$M_{xx}(j) = \sum_{i=0}^{m} [(i+1)P(i,j)] \quad (j=0,1,...,n)$$  \hspace{1cm} (3)

Area is given by the equation below.

$$A = \sum_{j=0}^{n} A_x(j)$$  \hspace{1cm} (4)

The coordinates of the centroid are given by the equations below.

$$C_x = \frac{1}{A} \sum_{j=0}^{n} [(j+1)A_x(j)]$$  \hspace{1cm} (5)

$$C_y = \frac{1}{A} \sum_{i=0}^{m} [(i+1)A_y(i)]$$  \hspace{1cm} (6)

The equation for elongation is based on moments of
inertia \((a\text{ and } c)\) and the product of inertia \(b\) about the horizontal and vertical axes through the centroid. Equations for these parameters are given below.

\[
a = \left\{ \sum_{j=0}^{n} [((j+1)^2 A_{x}(j))] - C_{x}^{2}A \right\}
\]

\[
b = 2\left\{ \sum_{j=0}^{n} [(j+1)M_{xx}(j)] - C_{x}C_{y}A \right\}
\]

\[
c = \left\{ \sum_{i=1}^{m} [(i+1)^2 A_{y}(i)] - C_{y}^{2}A \right\}
\]

Elongation is given by the equation below.

\[
E = \frac{\sqrt{b^2 + (a-c)^2}}{a+c}
\]

Using elongation, we can easily tell foil defects from other reflections. If the image comes from a package with foil flaws, \(E \geq 0.9\).
Chapter 3 Electrical and Mechanical Construction

3.1 Parallel Digital I/O Interface

Besides the vision system, there are also auxiliary devices in the installation. Two main sections are conveyor control and defective package rejection. Both of these require interface to the control computer.

A MetraByte Data Acquisition and Control board is plugged into an expansion slot in the computer. It provides 24 TTL/DTL compatible digital I/O lines, interrupt input and enable lines and external connections to the IBM PC bus power supplies(+5V, -5V, +12V, -12V). The 24 digital I/O lines are provided through an 8255-5 programmable peripheral interface (P.P.I.) divided into three ports, an 8 bit PA port, an 8 bit PB port, and 8 bit PC port. The PC port may also be used as two half ports of 4 bits, PC upper(PC 4-7) and PC lower(PC 0-3). Each of the ports and half ports may be configured as an input or an output by software control according to the contents of a write only control register (see Figure 3.1) in the P.P.I.

The 8255-5 P.P.I. uses 4 I/O address locations which are fully decoded within the I/O address space of the IBM PC. The base address is set by an 8 position DIP switch and can in theory be placed anywhere in I/O address space. In our case,
the base address is set at 280 hexadecimal (see Figure 3.2). The address map for the P.P.I. register is shown in Figure 3.3.

*Switches have decimal values as above in the "OFF" position. In the "ON" position decimal value is zero.

Figure 3.2 Base Address Switch Setting for 280 HEX (Courtesy of MetraByte Corporation)
3.2 Conveyor Control

A conveyor control block diagram is shown in Figure 3.4. The computer sends a signal through the parallel digital I/O interface to turn on the conveyor motor. The shaft of the motor is connected to an incremental optical encoder. The encoder is connected to a counter which can be read by the computer through the parallel digital I/O interface.

![Figure 3.4 Conveyor Motor Control](image-url)
3.2.1 Motor Control

An optically isolated I/O mounting rack made by OPTO-22 is used to control the conveyor motor. All eight positions of the optically isolated I/O mounting rack may be utilized as either inputs or outputs. The input modules must have a positive DC voltage (3 – 32V) and DC return applied to the terminals to activate them (see Figure 3.5b). Positive voltage should be applied to the odd numbered terminal and ground to the even numbered terminal. When wiring the output modules to control a load, positive voltage is applied to the odd numbered terminal and the load to the even numbered terminal (see Figure 3.5a). The terminals can be asserted by cabling the power modules to the parallel digital I/O interface mounted in the computer. Making computer output high will turn on the load.

In conveyor motor control, two output modules and two relays are used as shown in Figure 3.6. When output module 2 is off the motor stops. When output module 2 is on the motor runs. If output module 1 is on the motor rotates clockwise; if output module 1 is off the motor rotates counter-clockwise.

3.2.2 Incremental Shaft Encoder

Another important element is the optical encoder which measures how far the conveyor moves. Optical encoders can be classified into two types: incremental and absolute position encoders. Each has specific applications for which it is best
Figure 3.5 Example of Optically Isolated I/O Connections (Courtesy of Microbot, Inc.)
Figure 3.6 Motor Control Circuit Using Relay R2 for On/Off Control and Relay R1 to Reverse Motor

suited.

The incremental shaft encoder in the conveyor motor control is made by Datametrics Inc. The incremental encoder consists of a glass disc imprinted with one circular row of slots, all the same size and distance apart. Two sensors are focused on the row of slots with a light source on the opposite side of the disc. When the disc rotates, the encoder creates a series of equally spaced signals corresponding to the angular increment between slots on the disc, e.g. a disc with 1000 slots will produce 1000 square wave cycles per revolution. By using a counter to count those cycles, we can tell how far the shaft rotated. 100 counts would equal 36 degrees, 150 counts 54 degrees, etc. The encoder that is used
provides only 50 cycles per revolution. However this resolution is high enough in this application.

Most incremental systems use two output channels in quadrature for position sensing (see Figure 3.7). This allows us to count the transitions and to view the state of the opposite channel during these transitions. Using this information, we can determine if "A" leads "B" or vice versa, and thus derive direction.

![Figure 3.7 Incremental Encoder Output](image)

The schematic of the counter circuit designed by Dr. Eugene Fichter is shown in Appendix C. The output of the counter is connected to the parallel digital I/O interface input port A. The Load and Reset terminals are connected to pin 0 and pin 1 of the parallel digital I/O interface output port C. The sequence with which the counter works is

Start-up: 1) set LOAD high
2) run device to zero position
3) set RESET low then high

Read: 1) set LOAD low
2) read input port
3) set LOAD high

This counter can count up to 255 cycles.

3.3 Defective Package Rejection

After image processing, the computer can determine whether the package inspected is good or has foil flaws. As soon as a defective package is found, the computer will send a signal through the parallel digital I/O interface to pin 6 of output port B, thus module 3 of the optically isolated I/O mounting rack will be turned on and a rotary solenoid (Ledex Inc.) is energized.

A drawing of the diversion gate is shown in Figure 3.8. When the rotary solenoid is energized, the shaft turns 22°. The defective package will be pushed off the conveyor.
Chapter 4  Software

The software is written in QuickC and Turbo Pascal. The main program "INSPECT" calls four functions (QuickC) and one program (Turbo Pascal). The flow chart of "INSPECT" is shown in Figure 4.1, and program code is in Appendix A. Four functions and one Pascal program are introduced as follows.

4.1 Function Initialization

The function Initialization sets up parallel digital I/O interface port addresses, and control port states, resets the conveyor motor to off and the incremental encoder counter to 0. It prompts the user to enter thresholding level and elongation criterion, and returns an value that tells the main program to keep running or to stop.

4.2 Function Conveyor

The function conveyor always turns the conveyor motor on first, moves inspected package out of the inspection area, and sends a new package into the inspection area. Then it sets the diversion door open or closed depending on an input value from main program. While the motor runs, the program reads encoder counter input repeatedly accumulating a total encoder count. When the count reaches 768, the computer turns the
Call Function Initialization
Set D True and Open Gate

Run System?

Is D True?

N → Close Gate

Y →

Call Function Conveyor
Open Gate

Call Program Vision

Call Function Histogram

Call Function Elong

Elongation < Criterion

N → Set D False

Y →

Set D True

Continue Running?

N → Stop

Y →

Figure 4.1 Flow Chart of Program INSPECT
motor off.

4.3 Program Vision

The procedures that capture and expand digitized pictures created by the digitizer board are supplied by the Circuit Cellar Inc. with the ImageWise Digitizer. Since these procedures are written in Turbo Pascal, the program Vision has to use the same language. It is mainly used to call these procedures so that the image can be captured and expanded, and then saved into the file "PACKAGE.DAT".

4.4 Function Histogram

This function is used to show a histogram of the image. It is not necessary for system operation, but is very useful for the user to choose thresholding value.

4.5 Function Elong

Function Elong has two tasks. One is thresholding; the other is calculating elongation. It reads image pixel values from data file "PACKAGE.DAT", then performs thresholding. The thresholding level is an input variable. After thresholding, it calculate Area (the number of 1's in the pixel array). If Area is equal to 0, the package will be accepted. Otherwise it calculates and returns elongation. If this value is greater than elongation criterion, the package inspected is identified as a defective product.
Chapter 5 Testing and Conclusion

5.1 Testing Results

31 packages were tested in different positions. The results are listed in Table 5.1. Position 1, 2, and 3 are shown in Figure 5.1. The left columns under the position 1, 2, and 3 show "A" if the inspection system accepts the packages, or "R" if it rejects the packages. The right column shows "T" if the vision system accepts a good package or rejects a bad package, and "F" if it accepts a bad package or rejects a good package. If accuracy is defined as

\[ A = \frac{\text{The number of samples} - \text{The number of F's}}{\text{The number of Samples}} \times 100\%, \]

we can calculate accuracies of testing with different package positions 1, 2, and 3

\[ A_1 = \frac{31-1}{31} \times 100\% = 96.77\% \] (Position 1);

\[ A_2 = \frac{31-2}{31} \times 100\% = 93.55\% \] (Position 2);

\[ A_3 = \frac{31-2}{31} \times 100\% = 93.55\% \] (Position 3);

and total accuracy
Table 5.1 Testing Records

<table>
<thead>
<tr>
<th>Package #</th>
<th>Real Condition</th>
<th>Position 1</th>
<th>Position 2</th>
<th>Position 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bad</td>
<td>R T</td>
<td>R T</td>
<td>R T</td>
</tr>
<tr>
<td>2</td>
<td>Good</td>
<td>A T</td>
<td>A T</td>
<td>A T</td>
</tr>
<tr>
<td>3</td>
<td>Good</td>
<td>A T</td>
<td>A T</td>
<td>A T</td>
</tr>
<tr>
<td>4</td>
<td>Bad</td>
<td>R T</td>
<td>R T</td>
<td>R T</td>
</tr>
<tr>
<td>5</td>
<td>Good</td>
<td>A T</td>
<td>A T</td>
<td>A T</td>
</tr>
<tr>
<td>6</td>
<td>Bad</td>
<td>R T</td>
<td>R T</td>
<td>R T</td>
</tr>
<tr>
<td>7</td>
<td>Bad</td>
<td>R T</td>
<td>R T</td>
<td>R T</td>
</tr>
<tr>
<td>8</td>
<td>Bad</td>
<td>A F</td>
<td>A F</td>
<td>A F</td>
</tr>
<tr>
<td>9</td>
<td>Good</td>
<td>A T</td>
<td>A T</td>
<td>A T</td>
</tr>
<tr>
<td>10</td>
<td>Bad</td>
<td>R T</td>
<td>R T</td>
<td>R T</td>
</tr>
<tr>
<td>11</td>
<td>Good</td>
<td>A T</td>
<td>A T</td>
<td>A T</td>
</tr>
<tr>
<td>12</td>
<td>Good</td>
<td>A T</td>
<td>A T</td>
<td>A T</td>
</tr>
<tr>
<td>13</td>
<td>Bad</td>
<td>R T</td>
<td>R T</td>
<td>R T</td>
</tr>
<tr>
<td>14</td>
<td>Bad</td>
<td>R T</td>
<td>R T</td>
<td>R T</td>
</tr>
<tr>
<td>15</td>
<td>Good</td>
<td>A T</td>
<td>A T</td>
<td>A T</td>
</tr>
<tr>
<td>16</td>
<td>Good</td>
<td>A T</td>
<td>A T</td>
<td>A T</td>
</tr>
<tr>
<td>17</td>
<td>Good</td>
<td>A T</td>
<td>A T</td>
<td>A T</td>
</tr>
<tr>
<td>18</td>
<td>Good</td>
<td>A T</td>
<td>A T</td>
<td>A T</td>
</tr>
<tr>
<td>19</td>
<td>Good</td>
<td>A T</td>
<td>A T</td>
<td>A T</td>
</tr>
<tr>
<td>20</td>
<td>Bad</td>
<td>R T</td>
<td>R T</td>
<td>R T</td>
</tr>
<tr>
<td>21</td>
<td>Good</td>
<td>A T</td>
<td>A T</td>
<td>A T</td>
</tr>
<tr>
<td>22</td>
<td>Bad</td>
<td>R T</td>
<td>R T</td>
<td>R T</td>
</tr>
<tr>
<td>23</td>
<td>Good</td>
<td>A T</td>
<td>A T</td>
<td>A T</td>
</tr>
<tr>
<td>24</td>
<td>Good</td>
<td>A T</td>
<td>A T</td>
<td>A T</td>
</tr>
<tr>
<td>25</td>
<td>Good</td>
<td>A T</td>
<td>A T</td>
<td>A T</td>
</tr>
<tr>
<td>26</td>
<td>Good</td>
<td>A T</td>
<td>A T</td>
<td>A T</td>
</tr>
<tr>
<td>27</td>
<td>Good</td>
<td>A T</td>
<td>A T</td>
<td>A T</td>
</tr>
<tr>
<td>28</td>
<td>Good</td>
<td>A T</td>
<td>A T</td>
<td>A T</td>
</tr>
<tr>
<td>29</td>
<td>Bad</td>
<td>R T</td>
<td>R T</td>
<td>R T</td>
</tr>
<tr>
<td>30</td>
<td>Bad</td>
<td>R T</td>
<td>A F</td>
<td>A F</td>
</tr>
<tr>
<td>31</td>
<td>Bad</td>
<td>R T</td>
<td>R T</td>
<td>R T</td>
</tr>
</tbody>
</table>
Figure 5.1 Package Positions in System Testing
This accuracy is not satisfactory. But all classification errors occur with only two bad packages that were accepted by the inspection system. Since there is a much greater proportion of good packages in a real system than in our sample population, the accuracy in real system will be much better than in this experiment.

The classification errors could have been predicted by examination of the packages. Package 8 has a very narrow foil line, and package 30 has many wrinkles on the foil line. One way to correct these errors is to reduce the threshold value t. However, the risk of rejecting good packages will rise. An ideal system should minimize both types of errors, i.e., rejecting good packages and accepting bad packages.

5.2 Future Expansion

With two alternative package conditions (good and bad) and two inspection diagnostic alternatives (accept and reject), data are arranged in a two-by-two contingency table (see Table 5.2)(18). The "b" in this table corresponds to the first type of error – accepting bad packages, and the "c" corresponds to the second type of error – rejecting good packages. Minimizing both types of errors may be realized by optimizing threshold value. Increasing threshold value will
Table 5.2 Two-by-two Contingency Table

<table>
<thead>
<tr>
<th>Inspection</th>
<th>Package</th>
<th>Good</th>
<th>Bad</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept</td>
<td>a</td>
<td>b</td>
<td>a+b</td>
</tr>
<tr>
<td>True-accept</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>False-accept</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reject</td>
<td>c</td>
<td>d</td>
<td>c+d</td>
</tr>
<tr>
<td>False-reject</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>True-reject</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a+c</td>
<td>b+d</td>
<td>N=a+b+c+d</td>
<td></td>
</tr>
</tbody>
</table>

Where:
- \((a+b)/N\) = Probability of acceptance
- \((c+d)/N\) = Probability of rejection
- \((a+c)/N\) = Probability of good packages
- \((b+d)/N\) = Probability of bad packages

increase \(a+b\) and decrease \(c+d\). Hence this change will increase "b" errors and decrease "c" errors for a given ratio of \(a+c\) to \(b+d\).

**Strategy 1**: regulation of the threshold value depends on the prior probability of good and bad packages. If the good package has a high prior probability \((a+c)/N\) is large), threshold value should be increased. Then the quantity from Table 5.2 that reflects acceptance, \((a+b)/N\), will be raised with little effect on the number of errors.

**Strategy 2**, the regulation of the threshold value also depends on the benefits ascribed to correct outcomes and the costs ascribed to incorrect outcomes. If the cost of rejecting a good package is small relative to the cost of failing to reject a bad package, threshold value should be lowered.

In a real system both strategies may be used at the same
A better way of regulating threshold value is adaptive control. Since environmental conditions change with time, even though we find the optimal threshold value in static state, it may not be best over all time. During the processing, as long as a bad package appears, adjusting the threshold value according to the bottom value of the "valley" in the histogram as well as the strategies that are discussed above will result in dynamic threshold value optimization.

5.3 Conclusion

This work combined many different techniques including image processing, computer interfacing, electrical and mechanical technology, and then applied them in manufacturing. The most important section is machine vision. This work shows: 1) How machine vision can be used in quality control; 2) Modern manufacturing development needs to synthesize knowledge of many fields.

Before this work is applied in industry, it needs to be modified further. The most obvious shortcoming is the long process cycle of up to 31 seconds. The main reason is that ImageWise digitizer transfers image data to the computer over a serial interface. The part of the process from capturing a picture to saving it on hard disc takes 23 seconds. So, parallel data transmission interface is essential in a real production assembly line.
Bibliography


Appendices
Appendix A  Program INSPECT and VISION

#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <process.h>

main()
{
    /* FUNCTION DECLARATIONS */
    char initialization(int *threshLevel, float *elongLevel);
    void conveyor(int door);
    void histogram();
    float elong(char level);

    /* VARIABLE DECLARATIONS */
    int threshLevel;   /* Threshholding Level */
    int door;          /* Flag for Gate Position */
    char flag;         /* Flag for Program Control */
    float elongLevel;  /* Elongation Level */
    float elongation;  /* Elongation Value */

    /* PROGRAM CODE */
    flag = initialization(&threshLevel, &elongLevel);   /* Initialization */
    door = 1;                                           /* Set the Gate Open */

    while ((flag != 'q') && (flag != 'Q'))
    {
        conveyor(door);                                  /* Run the Conveyor */
conveyor(door);  /* Run the Conveyor */
spawne(P_WAIT, "vision.com", NULL);  /* Capture a Picture */
histogram();  /* Draw Histogram */

elongation = elong(threshLevel);  /* Calculate Elongation */
if (elongation < elongLevel)  /* Check Elongation */
{
    printf("\nElongation=%f. This is a good package!\n", elongation);
    door = 1;  /* Set the Gate Open */
}
else
{
    printf("\nElongation=%f. This is a bad package!\n", elongation);
    door = 0;  /* Set the Gate Close */
}

printf("\nPress any key to continue...");
printf("Press 'Q' to stop...");
flag = getche();

char initialization(int *threshLevel, float *elongLevel)
{
    /* DEFINE CONSTANTS */
    unsigned int turnoff=0xF6;  /* Value for Turning Off the Conveyor */
    /* SET INPUT AND OUTPUT PORTS' ADDRESSES */
    unsigned int pio=0x280;  /* Parallel IO Base Address */
    unsigned int porta=pio;  /* Port A Address */
    unsigned int portb=pio+1;  /* Port B Address */
    unsigned int portc=pio+2;  /* Port C Address */
    unsigned int ctrlpio=pio+3;  /* Control Port Address */

    /* PIO SETUP CONSTANTS */
}
unsigned int portain=0x10, portbin=0x2;
unsigned int portcin=0x9, modeset=0x80;
unsigned int portaout=0, portbout=0, portcout=0;

/* DECLARE VARIABLES */
char flag;w /* Flag for Program Control */

/* PROGRAM CODE */
/* SET CONTROL PORT */
outp(ctrlpio, portain portcout portbout modeset);
/* RESET MOTOR */
outp(portb, turnoff); /* Turn Off the Motor */
/* RESET ENCODER */
outp(portc, 3); /* set LOAD high and RESET high */
outp(portc, 2); /* set LOAD high and RESET low */
outp(portc, 3); /* set LOAD high and RESET high */
printf("Start to run the system? (y/n) \n");
flag = getche();
if (flag == 'y' flag == 'Y')
{
    printf("\nPlease enter thresholding level (0-63): ");
    scanf("%d", &(*threshLevel));
    printf("\nPlease enter elongation level (0.0-1.0): ");
    scanf("%f", &(*elongLevel));
}
else
    flag = 'q'; /* Quitting the Program */
return(flag);

/**********************************************************/
/* Function Conveyor always turns the conveyor motor on */
/* first, moves inspected package out of the inspection */
/* area and sends a new package into the inspection */
/* area. At meantime it sets the diversion gate open or */
/* closed depending on an input value from main program. */
/* During the motor running, it reads encoder counter */
/* inputs continuously. As soon as the number of wave */
/* cycles reaches, it turns the motor off. */
/**********************************************************/

void conveyor(door)
int door;
{
    /* DEFINE CONSTANTS */
    unsigned int turnoff=0xF6; /* Value for Turning Off the Conveyor */
    /* SET INPUT AND OUTPUT PORTS' ADDRESSES */
    unsigned int pio=0x280; /* Parallel IO Base Address */
    unsigned int porta=pio; /* Port A Address */
    unsigned int portb=pio+1; /* Port B Address */
    unsigned int portc=pio+2; /* Port C Address */
/* PIO SETUP CONSTANTS */
unsigned int portain=0x10, portbin=0x2;
unsigned int portcin=0x9, modeset=0x80;
unsigned int portaout=0, portbout=0, portcout=0;

/* DECLARE VARIABLES */
unsigned int outval;     /* Value for Controlling
                         the Motor, and the Gate */
unsigned int encodata;   /* Value Read from the Encoder */
unsigned int count;      /* Encoder Rotation Count */
unsigned int preval;     /* Last Value of encodata */
char valve;              /* Flag for Running or
                         Stopping the Motor */

/* PROGRAM CODE */
/* SET THE GATE POSITION */
if (door == 1)
   outval = 0xF2;     /* Turn on the Conveyor Motor */
else
   outval = 0x02;     /* Turn on the Conveyor Motor */

/* INITIAL VALUES */
count=0;
preval=0;
valve='c';
outp(portb, outval);

/* READ ENCODER DATA, AND CONTROL THE CONVEYOR */
do
{
   outp(portc, 1);
   encodata=inp(porta);
   outp(portc, 3);
   printf("Encoder Data = %d\n", encodata);
   if (encodata < preval)
      count=count+1;
   preval=encodata;
   if (count==2)
   {
      outp(portb, turnoff);
      valve=' ';  /* RESET THE ENCODER */
      outp(portc, 2);
      outp(portc, 3);
      outp(portc, 2);
   }
}
while (valve=='c');
void histogram()
{

    /* DEFINE CONSTANTS */
    int barchar=0xDB;    /* Display Char for Bar */
    int halfbar=0xDC;    /* Half Length Bar */
    int maxbar=20;       /* Length of Longest Bar */

    /* DECLARE FILE POINTER */
    FILE *fp;

    /* DECLARE VARIABLES */
    float maxval;       /* The Largest Bin Value */
    float maxvall;      /* The Next Largest Bin */
    float barstep;      /* Height of Steps */
    float halfstep;     /* Half of Barstep */
    float histo[64];    /* Histogram Value */
    float barmid;       /* Bottom of Bar */
    float barbase;      /* Middle of Bar */
    char ch;
    int i, j, flag;

    /* PROGRAM CODE */
    for (i = 0; i <= 63; i++)
        histo[i] = 0.0;       /* Initial Array */
    fp=fopen("package.dat", "rb"); /* Open Image Data File */
    while (!feof(fp))
    {
        ch=fgetc(fp);        /* Read Image Data */
        histo[ch] += 1.0;    /* Build Histogram */
    }
    fclose(fp);            /* Close Image Data File */

    /* FIND LARGEST VALUE */
    maxval = 1.0;
    maxvall = maxval;
    for (i = 0; i <= 63; i++)
    {
        if (histo[i] > maxval)
        {
            maxvall = maxval;    /* Save Previous High */
            maxval = histo[i];   /* Set New High */
        }
        else if (histo[i] > maxvall)
            maxvall = histo[i]; /* Set 2nd Highest so far */
    }
```c
/* Calculate Steps between Lines */
barstep = maxvall / maxbar;
halfstep = barstep / 2.0;

;/* Show Histogram on the Screen */
for (j = maxbar; j >= 1; j--)
{
  barbase = (float)(int)(barstep * j);
barmid = barbase + halfstep;
printf("%6.0f", barbase);
for (i = 0; i <= 63; i++) /* Show in Each Line */
{
  if (histo[i] > barmid) printf("%c", barchar);
  else if (histo[i] > barbase) printf("%c", halfbar);
  else printf(_");
}
printf("\n");
}
printf("0");
for (i = 0; i <= 63; i++) /* Show in Bottom */
{
  if (histo[i] > halfstep) printf("%c", barchar);
  else if (histo[i] > 0.0) printf("%c", halfbar);
  else printf(_");
}
printf("\n");

;/* Show coordinate */
printf("0");
for (i = 1; i <= 6; i++)
  printf("%d", i);
printf("\n");
printf(" ");
for (i = 0; i <= 5; i++)
  for (j = 0; j <= 9; j++)
    printf("%d", j);
printf("0123\n");
}

/**********************************************************/
/* Function Elong has two tasks. One is thresholding; */
/* the other is calculating image’s elongation. It */
/* reads image pixels’s value from a data file */
/* "PACKAGE.DAT", then processes thresholding. */
/* Thresholding level is an input variable translated */
/* from main program. After thresholding, it calculates */
/* image’s elongation, then return elongation value to */
/* main program. */
/***********************************************************/
float elong(level)
char level;
{
    /* DECLARE FILE POINTER */
    FILE *fp;

    /* DECLARE VARIABLES */
    char pixel[81][256];    /* Array of Pixels' Gray Scale */
    unsigned int area_x[256];     /* Area Histograms in x Direction */
    unsigned int area_y[81];        /* Area Histograms in y Direction */
    unsigned int moment_x[256];    /* Moment Histograms in x Direction */

    unsigned int i, j;
    unsigned long int n;       /* Area: Number of Ones */
    float inertia_a;           /* Moments of Inertia */
    float inertia_b;           /* Product of Inertia */
    float inertia_c;           /* Moments of Inertia */
    float elongation;          /* Elongation */
    float coordinate_x;        /* Coordinate in x Direction */
    float coordinate_y;        /* Coordinate in y Direction */
    char ch;

    /* INITIAL VARIABLES */
    n = 0;
    inertia_a = inertia_b = inertia_c = 0.0;
    elongation = coordinate_x = coordinate_y = 0.0;

    /* READ PIXELS' VALUE FROM A DATA FILE */
    fp=fopen("package.dat", "rb");
    for (i = 0; i <= 80; i++)
        for (j = 0; j <= 255; j++)
            { 
                ch = fgetc(fp);
                if (ch < level)
                    pixel[i][j] = 0;
                else
                    pixel[i][j] = 1;
            }
    fclose(fp);

    /* CALCULATE area_x[j] AND moment_x[j] */
    for (j = 0; j <= 255; j++)
        { 
            area_x[j] = moment_x[j] = 0;
            for (i = 0; i <= 80; i++)
                { 
                    area_x[j] += pixel[i][j];
                    moment_x[j] += (i + 1) * pixel[i][j];
                }
        }
/* CALCULATE area_y[i] AND n */
for (i = 0; i <= 80; i++)
{
    area_y[i] = 0;
    for (j = 0; j <= 255; j++)
        area_y[i] += pixel[i][j];
    n = n + area_y[i];
}

/* CHECK n */
if (n == 0)
{
    elongation = 0.0;
    goto output;
}

/* CALCULATE coordinate_x */
for (j = 0; j <= 255; j++)
    coordinate_x += (float)(j + 1) * area_x[j];
coordinate_x /= (float)n;

/* CALCULATE coordinate_y */
for (i = 0; i <= 80; i++)
    coordinate_y += (float)(i + 1) * area_y[i];
coordinate_y /= (float)n;

/* CALCULATE inertia_a AND inertia_b */
for (j = 0; j <= 255; j++)
{
    inertia_a += (float)(j + 1) * (j + 1) * area_x[j];
    inertia_b += (float)(j + 1) * moment_x[j];
}
inertia_a -= coordinate_x * coordinate_x * n;
inertia_b = (inertia_b - coordinate_x * coordinate_y * n) * 2.0;

/* CALCULATE inertia_c */
for (i = 0; i <= 80; i++)
    inertia_c += (float)(i + 1) * (i + 1) * area_y[i];
inertia_c -= coordinate_y * coordinate_y * n;

/* CHECK inertia_a AND inertia_c */
if (inertia_a == 0.0 && inertia_c == 0.0)
{
    elongation = 0.0;
    goto output;
}

/* CALCULATE elongation */
elongation = inertia_b * inertia_b + (inertia_a - inertia_c) * (inertia_a - inertia_c);
elongation = sqrt(elongation) / (inertia_a + inertia_c);

output: return(elongation);

}
This program is created for "Package Inspection with a Machine Vision System". It is written by Turbo Pascal. It is used to call some procedures supplied by Circuit Cellar Inc. so that the image can be caught and expended, and then saved into the file "PACKAGE.DAT".

** PROGRAM VISION; **

```pascal
_PROGRAM VISION;

{$U- control-break checking during execution }
{$C- control-break checking during I/O operations }
{$R- array range checking }

{$Ideclares.p declarations }
{$Ihexutil.p hex utilities }
{$Iserial.p serial interface code }
{$Ipictures.p picture file code }

VAR
  grey : BYTE;
  filename: FILE OF BYTE;
  lndx : INTEGER;
  pndx : INTEGER;

BEGIN
  LowVideo;
  ComOn(bitsec);
  Port[comMCR] := $03; { set up serial port }
    { PC <-> trans serial }
    { camera -> monitor }
  pic1 := NIL;
  PicSetup(pic1);
  Writeln('Loading... ');
  GetPicture(pic1,fullres); { get the byte }
  pic2 := NIL;
  PicSetup(pic2);
  Writeln('Expanding');
  Expand(pic1,pic2); { expand image }
  assign(filename, 'PACKAGE.DAT'); { link file name }
```
rewrite(filename);  { open file for output }
for lndx := 20 to 100 do  { save grey scales of }
  for pndx := 0 to 255 do  { pixcels to data file }
    begin
      grey := pic2^fmt.lines[lndx].pels[pndx];
      write(filename, grey);
    end;
    close(filename);  { close data file }
  end;
end;
Appendix B  Procedures Supplied with ImageWize Digitizer

{ DECLARES.P -- declarations for picture handlers }
{ Copyright (c) 1987, Ciarcia's Circuit Cellar }
{   All Rights Reserved }

{-----------------------------}
{ Global constants }

CONST

{--- defaults for serial port setup }
bitsec   = 28800;          { default serial speed }
COMport  = $3F8;           { 3F8 = COM1 }
                { 2F8 = COM2 }

{--- resolution control bytes to receiver & transmitter }
fullres  = $80;            { set high resolution }
halfres  = $81;            { set high resolution }
quartres = $82;            { set high resolution }

{--- control bytes from transmitter }
fieldsync = $40;           { new field! }
linesync  = $41;           { new line }
fldend    = $42;           { end of field }
repl      = $80;           { repeat x1 }
rep16     = $90;           { repeat x16 }

{--- image structure }
maxbit    = $3F;           { bits used in pel }
maxpel    = 255;           { highest pel index }
maxline   = 243;           { highest line index }
maxbuffer = 32766;         { highest "INT" index }

{-----------------------------}
{ Global types }

TYPE

bitrng    = 0..maxbit;      { bit range }
bitrng = 0..maxbit;  { bit range }  
pelrng = 0..maxpel;  { pel indexes }  
linerng = 0..maxline;  { line indexes }  
ubtrng = 0..maxbuffer;  { raw data indexes }  
pelrec = RECORD  { one scan line }  
  syncL : BYTE;  
  pels : ARRAY[pelrng] OF BYTE;  
END;  
linerec = RECORD  { complete binary field }  
  syncF : BYTE;  
  lines : ARRAY[linerng] OF pelrec;  
  syncE : BYTE;  
END;  
rawrec = ARRAY[subtrng] OF INTEGER;  
picptr = ^pictype;  { picture ptr }  
pictype = RECORD CASE INTEGER OF  { picture formats }  
  0 : (fmt : linerec);  
  1 : (words : rawrec);  
  1 : (bytes : BYTE);  { dummy }  
END;  
histtype = ARRAY[bitrng] OF REAL;  { pel histograms }  
regrec = RECORD CASE INTEGER OF  
  1 : (AX : INTEGER;  
       BX : INTEGER;  
       CX : INTEGER;  
       DX : INTEGER;  
       BP : INTEGER;  
       SI : INTEGER;  
       DI : INTEGER;  
       DS : INTEGER;  
       ES : INTEGER;  
       FLAGS : INTEGER);  
  2 : (AL,AH : BYTE;  
       BL,BH : BYTE;  
       CL,CH : BYTE;  
       DL,DH : BYTE);  
END;  
byteptr = ^BYTE;  { general ptr }  
strtype = STRING[255];  { strings }  

{-----------------------------------------------}  
{ Global variables }  

VAR
picfile : FILE OF pictype;
pic0 : picptr;
pic1 : picptr;
pic2 : picptr;
pic3 : picptr;
histo : histtype;

IOerror : BYTE;
filespec : strtype;
filespec2 : strtype;
{ HEXUTIL.P -- Hexadecimal conversion routines }

{ Copyright (c) 1987, Ciarcia's Circuit Cellar }
{ All Rights Reserved }

{---------------------------------------------------------------------}

TYPE

Hextype = STRING[4];       { input & output string }

{---------------------------------------------------------------------}

{ Converts a hex string of some hexits into the }
{ corresponding integer. }
{ Returns error code in case the numbers were bad }

PROCEDURE HexToInt(hexits : Hextype;
VAR intval : INTEGER;
VAR errcode : INTEGER);

BEGIN
  Val(''$'+hexits,intval,errcode);
END;

{---------------------------------------------------------------------}

{ convert "unsigned" 16-bit number to REAL 0..64K }

FUNCTION WordToReal(fullword:INTEGER) : REAL;

BEGIN
  IF fullword < 0 THEN WordToReal := 65536.0 + fullword
  ELSE WordToReal := fullword;
END;

{---------------------------------------------------------------------}

{ converts one 4-bit nybble to a character }

FUNCTION NybToHex(nybval:INTEGER) : Hextype;

BEGIN
  CASE nybval OF
    0..9 : NybToHex := Chr($30+nybval);
    10..15 : NybToHex := Chr($41+nybval-10);
  END;
FUNCTION IntToHex(intval:INTEGER) : Hextype;

VAR
    tempstring : Hextype;
    tempval : INTEGER;

BEGIN

    tempstring := '0000';

    {--- the first nybble depends on the sign }
    IF intval < 0 { < 0, need trick }
        THEN BEGIN
            tempval := intval XOR $FFFF; { 1's compl }
            tempval := tempval DIV 4096; { bits to LSD }
            tempval := tempval XOR $000F; { flip again }
            tempstring[1] := NybToHex(tempval); { now convert }
        END
    ELSE BEGIN
        { >= 0, simple convert }
        tempstring[1] := NybToHex(
            (intval AND $F000) DIV 4096);
    END;

    {--- the rest are easy }
    tempstring[2] := NybToHex((intval AND $0F00) DIV 256);
    tempstring[3] := NybToHex((intval AND $00F0) DIV 16);
    tempstring[4] := NybToHex( intval AND $000F);

    IntToHex := tempstring;

END;

FUNCTION ByteToHex(intval:INTEGER) : Hextype;

VAR
    tempstring : Hextype;

BEGIN
tempstring := '00';

tempstring[1] := NybToHex((intval AND $00F0) DIV 16);
tempstring[2] := NybToHex( intval AND $000F);

ByteToHex := tempstring;

END;
( SERIAL.P -- serial interface code )

{ Copyright (c) 1987, Ciarcia's Circuit Cellar }
{ All Rights Reserved }

{-----------------------------}
{ Global constants }

CONST

maxtime = 100;  \{ serial timeout \}

{-----------------------------}
{ Global variables }

VAR

COMerror : INTEGER;

{-----------------------------}
{ Async card global constants }

CONST

XON = $11;
XOFF = $13;

DataReady = $01; \{ receive data ready \}
THRE = $20;  \{ transmit data ready \}

{-----------------------------}
{ Async card global variables }

VAR

comdata : INTEGER; \{ data for async I/O \}
comien : INTEGER; \{ interrupt enable reg \}
comiir : INTEGER; \{ interrupt ID reg \}
comlcr : INTEGER; \{ line control reg \}
commcr : INTEGER; \{ modem control reg \}
comlsr : INTEGER; \{ line status reg \}
commsr : INTEGER; \{ modem status reg \}

{-----------------------------}
{ Set up the async card }
{ Rate is in bits/second }

PROCEDURE ComOn(rate : INTEGER);

CONST

serialmax = 115200.0; \{ rate -> divisors \}

VAR
dumm y  : BYTE;
counts  : INTEGER;

BEGIN

{--- set up global variables

comdata := comport;
comien  := comport + 1;
comiir  := comport + 2;
comlcr  := comport + 3;
commcr  := comport + 4;
comlsr  := comport + 5;
commsr  := comport + 6;

{--- set up port registers

counts := Trunc(serialmax/rate);

Port[comlcr] := $80;  { set DLAB to set rate  }
Port[comdata] := Lo(counts);  { set divisor LSB  }
Port[comien] := Hi(counts);  { set divisor MSB  }
Port[comlcr] := $13;    { no pty 1 stop 8 dat  }
dummy := Port[comdata];   { discard pending char  }

END;

{ Send a byte to the serial port
{ If there's an XOFF in the receiver, we wait...
{ This is not likely, but it has been known to happen

PROCEDURE SendByte(databyte : BYTE);

BEGIN

COMerror := 0;    { can't have error...  }

WHILE (Port[comlsr] AND THRE) = 0 DO;   { send done?  }

WHILE Port[comdata] = XOFF DO;  { XOFF pending?  }

Port[comdata] := databyte;   { send data

END;
{ PICTURES.P -- picture file routines }

{ Copyright (c) 1987, Ciarcia’s Circuit Cellar }
{ All Rights Reserved }

{ Dump a segmented address with a message }

PROCEDURE DumpAddr(msg : strtype;
  segment,offset : INTEGER);
BEGIN
  Writeln(msg,IntToHex(segment),':',
    IntToHex(offset));
END;

{ Get file spec if not present }

FUNCTION GetFSpec(fn: strtype) : strtype;
BEGIN
  IF Length(fn) = 0
      THEN BEGIN
          Write('Picture file name: ');
          Readln(fn);
          GetFSpec := fn;
      END;
  ELSE GetFSpec := fn;
END;

{ Present message, return boolean response }

FUNCTION Askit(msg : strtype) : BOOLEAN;
VAR
  resp : STRING[5];
BEGIN
  Write(msg,' ');
  Readln(resp);
  Askit := FALSE;
  IF Length(resp) <> 0
      THEN IF UpCase(resp[1]) = 'Y'
        THEN Askit := TRUE;
END;
PROCEDURE PicSetup(VAR newpic : picptr);

VAR
pelrs : pelerng;
lines  : linerng;

BEGIN
IF newpic <> NIL THEN Dispose(newpic);
New(newpic);
END;

PROCEDURE GetPicture(pic : picptr;
resol : BYTE);

VAR
picbyte : BYTE;
bptra : byteptr;

BEGIN
Port[comMCR] := $03;
Delay(200);

(* Write('Waiting for key press...');
Readln; *)

bptra := Ptr(Seg(pic^),Ofs(pic^)-1);

(* Writeln('KeyPressed is: ',KeyPressed);
Writeln('port end is: ',(Port[comdata]=fend)); *)
SendByte(resol); { specify resolution }
SendByte(XON); { prompt transmitter }

REPEAT { for each line }
bptr := Ptr(Seg(bptr^),Ofs(bptr^)+1); { tick ptr }
WHILE ((Port[comLSR] AND DataReady) = 0) AND
    NOT KeyPressed DO; { stall waiting }
bptr^ := Port[comdata]; { snag the byte }
UNTIL (bptr^ = fldend) OR KeyPressed;

(*
    Writeln('KeyPressed is: ',KeyPressed);
    Writeln('port end is: ',(Port[comdata]=fldend));
    Writeln('data end is: ',(bptr^=fldend));
*)

Port[comMCR] := $00; { PC <-> rec serial }
    { rec -> monitor }

END;

{------------------------------------------------------------------------}
{ Save picture file on disk }
{ Uses the smallest number of blocks to fit the data }

PROCEDURE SavePicture(filespec : strtype;
           pic : picptr);

VAR
    ndx     : subrng; { index into word array }
    rndx    : REAL;  { real equivalent }
    nbblocks : INTEGER; { number of disk blocks }
    xfered  : INTEGER; { number actually done }
    pfile   : FILE;   { untyped file for I/O }

BEGIN

Writeln('Writing ',filespec);
Assign(pfile,filespec);
Rewrite(pfile);

ndx := 0; { start with first word }

Write(' Data length = '); { start with first word }

WHILE (ndx < maxbuffer) AND { WHILE not end of pic }
    (Lo(pic^.words[ndx]) <> fldend) AND
    (Hi(pic^.words[ndx]) <> fldend) DO
    ndx := ndx + 1;

ndx := ndx + 1; { fix 0 origin }
rndx := 2.0 * ndx;  \{ allow >32K numbers... \}
Write(rndx:6:0,' bytes, file length = ');  

nblocs := ndx DIV 64;  \{ 64 words = 128 bytes \}
IF (ndx MOD 64) <> 0  \{ partial block? \}
  THEN nblocs := nblocs + 1;

rndx := 128.0 * nblocs;  \{ actual file size \}
Writeln(rndx:6:0,' bytes');

BlockWrite(pfile,pic^.words[0],nblocs,xfered);

IF xfered <> nblocs  \{ completed? \}
  THEN BEGIN
    Writeln('Problem writing the file, error code: ', I0error);
    Writeln(' Blocks computed: ',nblocs);
    Writeln(' Blocks written: ',xfered);
  END;

END;

{-------------------------------------------------------------------}
{ Load picture file from disk }

PROCEDURE LoadPicture(filespec : strtype;
        pic : picptr);
BEGIN

  Writeln('Reading ',filespec);
  Assign(picfile,filespec);

  {$I- turn off I/O checking}
  Reset(picfile);
  I0error := IOresult;
  {$I+ turn on I/O checking again}

  IF IOresult <> 0
    THEN BEGIN
      Writeln('Problem reading the file, IO error: ', I0error);
      HALT;
    END;

  {$I- turn off I/O checking}
  Read(picfile,pic^);  \{ this does the read \}
  I0error := IOresult;
  {$I+ turn on I/O checking again}
IF NOT (IOresult IN [0,$99])  { $99 = short block, OK }
THEN BEGIN
   Writeln('Problem reading the file, IO error: ', IOerror);
   HALT;
END;
END;

{------------------------------------------------------------------------}
{ Send picture to display  }
{ Sets RTS and DTR to switch the relay box to ensure  }
{ a good connection  }
PROCEDURE SendPicture(pic : picptr);

VAR
   bptr       : byteptr;  { fake pointer to pic  }
BEGIN
   Port[comMCR] := $00;  { PC <-> rec serial  }
      { rec -> monitor  }
   Delay(100);  { pause to stabilize  }
   bptr := Ptr(Seg(pic^),Ofs(pic^)-1);  { set byte ptr  }
   REPEAT  { for each line  }
      bptr := Ptr(Seg(bptr^),Ofs(bptr^)+1);  { tick ptr  }
      WHILE (Port[comdata] = XOFF) AND NOT KeyPressed DO;
      WHILE ((Port[comLSR] AND THRE) = 0) AND
         NOT KeyPressed DO;  { stall for data  }
      Port[comdata] := bptr^;  { send the byte  }
   UNTIL (bptr^ = fldend) OR KeyPressed;
END;

{------------------------------------------------------------------------}
{ Set up frame and line syncs in a buffer  }
{ This should be done only in freshly allocated buffers  }
PROCEDURE SetSyncs(pic1 : picptr);

VAR
   lndx       : linerng;  { index into lines  }
BEGIN
   pic1^.fmt.syncF := fieldsync;  { set up empty picture  }
FOR lndx := 0 TO maxline DO BEGIN
  pic1^ . fmt . lines[ lndx ] . syncL := linesync;
  FillChar ( pic1^ . fmt . lines[ lndx ] . pels[ 0 ] , maxpel + 1 , 0 ) ;
END;

  pic1^ . fmt . syncE := fldend;       { set ending control   }

END;

{---------------------------------------------------------------}
{ Decompress pic1 into pic2  }

PROCEDURE Expand ( pic1 , pic2 : picptr ) ;

CONST
  errthresh = 10 ;                      { max errors in frame   }

VAR
  bptr     :    ^byte ;
  lndx     :    linerng ;
  pndx     :    pelrng ;
  overflow :    BOOLEAN ;
  oldbyte  :    BYTE ;
  reps     :    INTEGER ;
  frametop :    BOOLEAN ;
  giveup   :    BOOLEAN ;
  errcount :    INTEGER ;

BEGIN

  bptr := Ptr ( Seg ( pic1^ ) , Ofs ( pic1^ ) ) ;

  SetSyncs ( pic2 ) ;                    { fill in the syncs    }

  lndx := 0 ;
  pndx := 0 ;

  frametop := TRUE ;
  giveup := FALSE ;
  errcount := 0 ;

  WHILE ( bptr^ <> fldend ) AND NOT giveup
  DO BEGIN { and now the data...       }
    CASE bptr^ OF
      fieldsync : BEGIN
        IF ( lndx <> 0 ) OR ( pndx <> 0 )
          THEN BEGIN
            Writeln ( 'Field sync found after data' ) ;
            END;
        oldbyte := 0 ;
        frametop := TRUE ;

        (*
          Writeln ( 'Field sync' ) ; *)

      END;

  END ;

  END ;

END ;
END;
linesync : BEGIN
  IF (lndx < maxline) AND NOT frametop
    THEN lndx := lndx + 1
  ELSE frametop := false;
oldbyte := 0;
pndx := 0;
overflow := FALSE;
(*
  Write(‘.’); *)
END;
fldend : BEGIN
  { can’t get here... }
  Writeln;
  Writeln(‘Surprise at having found field end!’);
END;
ELSE BEGIN
  CASE (bptr\^ AND $F0) OF
    $00..$3F : BEGIN
      pic2\^ .fmt .lines[lndx] .pels[pndx] := bptr\^;
      oldbyte := bptr\^;
      IF pndx < maxpel
        THEN BEGIN
          pndx := pndx + 1;
          IF overflow
            THEN BEGIN
              Write(‘Too much data on line ’,lndx:3);
              Writeln(‘ pel data ’,ByteToHex(bptr\^));
              errcount := Succ(errcount);
            END;
          END
        ELSE BEGIN
          pndx := 0;
          overflow := TRUE;
        END;
    END;
    repl : BEGIN
      FOR reps := 1 TO (bptr\^ AND $0F) DO BEGIN
        pic2\^ .fmt .lines[lndx] .pels[pndx] := oldbyte;
        IF pndx < maxpel
          THEN BEGIN
            pndx := pndx + 1;
            IF overflow
              THEN BEGIN
                Write(‘Too much data on line ’,lndx:3);
                Writeln(‘ 1x rep ’,ByteToHex(bptr\^));
                errcount := Succ(errcount);
                pndx := 0;
              END;
            END
          END;
      END
      ELSE BEGIN
        pndx := 0;
        overflow := TRUE;
    END
  END
END
END;
(*
   Writeln('Repl: ',ByteToHex(bptr^)); *)
END;
END;

rep16 : BEGIN
   FOR reps := 1 TO (16 * (bptr^ AND $0F)) DO BEGIN
      pic2^.fmt.lines[lndx].pels[pndx] := oldbyte;
      IF pndx < maxpel
         THEN BEGIN
            pndx := pndx + 1;
            IF overflow
               THEN BEGIN
                  Write('Too much data on line ',lndx:3);
                  Writeln('16x rep ',ByteToHex(bptr^));
                  errcount := Succ(errcount);
                  pndx := 0;
               END
         ELSE BEGIN
            pndx := 0;
            overflow := TRUE;
         END;
      END;
      ELSE BEGIN
         Writeln('Garbage byte: ',ByteToHex(bptr^),
                   ' at line ',lndx,' pel ',pndx);
         errcount := Succ(errcount);
      END;
   END;
   IF errcount > errthresh
      THEN giveup := TRUE;
   bptr := Ptr(Seg(bptr^),Ofs(bptr^)+1);  { next input byte }
END;

IF giveup
   THEN BEGIN
      Writeln('Too many errors -- giving up!');
      HALT;
   END;
   (* Writeln; *)
END;
PROCEDURE DoCount(reps : INTEGER;
VAR bptr : byteptr);
BEGIN

IF reps >= 256 THEN BEGIN
  bptr^ := repl6;
  { default = 256 }
  bptr := Ptr(Seg(bptr^),Ofs(bptr^)+1);
  reps := reps - 256;
  { fix the remainder }
END;

IF (reps AND $F0) <> 0 THEN BEGIN
  bptr^ := repl6 + (reps SHR 4);
  bptr := Ptr(Seg(bptr^),Ofs(bptr^)+1);
  reps := reps AND $0F;
END;

IF reps <> 0 THEN BEGIN
  bptr^ := repl + reps;
  bptr := Ptr(Seg(bptr^),Ofs(bptr^)+1);
END;

END;

PROCEDURE Compress(pic1,pic2 : picptr);
VAR
  bptr : ^byte;
  lndx : linerng;
  pndx : pelrng;
  oldbyte : BYTE;
  reps : INTEGER;
BEGIN

{--- fill buffer with zeros to ensure no trash }
FillChar(pic2^.words[0],maxbuffer,0);
FillChar(pic2^.words[maxbuffer DIV 2],maxbuffer,0);
bptr := Ptr(Seg(pic2^),Ofs(pic2^));

bptr^ := fieldsync;          { flag the start     }
bptr := Ptr(Seg(bptr^),Ofs(bptr^)+1);

FOR lndx := 0 TO maxline DO BEGIN
  bptr^ := linesync;          { flag new line    }
bptr := Ptr(Seg(bptr^),Ofs(bptr^)+1);
  oldbyte := 0;
  reps := 0;                  { force leading zero   }

  FOR pndx := 0 TO maxpel DO BEGIN
    IF pic1^.fmt.lines[lndx].pels[pndx] = oldbyte
      THEN reps := reps + 1       { accumulate count    }
    ELSE BEGIN
      IF reps > 1
        THEN DoCount(reps,bptr);  { n reps, send count    }
      IF reps = 1
        THEN BEGIN
          bptr^ := oldbyte;
          bptr := Ptr(Seg(bptr^),Ofs(bptr^)+1); { step ptr    }
        END;
      reps := 0;                  { reset counter       }
      oldbyte := pic1^.fmt.lines[lndx].pels[pndx];
    END;
  END;

  (--- send last count and trailing zero)

  IF reps > 1
    THEN DoCount(reps,bptr);  { n reps, send count    }
  IF reps = 1
    THEN BEGIN
      bptr^ := oldbyte;
      bptr := Ptr(Seg(bptr^),Ofs(bptr^)+1); { step ptr    }
    END;

  bptr^ := 0;                  { force trailer zero   }

  (*Write('.'); *)
END;

bptr^ := fldend;              { flag the ending    }
Writeln;

END;
Appendix C  Shaft Encoder Counter Circuit
Protocol

Startup: 1) set LOAD high
2) run device to zero position
3) set RESET low then high

Read: 1) set LOAD low
2) read input port
3) set LOAD high

Notes

1) low is ground potential; high is +V
2) +V is CMOS positive supply range
3) LOAD latches data to read on falling edge
4) RESET sets counter to zero when pulsed low

Shaft Encoder Counter Circuit

Eugene Fichter
28 Sept 1989
### Appendix D Equipment List

<table>
<thead>
<tr>
<th>Name</th>
<th>Qty.</th>
<th>Company</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Personal Computer</td>
<td>1</td>
<td>AST Research, Inc.</td>
<td>Premium/286</td>
</tr>
<tr>
<td>Digitizer</td>
<td>1</td>
<td>Circuit Cellar Inc.</td>
<td></td>
</tr>
<tr>
<td>Camera</td>
<td>1</td>
<td>RCA Closed-Circuit Video Equipment</td>
<td>TC150116</td>
</tr>
<tr>
<td>Monitor</td>
<td>1</td>
<td>RCA Closed-Circuit Video Equipment</td>
<td>TC1910A 9&quot;; B-W</td>
</tr>
<tr>
<td>lens</td>
<td>1</td>
<td>KODAK Corp.</td>
<td>f:2.7; 15mm</td>
</tr>
<tr>
<td>Parallel Digital I/O Interface Board</td>
<td>1</td>
<td>MetraByte Corp.</td>
<td>P1012</td>
</tr>
<tr>
<td>Screw Terminal Board</td>
<td>1</td>
<td>MetraByte Corp.</td>
<td></td>
</tr>
<tr>
<td>Optically Isolated I/O Mounting Rack</td>
<td>f</td>
<td>OPTO-22</td>
<td></td>
</tr>
<tr>
<td>Conveyor</td>
<td>1</td>
<td>Dorner Mfg. Corp.</td>
<td>4100 Series</td>
</tr>
<tr>
<td>Shaft Encoder</td>
<td>1</td>
<td>Datametrics Inc.</td>
<td>LDM-50-5LD-1</td>
</tr>
<tr>
<td>Relay</td>
<td>2</td>
<td>AMF Potter &amp; Brumfield</td>
<td>KRP14DG; 24VDC</td>
</tr>
<tr>
<td>Rotary Solenoid</td>
<td>1</td>
<td>Ledex Inc.</td>
<td>H-1082-032</td>
</tr>
<tr>
<td>Power Supplier 1</td>
<td>1</td>
<td>Intel Corporation</td>
<td>CBS 635 (+5, -5, +12, -12V)</td>
</tr>
<tr>
<td>Power Supplier 2</td>
<td>1</td>
<td>Xentek Inc.</td>
<td>XE180-24 W/OVP</td>
</tr>
<tr>
<td>Power Supplier 3</td>
<td>1</td>
<td>Amcodyne</td>
<td>+5V</td>
</tr>
<tr>
<td>Counter Circuit</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>