

DESIGN OF A PC CONTROLLED VISION INSPECTION SYSTEM

A thesis written at

DMC

and submitted to

KETTERING UNIVERSITY

in partial fulfillment
of the requirements for the
degree of

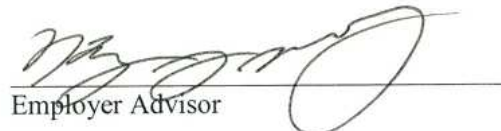
BACHELOR OF SCIENCE IN ELECTRICAL ENGINEERING

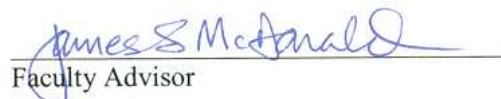
by

MATTHEW M. PUSKALA

March 2002


Author


Employer Advisor


Faculty Advisor

DISCLAIMER

This thesis is being submitted as partial and final fulfillment of the cooperative work experience requirements of Kettering University needed to obtain a Bachelor of Science in Electrical Engineering Degree.

The conclusions and opinions expressed in this thesis are those of the writer and do not necessarily represent the position of Kettering University or my employer, or any of its directors, officers, agents, or employees with respect to the matters discussed.

PREFACE

This thesis represents the capstone of my five years combined academic work at Kettering University and my job experience at DMC and other sponsors. Academic experience in programming, electronics, and design combined with work assignments in control systems, software design, and electronics, proved to be valuable assets while I developed my thesis and addressed the problem it concerned.

Although this thesis represents the compilation of my own efforts, I would like to acknowledge and extend my sincere gratitude to the following persons for their valuable time and assistance, without whom the completion of this thesis would not have been possible:

1. Dr. James McDonald, Associate Professor of Computer Engineering at Kettering University, for his support and guidance in the organization and development of this thesis, while acting as my faculty advisor
2. Ken Brey, Technical Advisor at DMC, for all his help and guidance with regard to the technical aspects of this thesis, while acting as my employee advisor.

TABLE OF CONTENTS

DISCLAIMER	ii
PREFACE.....	iii
LIST OF ILLUSTRATIONS	vi
I. INTRODUCTION	1
Problem Topic.....	1
Background.....	1
Criteria and Parameter Restrictions	2
Methodology.....	2
Primary Purpose.....	3
Overview.....	3
II. HARDWARE	4
Personal Computer.....	5
Camera and Lighting.....	5
Triggering Hardware.....	5
III. USER INTERFACE	8
Setup screen	8
Main Screen	9
Parameters screen.....	11
Access Database.....	12
IV. PROGRAM IMPLEMENTATION	15
Key Controls	15
DVTSID(0) and DVTSID(1).....	15
DataLinkWinsock(0).....	16
Winsock(0)	16
Infile1	16
Program Startup and Initialization	17
Database Operations	19
Inspection Data Reception and Handling	23
V. CAMERA SETUP AND PROGRAMMING	26
Camera, Lens, Light, and Fixture Setup	26
DVT Program.....	29

Soft sensors.....	30
IO setup	35
VI. CONCLUSIONS AND RECOMMENDATIONS	40
Verification Methods and Results.....	41
Future Recommendations	41
APPENDICES	42
APPENDIX A: CODE SAMPLES.....	43
APPENDIX B: FLOW CHARTS.....	47
APPENDIX C: SYSTEM WIRING DIAGRAM	51

LIST OF ILLUSTRATIONS

<u>Figure</u>	<u>Page</u>
1. Basic system block diagram.....	4
2. Sample of material	6
3. Inspection Program Setup screen.....	9
4. Inspection Program Main Screen with a reel open	10
5. Inspection Program Main Screen with all reels closed.....	10
6. Inspection Program Parameters screen	12
7. Access database tables	13
8. Reel table.	14
9. Failure table.	14
10. Main Screen Form_Load subroutine	18
11. DAO variable declarations.....	20
12. Excerpt from Parameters screen Form_Load subroutine.....	21
13. Excerpt from Parameters screen cmdDone_Click subroutine	22
14. CreateReelRecord subroutine	23
15. DataLinkWinsock_DataArrival subroutine	23
16. HandleNewInspection subroutine.....	25
17. Lighting and fixture	27
18. DVT soft sensors (initial chain links).....	31

19.	DVT soft sensors (final chain links)	33
20.	Framework I/O Parameters: Timing settings	36
21.	Framework I/O Parameters: Outputs settings	37
22.	Framework I/O Parameters: User settings	37
23.	Data Link Parameters: Sensors screen	38
24.	Data Link Parameters: Strings screen	39

Appendices

A-1.	MainMenu Initialize subroutine.....	44
A-2.	CreateReelTable Subroutine	45
A-3.	OpenDatabase subroutine	46
B-1.	Flowchart for MainMenu Form_Load subroutine	48
B-2.	Flowchart for MainMenu Initialize subroutine.....	49
B-3.	Data handling subroutine	50
C-1.	System Wiring Diagram	52

I. INTRODUCTION

This paper documents a solution provided by DMC to one of its customers in the form of a vision inspection and data acquisition system. DMC is an engineering consulting company that provides custom software solutions for its customers. For this project the customer is a company that operates chemical plating processes for metal products. In this solution, a “smart” digital camera is set up and programmed to take images and pass inspection results and measurements to a custom program. This program, the Inspection Program, records data and runs on a personal computer (PC).

Problem Topic

DMC has agreed to assist the customer with problems related to quality control of a specific product. This product, in reel form, has occasional bent parts, or pins, at the end of a metal plating process. DMC’s task is to design and implement a vision and data acquisition system to identify bent pins, providing consulting services by specifying hardware and wiring, and programming the system.

Background

DMC provides many different types of software and control systems solutions for its customers. Vision inspection and data acquisition systems are common solutions which customers ask DMC to provide.

In this system, the product is a series of metal pins in the shape of two pronged forks on a reel. Each fork will later be folded over to create a female pin connection used

on D-Sub cable connectors. This customer wishes to find a solution to this problem because their end customer has had quality issues with the end product they received. Pins that are bent before or during the metal plating process interfere with the end customer's next portion of the manufacturing process.

Criteria and Parameter Restrictions

The primary constraints for this system are budgetary and time related. The customer pays for all hardware used and all DMC time spent working on the project. The system is to be capable of inspecting parts at a rate of 900 parts per minute, because this is the maximum rate at which the line will run. The system must be capable of detecting variations in the part's tip location within +/- .003 Inches. The system will display live images and continuously updated statistics to the operator during the plating process. This allows the operator to quickly assess problems in the metal plating process and potential malfunctions in the vision inspection system. Detailed data on failed parts must be stored for later access by the customer. Ultimately, the customer can use this statistical data for process improvement and maintenance of the metal plating system. These constraints and criteria are developed and agreed upon by the customer and DMC during the proposal/estimate stage and the specifications stage of the project.

Methodology

The methodology used to develop a successful system relied primarily upon DMC's and the author's past experience and knowledge gained from similar successful projects in vision inspection and data acquisition. The first step was to develop a system proposal and estimate, which was agreed upon by DMC and the customer, including basic system requirements and costs. The next step was to develop system specifications,

which also was to be agreed upon by DMC and the customer. The specifications included details on hardware and software to be used in system development, performance specifications, and general information about all screens contained within the Inspection Program. Starting from the basic system guidelines from the proposal and specifications, the entire system was developed and debugged using a method of trial and error. After the general system was running, it was tested and further fine-tuned by running actual reels of the part through. Worst case scenario testing was used to check the system's error handling and recovery.

Primary Purpose

The purpose of this document is to present the end solution provided by DMC for the customer.

Overview

Both general use and theory of operation for the system are discussed within this document. The Chapter II deals with all system hardware and wiring. Additionally, the general flow of digital inputs and outputs (IO) is explained. Chapter III deals with the user interface, and general use of the system. It includes detailed explanations of all screens and features of the Inspection Program. Accessing the stored data through Microsoft Access is also discussed. Chapter IV considers the code used to create the Inspection Program. PC-camera communications along with program manipulations of the Access database are discussed in this chapter. Physical camera, lens, light, and fixture setup and the camera's program are discussed in the Chapter V.

II. HARDWARE

The system consists of four major pieces of hardware. An IBM-compatible PC and a digital camera handle all inspection data processing. The other two pieces of hardware, a high speed counter and a photo eye sensor, are used to trigger the camera once per two parts. Additional hardware in the system includes several relays and switches, a 7.5W 24VDC power supply, green and red indicating pass and fail lights, and a fail indicating buzzer. All hardware, with the exception of the PC, runs on 24VDC power. The wiring diagram of the system can be seen in Appendix C. A basic block diagram of the system is shown in Figure 1.

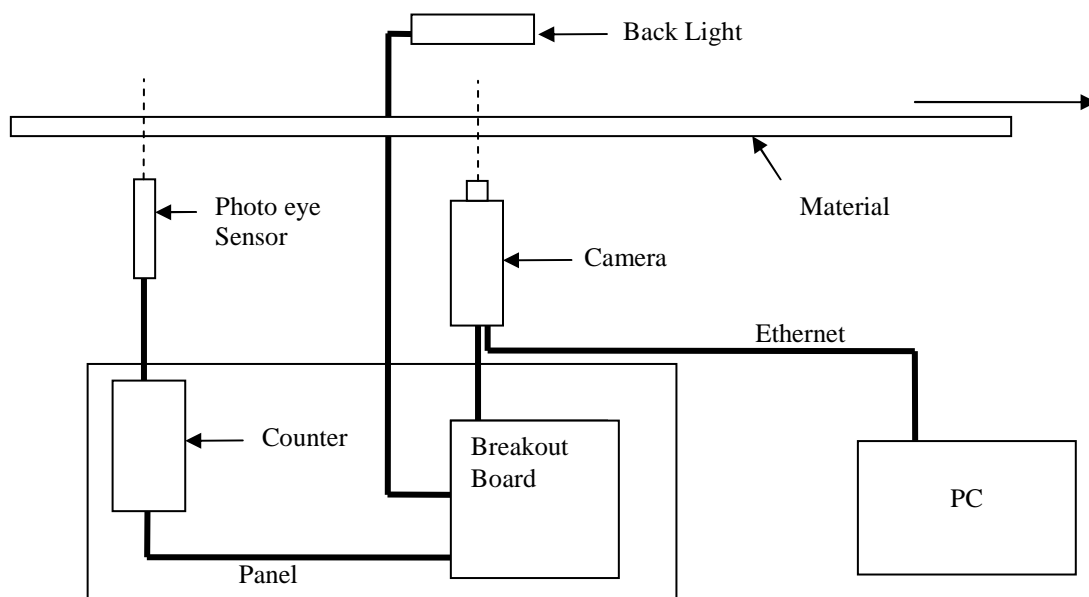


Figure 1. Basic system block diagram.

Personal Computer

The IBM-compatible PC has a Pentium I® processor with a 166MHz clock speed and 128 Megabytes of system RAM. The operating system running on the PC is Microsoft® Windows® 98 Second Edition. The computer receives information from the DVT camera through a crossover Ethernet cable attached to its network card.

Camera and Lighting

The 630 Smart Sensor Camera®, manufactured by DVT® Corporation, uses DVT's LUD-25F 25-mm focal distance lens with a 1-mm spacer ring. A 15 pin high density (D-Sub) cable connects the camera to the DVT breakout board. The DVT breakout board serves as the central connection point for the system's digital IO. The pass and fail indicating lights are controlled by IO2 (Pin 4) of the breakout board through a latching relay that has both a standard and inverted output. The indicating buzzer is controlled by IO8 (Pin 10) of the breakout board, also through a latching relay. Both latching relays are set to latch on a fail condition and are reset through a momentary switch on the panel door.

The system uses the IDRA-D Diffused Red LED Array, also manufactured by DVT, as the illumination source for vision inspections. This device receives 0VDC from DVT breakout board Pin 15 on its blue wire and 24VDC from DVT breakout board Pin 16 on its brown wire. The black strobe input of the IDRA-D is connected to IO12 (Pin 14) of the breakout board.

Triggering Hardware

A WF 15-B4150 15-mm photo eye, manufactured by SICK, Inc., is set to trigger when the light path is uninterrupted, outputting a high signal. This sensor is designed

like a fork, with the light emitter on one part of the fork and the photoelectric sensor on the other part. The material passes through the 15-mm space between the fork tips. The photo eye triggers when the hole in the material's carrier, as seen in Figure 2, passes in between the fork tips. The photo eye is set to have a sourcing output. The brown wire is wired to 24VDC and the blue wire is wired to 0VDC. The black output wire is wired to the high speed counter.

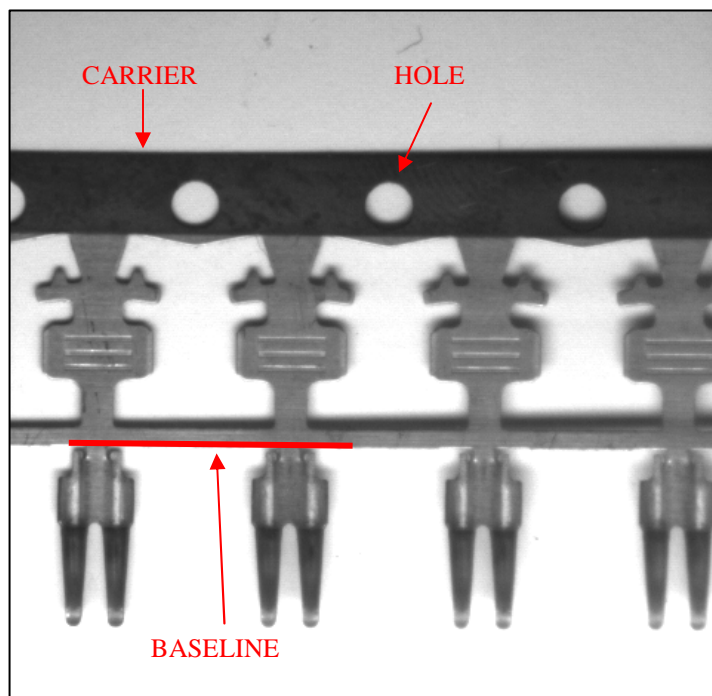


Figure 2. Sample of material.

The NE131.02.3AX01 Programmable Counter, manufactured by IVO, is wired to receive the output of the SICK photo eye through pin 4. The count signal received is a differential input between pin 4 and pin 3 on the counter. A +12VDC or higher differential between these pins will cause the counter to increment. 0VDC is wired to

pin 3. The counter is powered by 24VDC to pin 1 and 0VDC to pin 2. The output of the NE131 is a physical relay. The relay input, pin 7, is wired to 24VDC. The output of the NE131 is wired to pin 6 for a normally open output. Pin 6 is wired to IO1 (Pin 3), the trigger input of the DVT breakout board. All other pins on the NE131 are left unconnected. The programmable counter is programmed to reset the count and output a 10-ms pulse signal when the count reaches two. This causes the DVT Camera to be triggered one time for every two parts that pass by the photo eye.

III. USER INTERFACE

The system is operated through the Inspection Program running on the PC. This Inspection Program consists of three screens: the Setup screen, Main Screen, and Parameters screen. Additionally the user is able to view the data collected by the system by opening created database files in Microsoft™ Access 2000®.

Setup Screen

The Setup screen, shown in Figure 3, is accessed by holding “control” and pressing “s” on the keyboard during the program start up while the Splash screen is shown. Engineering parameters can be changed from this screen. The Database Path contains the path where the Access database will be stored on the PC. The path can be changed to point to a database file that does not yet exist or a preexisting database. The program will automatically create a new database if the file does not exist. Camera IP Address contains the Ethernet Internet Protocol (IP) address of the DVT camera being used in the system. Scale contains the multiplier value used to convert DVT measurements (given in pixels) to thousandths of Inches before it is stored in the database. All changes made to the Setup screen are remembered each time the program is run until they are changed again.

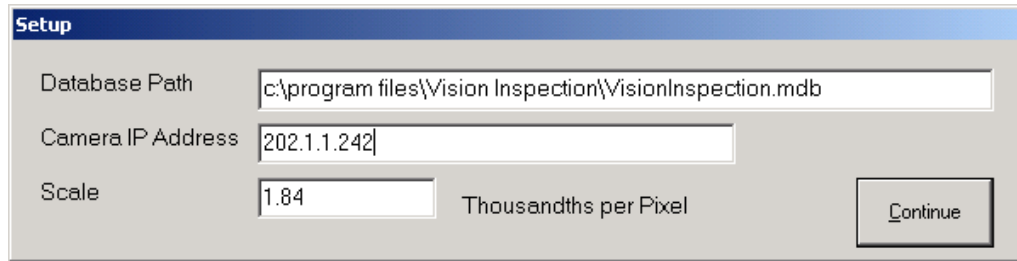


Figure 3. Inspection Program Setup screen.

Main Screen

The Main Screen, shown in Figure 4, gives general feedback to the user during normal operation and allows access to the Parameters screen. This program uses a concept of open and closed reels. Only one reel is allowed to be open at any given time. The open reel is the reel which is currently being inspected. Once a reel is closed, or ended, it cannot be opened again without editing the database. Any data the Inspection Program receives from the camera while there are no open reels is ignored. While a reel is open and inspections are running, the Main Screen looks like Figure 4. When all reels are closed the Main Screen looks like Figure 5, with all the Cumulative Analysis data showing “?” and both displays showing WAITING FOR INSPECTION.

The data shown in Cumulative Analysis is pertinent to the currently open reel only. The Current Inspection updates the most recent image taken by the camera as quickly as possible. Due to the high rate of inspections, the low processing power of the PC running the computer, and limitations of the DVT camera’s processing power, not every image can be seen on the Current Inspection display. It is capable of displaying approximately two images per second. In most cases, all of the failed images are updated on the Most Recent Failed Inspection display. The Most Recent Failed Inspection display may have

similar problems to the Current Inspection display in situations where there are multiple failures in quick succession.



Figure 4. Inspection Program Main Screen with a reel open.

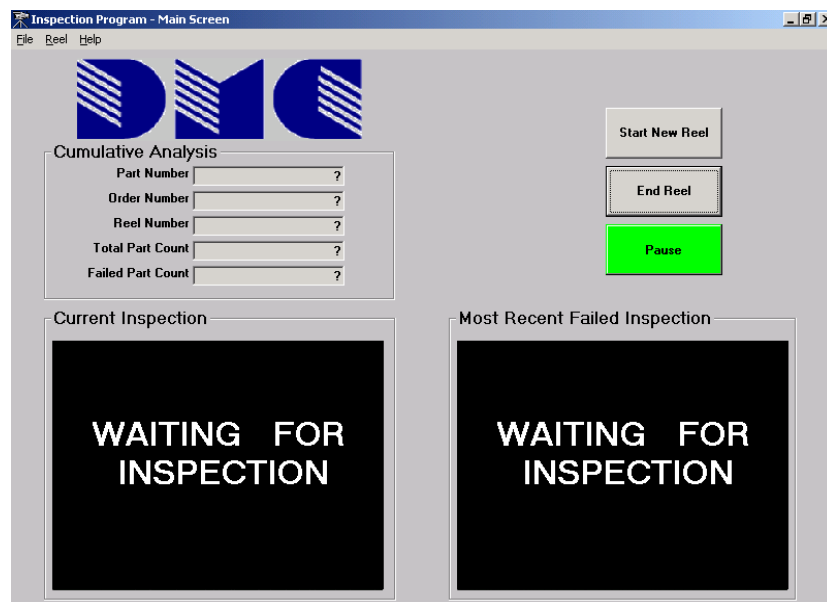


Figure 5. Inspection Program Main Screen with all reels closed.

To create a new reel, the Start New Reel button is pressed, which opens the Parameters screen. The program does not allow a new reel to be created if there is already an open reel. When a reel is finished being run, the operator presses the End Reel button to permanently close it. When the pause feature is used, any inspection information received from the DVT camera is ignored. This allows adjustments to be made to the reel, which cause inadvertent camera triggers, without adding false failures to the inspection database. To use the pause feature, the user presses the green Pause button. The button then turns red, indicating that the system is not running, and the caption on the buttons changes from Pause to Resume. To turn off the pause feature, the button is pressed again.

Under File on the menu bar, the user can launch DVT Framework 2.3, manually force the camera to inspect, or exit the program. Under Reel on the menu bar, the user has access to the same functions the buttons allow: Start New Reel, End Reel, and Pause/Resume. Under Help, the About screen can be displayed, which shows the program name, program version number, and support contact information.

Parameters Screen

When the operator starts a new reel, the part, order, and reel numbers are all entered into the Inspection Program through the Parameters screen, shown in Figure 6. To facilitate quick change over to a new reel, the part number and order number of the most recent reel are already automatically selected on this screen. Additionally, any previously entered parameters for all three numbers can be chosen from the combo boxes by clicking on the arrows. These previously entered values are obtained from the database. Alternately the user can enter an entirely new value through the keyboard.

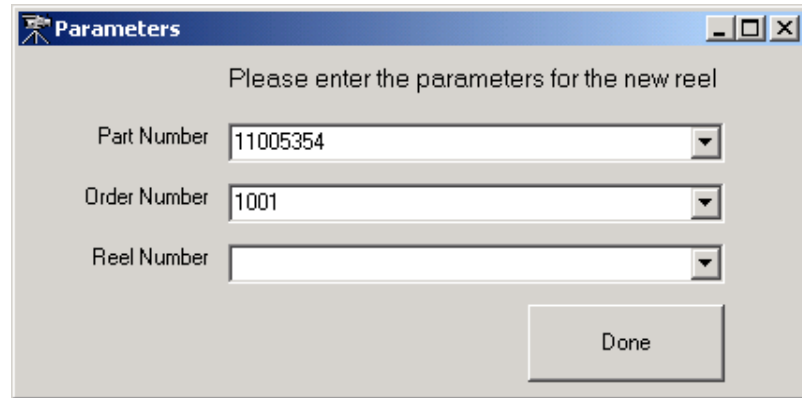


Figure 6. Inspection Program Parameters screen.

When all parameters are satisfactory, the user presses the Done button. The part number and order number are checked against preexisting information in the database. If there is already a reel with both an identical part and order number, an error message appears, stating, A Reel with these values for Order Number and Reel Number already exists. Please type in a unique value. The part and order numbers are reset to the last valid entries and the reel number is cleared. If the user leaves any fields blank and presses the Done button, the Inspection Program will ask for a value to be entered into all fields before proceeding. Alphanumeric characters and spaces are allowed to be entered for all three parameters.

Access Database

All data recorded by the program is entered into a Microsoft Access 2000 database, specified in the Setup screen. The Inspection Program does not allow the user to access inspection data besides overview information for the currently open reel displayed on the Main Screen. To access detailed information on specific part failures within a reel, or on previously closed reels, the user must open the database with

Microsoft Access 2000, Microsoft Access XP, or a compatible program. The purpose of this database is to allow the user to perform any data analysis desired using tools contained within Microsoft Access, such as SQL statements and queries, or by exporting the data to other software packages.

There are two tables in the inspection database: the failure table and reel table. To open either one, the user first opens the database file and then double-clicks on the table in the screen shown in Figure 7.

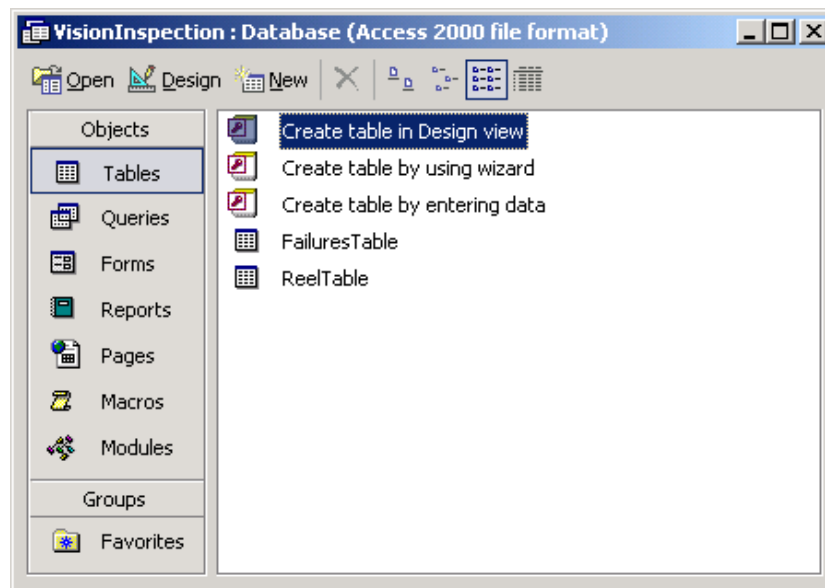


Figure 7. Access database tables.

Figure 8 shows a sample reel table with two reel entries. The Index field is an integer field that automatically increments for each new entry. This field is used to link the reel table with the failure table. Start Time records the date and time when the reel was first entered into the table, i.e., the time when the Done button is pressed on the Parameters screen. The End Time field contains the time when the reel is closed, i.e., the

time when the End Reel button is pressed on the Main Screen. A blank value for End Time signifies that the reel is still open.

Index	Order Number	Reel Number	Part Number	Start Time	End Time	Total Part Count	Failed Part Count
1	1001	1	11005354	1/14/2001 10:29:40 AM	14/2001 10:29:58 AM	3456	3
2	1001	2	11005354	1/14/2001 11:03:27 AM		354	0

Record: 3 of 3

Figure 8. Reel table.

The failure table contains detailed measurement information about every part that fails the inspection. A sample failure table is shown in Figure 9. The Index field is set to be identical to the Index field of the corresponding reel in the reel table. Sequence Number contains the sequence of the part being inspected from the beginning of the reel, with the first part being numbered 1. From the camera’s perspective, the part to the right is the first part, and thus will have a sequence number one less than the part to its left. Time contains the date and time at which the inspection of that part took place. Tip 1 Measurement and Tip 2 Measurement fields store the distance, in thousandths of an Inch, from the tip of the part to the baseline. The baseline, shown with a red line in Figure 2 on page 6, is simply a datum point used for measurement reference. Tip 1 is the tip farthest to the right from the camera’s perspective.

Index	Order Number	Reel Number	Sequence Number	Time	Tip 1 Measurement (in thousandths)	Tip 2 Measurement (in thousandths)
1	1001	1	6	11/14/2001 10:29:42 AM	13.9	12.1
1	1001	1	17	11/14/2001 10:29:48 AM	15.7	18.4
1	1001	1	153	11/14/2001 10:29:56 AM	87.6	88.2

Record: 4 of 4

Figure 9. Failure table.

IV. PROGRAM IMPLEMENTATION

The Inspection Program is written in Microsoft® Visual Basic® 6.0 (with Service Pack 5). The source code uses three add-ins: DMC ini File version 1.1, Far Point Button Objects, and DVT SID Active X Control 1.1.

Key Controls

Embedded within the Main Screen are five important control objects which are used by the Inspection Program to communicate to external objects. Four of these objects connect to the camera while the fifth connects to a text file used to store settings information.

DVTSID(0) and DVTSID(1)

There are two DVTSID objects (DVT Sampled Image Display): one for the Current Inspection display (DVTSID(0)) and one for the Most Recent Failed Inspection display (DVTSID(1)). Throughout the program, the connection is left active, but the DVTSID(0).StopImages and DVTSID(0).PlayImages(All) (or DVTSID(1).PlayImages(Fail_Only) for the second DVTSID control) commands are used in order to toggle showing live images. If these displays are set to play images, they will show a new image every time the camera's image updates, regardless of whether or not corresponding inspection data is received by the Inspection Program.

DataLinkWinsock(0)

`DataLinkWinsock(0)` is a Microsoft® Windows Socket (Winsock®) control set to receive inspection data from the DVT camera's Data Link port. To receive this data the Winsock's remote port is permanently set to 5001 and protocol is permanently set to 0 (TCP) from the properties window. The camera's Data Link at port 5001 is only capable of outputting strings that are custom set in Framework; it is not capable of receiving data from the Inspection Program.

Winsock(0)

`Winsock(0)` is a Winsock control that connects to the DVT camera's command terminal. Its properties are set the same as the Data Link port, except it connects through the camera's command terminal at port 5000. The command terminal can be a very powerful tool, but requires a large amount of string parsing code to handle the different strings of information it can return. However, in this program, the command terminal is only used to manually force the camera to trigger from the File menu on the Main Screen. This operation is only used to debug or setup the system.

Inifile1

The fifth crucial control on the Main Screen is the inifile control. The inifile control was developed in-house at DMC for previous projects. It is designed to write and read data from an ini file; a file where program settings are stored and available for use in between sessions. The inifile control in the Inspection Program, `Inifile1`, is linked to the file `Vision Inspection.ini`, stored in the program directory. Contained within this file are all of the settings from the Setup screen, along with the most recently selected order and part numbers.

Program Startup and Initialization

Upon starting the program, the Splash screen appears which gives the user the option to hold “control” and press “s” during the five seconds it is displayed to bring up the Setup screen. If the Setup screen is displayed it will show the most recently entered parameters, loaded from the inifile. After the user is done entering parameters, the new parameters will be stored back into the inifile and the global variables corresponding to each parameter are set. If the user does not enter the Setup screen, the global variables are set from the previously stored inifile values.

Next the Main Screen form is loaded, using the command `frmMainMenu.Show`. This causes the Main Screen’s `Form_Load` subroutine to run, shown in Figure 10 (a flowchart for `Form_Load` is shown in Appendix B-1). The very first line of this subroutine calls the function `Initialize`. If `Initialize` runs successfully without errors, then it returns a `True` value, allowing the Inspection Program to exit the while-loop and execute the central portion of the program. If an error does occur, a message box is shown, telling the user to check cable connections and make sure the DVT camera is completely booted. The program then loops back and shows the Setup screen.

The `Initialize` function, shown in Appendix A-1, sets up connections to most controls and the database (a flowchart for `Initialize` is shown in appendix B-2). At the beginning of this function, the first assumption is that all reels are closed, thus the `Paused` and `ReelOpen` variables are both set to `False`. These variables are changed at the end of this function if there is found to be an open reel in the database. `Inifile1` is set so that it is ready to be read from or written to later on. The `ClearPassFailImg` subroutine is called every time all reels are closed.


```

Private Sub Form_Load()
'Initialize is a function. A returned false value means an error occurred
'during initialization
While Initialize = False

    selection = MsgBox("Setup Error. There was an error during initializing " & _
        "connections to the camera and the database. Please check to make " & _
        "sure that all cables are connected, the cameras are on and fully " & _
        "booted before reconnecting. Press OK to Enter Setup.", vbOKCancel)

    If selection = vbOK Then
        frmSetup.Show 1 'Go back to the Setup Screen. Show it modally.
    Else
        End          'End Program
    End If
Wend
    frmMainMenu.Show
End Sub

```

Figure 10. Main Screen Form_Load subroutine.

The next several lines of code deal with control objects that need active connections to the DVT camera. The `Initialize` function first checks to see that each of these controls are disconnected and if they are not it disconnects them. Next it sets the remote host property equal to the value set for the DVT camera's IP address in the Setup screen. All of the devices are then connected through four separate calls to the same `ConnectToWinsock` subroutine, each time passing the control as a Winsock by reference. This subroutine contains a simple loop that continuously attempts to connect to the remote host until either a connection is made or a timeout value of one second has passed. There is a 10-ms delay between each attempt to connect. If any of these connections fail, the `Initialize` function will return a `Fail` result.

Next the function calls the `OpenDatabase` subroutine. This subroutine checks for the existence of the database file specified in the Setup screen, and both the reel and failure tables within it. It creates the database and tables if they do not exist. The subroutine then links the tables each to a DAO record set.

The `Initialize` subroutine ends by calling `CheckForOpenReel`, which searches for open reels with an SQL statement. If this subroutine finds an open reel, it sets that reel to the currently opened reel, pauses the program, and warns the user with a message box. The logic behind this portion of the code is a large part of the error handling. It is only possible for one reel to be open at a time, with the exception of manually editing the database, which is a situation the program does not handle. If the program is either closed with an open reel, or if the system goes down unexpectedly, the Inspection Program will not lose track of reels that are left open.

Database Operations

In the Inspection Program, a major part of handling data is managing the databases. This is done through the use of Microsoft Data Access Objects (DAO) and Structured Query Language (SQL) statements. Figure 11 shows the DAO related variable declarations that can be found in the declarations section of the Inspection Program's `DatabaseOperations` module. A `DataBaseWorkSpace` is a workspace, needed to setup a database. `DataBase1`, the active database, is a subset of that workspace. The tables must be assigned to record sets in order to be manipulated using DAO and SQL. Record sets also can be filled using SQL queries. In DAO, record sets are a subset of databases.

A major portion of the `OpenDatabase` subroutine, which initializes the database and record sets, can be seen in Appendix A-3. First the workspace is set. Then, if the database, `Database1`, does not exist, it is created and set using the `CreateDatabase` command. If the database already exists, it is set using the `OpenDatabase` command. After the database is set, the subroutine checks for the

existence of ReelTable within the database. If it is not found, the table is created by calling the subroutine CreateReelTable, which is shown in Appendix A. After ReelTable is either created or found to already exist, the record set by the same name is set to it.

```
'DAO Variables
Public DataBaseWorkSpace As Workspace

'Active Database containing Reel and Failure Tables
Public DataBase1 As Database

Public ReelTable As DAO.Recordset      'RecordSet containing ReelTable
Public FailuresTable As DAO.Recordset  'RecordSet containing FailuresTable

'Temporary RecordSet used in several misc. queries
Public TempRecordSet As DAO.Recordset

'RecordSet used in query to find OpenReels
Public OpenReels As DAO.Recordset
```

Figure 11. DAO variable declarations.

SQL statements are used multiple times in this program for different reasons, such as searching for open reel records, grabbing all possible values of a field, and making sure that the correct record is selected before editing it. Figure 12, which shows part of the Parameters screen form's loading routine, contains a SQL query example. The code shown is used to fill the order number combo box on the Parameters screen with all previously entered values for the order number. The temporary record set is filled with distinct values of the field Order Number, using the `distinct` command in the SQL statement. The for-loop loads each order number value into the combo box. This is done using the `MoveFirst` command to move to the first record in the temporary record set, and then using `MoveNext` command to move to each subsequent record. The

Distinct command in the SQL statement ensures that there are no duplicate order number values in the combo box.

```
'Set Order Number Combo Box--place all previously entered values for Order
'Number (taken from the database) into the Order Number Combo Box on the
'Parameters Screen.
Dim sql As String

'This query creates a RecordSet that only contains DISTINCT values for the
'Order Number field. That means that if there are several records with the
'same order number, only the first appearing record is taken and the rest
'are discarded. This will keep redundant values from being placed in the
'combo box.
sql = "Select Distinct [Order Number] from ReelTable"
Set TempRecordSet = DataBase1.OpenRecordset(sql, dbOpenDynaset)

If TempRecordSet.RecordCount > 0 Then
TempRecordSet.MoveFirst 'Move to first record in recordset RecordSet
  For i = 0 To TempRecordSet.RecordCount - 1
    cmbOrderNumber.AddItem TempRecordSet("Order Number")
    TempRecordSet.MoveNext 'Move to next record in the recordset
  Next i

  'RecentOrderNumber contains the most recently selected Order Number. If
  'There is a value for this, set the combo box to it.
  If RecentOrderNumber <> "" Then
    Call SetCombobox (RecentOrderNumber, cmbOrderNumber)
  End If
End If
```

Figure 12. Excerpt from Parameters screen Form_Load subroutine.

Another example of a SQL query is shown in Figure 13. Figure 13 is a sample of code from the Parameters screen's cmdDone_Click subroutine (the subroutine that runs after the user has clicked the Done command button). Before creating a new record from the selected values of order number, part number, and reel number, the Inspection Program must first check that there is not already a record with identical values for both order number and reel number. The SQL statement shown in Figure 13 puts any such records found in the record set DuplicateNames. If one or more records exist in DuplicateNames, the subroutine forces the user to select different values.

All the data received from the camera must be stored to the database. This is done by adding and editing records in the tables. Figure 14 shows the CreateReelRecord subroutine, which is called to add a new blank record to the reel table. The general structure of adding a new record is to use the Table.AddNew command, followed by commands filling in all of the field values, and finished with the Table.Update command. The process for editing a record is very similar. First the correct record needs to be selected using an appropriate command, such as ReelTable.FindFirst[SQL statement]. Then the same procedure shown in Figure 14 is used, except replacing the Table.AddNew command with a Table.Edit command.

```
'Do Query to make sure there are no values already in the table with
'identical values for both Order and Reel number that has been selected
'by the user.
  Dim sql As String

  'cmbOrderNumber.Text is the value typed in the Order Number box by the user and
  'cmbReelNumber.Text is the value typed in the Reel Number box.
  sql = "Select * from ReelTable where [Order Number] = '" & _
        cmbOrderNumber.Text & "' and [Reel Number] = '" & cmbReelNumber.Text & "'"

  ' Generates the following string, sql:
  ' Select * from ReelTable where [Order Number] = 'cmbOrderNumber.Text'
  '                               and [Reel Number] = 'cmbReelNumber.Text'

  'Perform the query. Any records matching the criteria will be placed into a
  'new RecordSet named DuplicateNames
  Set DuplicateNames = DataBase1.OpenRecordset(sql, dbOpenDynaset)

  'If there are any records in DuplicateNames, then the values selected by the
  'user exist already and thus are invalid.
  If DuplicateNames.RecordCount > 0 Then
```

Figure 13. Excerpt from Parameters screen cmdDone_Click subroutine.

```

Public Sub CreateReelRecord()
    'This subroutine simply creates a new, blank record.

    ReelTable.AddNew
        ReelTable("Order Number") = frmParameters.cmbOrderNumber.Text
        ReelTable("Reel Number") = frmParameters.cmbReelNumber.Text
        ReelTable("Part Number") = frmParameters.cmbPartNumber.Text
        ReelTable("Start Time") = FormatDateTime(Now, vbGeneralDate)
        ReelTable("End Time") = Null
        ReelTable("Total Part Count") = 0
        ReelTable("Failed Part Count") = 0
    ReelTable.Update

End Sub

```

Figure 14. CreateReelRecord subroutine.

Inspection Data Reception and Handling

The Inspection Program is set to receive data through a Winsock control named DataLinkWinsock(0). Whenever data is transmitted by the camera through this Ethernet TCP connection, the subroutine DataLinkWinsock_DataArrival, shown in Figure 15, is automatically executed. Within this subroutine and all the other routines that it calls, all of the data processing takes place. The flowchart for the data handling is shown in Appendix B-3.

```

Private Sub DataLinkWinsock_DataArrival(Index As Integer, ByVal bytesTotal As Long)
    Dim ResultString As String
    Dim CameraNumber As Integer

    'Received data is put into ResultString
    DataLinkWinsock(Index).GetData ResultString, vbString

    'If the program is not paused and there is a reel open, then process inspection.
    'Otherwise discard the data.
    If Not Paused And ReelOpen Then
        Dim i As Integer
        For i = 0 To 3
            'This clears all "WAITING FOR INSPECTION" and other such images
            'on the display if they are not already cleared.
            txtSIDClear(i).Visible = False
        Next i
        Call HandleNewInspection(ResultString)
    End If
End Sub

```

Figure 15. DataLinkWinsock_DataArrival subroutine.

The data is first read with the `GetData` command before the program checks for the paused state and open reels. The `GetData` command must be executed, even if the program is paused and all reels or closed. This makes certain that the data is cleared out and ignored, instead of being received the next time the `DataArrival` event occurs. Data is received as a string with eight values separated by commas followed by a colon. The colon is used as a delineator between inspections. This is done so that if the Inspection Program happens to miss data transfer for any reason, such as being in stuck in modal mode, the program will still be able to handle an extended string containing two or more inspection results.

The `HandleNewInspection` subroutine, shown in Figure 17 loops through three subroutines until the string containing the received inspection data, `ResultString`, is empty. `SeparateInspection` first looks for the first colon from the left, and breaks the string at that point (all done using the `Split` command). The data to the left of the colon, which would be the first inspection to have occurred chronologically, is placed in the variable, `OneInspection`. The remaining data is placed back in `ResultString` and the colon is discarded. Next the comma separated string, `OneInspection`, is separated into an array of eight singles (four-byte floating-point numbers). Finally the array is received by `AddNewInspection`, which uses it to determine the actual results and edit the tables accordingly. All variables passed from `HandleNewInspection` to the three subroutines it calls are passed by reference, except for `ProductResults`, which is available to all routines because it is declared as a public variable.

```

Public Sub HandleNewInspection(ByRef ResultString As String)

    Dim ResultsNumber As Long
    Dim i As Integer
    Dim OneInspection As String

    Do While (ResultString <> "")

        'Data is passed through by reference in ResultString. The first inspection is
        'removed from ResultString and returned through OneInspection
        Call SeparateInspection(ResultString, OneInspection)

        'The OneInspection is sent as a comma seperated string and is returned as an
        'array of singles through ProductResults(). True is sent as a boolean to tell
        'the subroutine to dimension ProductResults(). ResultsNumber is returned with
        'the length of ProductResults().
        Call StringToArray(OneInspection, ProductResults(), ResultsNumber, True)

        'The data is checked and based upon the results the tables are edited and
        'updated.
        Call AddNewInspection

    Loop

End Sub

```

Figure 16. HandleNewInspection subroutine.

V. CAMERA SETUP AND PROGRAMMING

The setup of the main sensor of the system, the DVT camera, is a crucial part of the overall system. The physical setup and positioning of the DVT camera and related items are the first half of this portion of the system. Achieving reliability and repeatability in images captured by the DVT camera depends primarily on the DVT camera positioning, lighting, and lensing, along with material fixturing. The second half of this system portion is programming the camera. All of the image processing occurs within the DVT camera's embedded controller. Besides taking measurements and determining the pass/fail result of each part, the DVT camera acts as a central point for IO.

Camera, Lens, Light, and Fixture Setup

The DVT camera is set up to find pins bent in the direction perpendicular to the plane of the material. The DVTSID display on the right in Figure 4 on page 10 shows an example of a bent pin. The pin on the right in this image is bent downward compared to the pin on the left. The camera is set to detection of bent pins possible by viewing the part nearly head on with the pins. The camera is angled 10 degrees off axis from the part, so that from the DVT sensor's aspect, a perfectly straight pin will be pointed slight below the rest of the part.

A picture of the fixture and lighting is shown in Figure 17. The DVT camera uses diffused backlighting from the DVT IRDA-D light. Backlighting is used because it

provides optimum contrast between the part and its background. In an ideal backlighting situation, the object being inspected becomes a completely black silhouette on a completely white background. This gives a large amount of contrast and high gradients at the edges of the object, thus making edges easier to find by vision inspection software. The backlight is a red LED array that comes with a thin plastic diffuser on the front. In addition to this, a larger white piece of plastic is put in front of the light to cause further diffusion. The light is diffused so that it is spread evenly across the camera's field of view. Without diffusion, the LED array appears as several bright circles (one circle at each LED) as opposed to one bright, uniform plane.

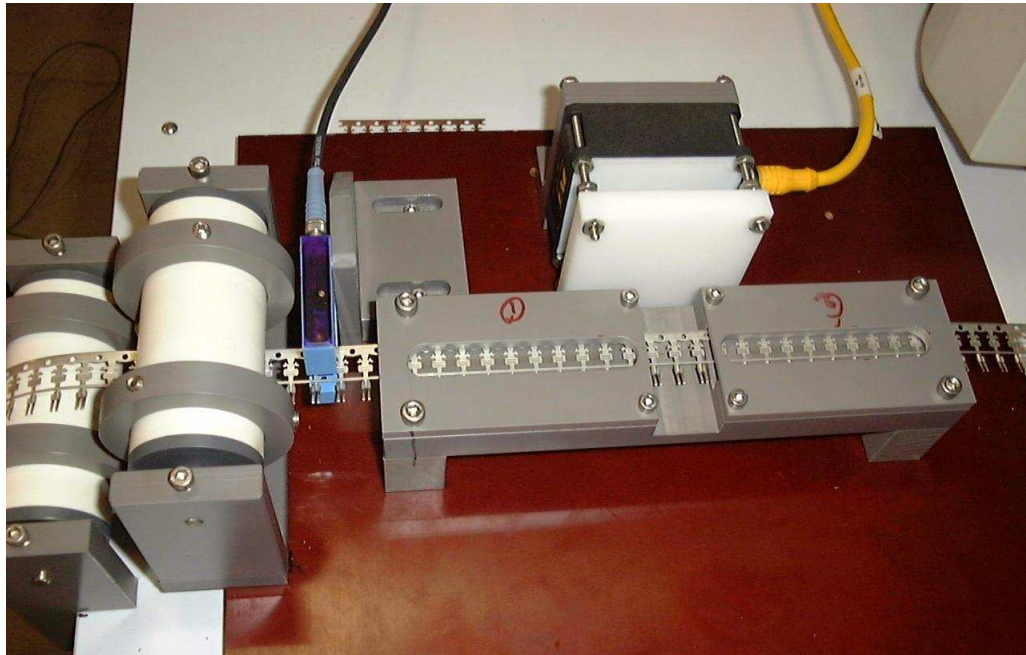


Figure 17. Lighting and fixture.

The lens used for the camera is a DVT LUD-25F 25-mm lens. It has variable focus, but no aperture control. A 1-mm spacer ring is placed between the camera and the

lens. The spacer ring is not necessary to achieve focus; however it allows the lens to be screwed in more tightly when focused. This makes the lens focus more stable and less likely to change if the camera is bumped. The focus is set by looking at a still part and adjusting the lens until the focus is optimized at the ends of the part's tips. The exposure time is set to .7 ms using the DVT Framework software. This gives good contrast between the bright background and the dark part and is more than short enough to keep the part from blurring significantly at even the highest speeds.

To assure that the part does not blur at high speeds, the number of pixels that the part will move during the exposure time is calculated. If this value is greater than one pixel, it can be assumed that there will be at least minor blurring. The speed of the part in pixels per second is determined by dividing the speed (30 feet per minute or 6 inches per second) by the resolution (.00184 inches per pixel). The distance the part travels during exposure time is calculated by multiplying the speed (approximately 3,260 pixels per second) by the exposure time (.0007 seconds). The part moves a maximum of 2.3 pixels. Even when the line is run at top speed, this blurring is barely noticeable. Furthermore, because the movement and blurring occur in the horizontal direction and the precision measurements are taken in the vertical direction, there is no effect on measurement precision and accuracy.

The part is placed in a firm fixture so that it will have consistent placement within the camera's field of view at varying speeds. Rollers to the right and left of the fixture guide the material through the fixture. The fixture has a hollowed out area that the material travels through. In the center, the fixture is completely opened up so that the camera has an unobstructed view of the part.

DVT Program

The DVT camera runs version 2.3 of firmware, and thus is programmed through a PC that runs Framework 2.3. The DVT camera is programmed to perform vision inspections using “soft sensors”. A soft sensor can be anything from a detection tool or positioning tool of some sort that is drawn on the image, to an invisible distance sensor that gives the distance between two other sensors, to a script sensor, that is a bit of code written to perform a task. These soft sensors can be referenced off each other, to form a chain of dependencies; i.e., sensor 1 is positioned off sensor 2, which in turn references sensors 3 and 4. A reference can be a position reference, where, for example, sensor 1 might be moved three pixels to the left and rotated 90 degrees based upon the results of sensor 2. Or a reference can be a soft sensor that is referred to as a variable in a script sensor. Just as with most programming languages, soft sensors are not allowed to have circular references. Every group of soft sensors configured to inspect a part make up one system program, called a “product.” Although the camera is capable of storing and switching between multiple products, only one program is used in this application.

The images of the parts were found to move around a fair amount in the screen during actual inspections, depending upon the speed of the line. The cause of this was not determined, though some possibilities include delay in the mechanical relay of the high speed counter, or inconsistency in the point at which the photo eye was triggering. The field of view is wide enough so that there can be as many as three parts displayed at one time. This ensures that there will always be at least two parts available for inspection. Because of the movement, the general positioning soft sensors must have a robust way of determining which two parts need to be inspected.

Because the inspections occur at a high rate, the camera must finish capturing the image, digitally acquiring it, performing all inspection logic and measurements, and setting IO before it is time to take the next image. The camera has 133 ms to complete all of these tasks because there are a maximum of 450 inspections per second. The current inspection program has a total inspection time of around 80 ms, never clearing 100 ms.

Soft sensors

MinimumDistance, shown in Figure 18, is a “Precision Measurement: Area Edge Line” soft sensor. This soft sensor has several vertical scan lines set across the area it is drawn in, from bottom to top. Each scan line returns the first pixel it finds with a high enough gradient (set to 8.7 for this sensor). The scan density parameter is set to 20 percent, so that a scan line will be placed on one of every five pixels along MinimumDistance’s width, in order to conserve camera processing power.

MinimumDistance’s measurement parameter is set to “Minimum Distance,” which means it will return the position found by the scan with the lowest vertical position. The purpose of this soft sensor is to find the general position of just one of the parts.

MinDistYScript is a script sensor that has the following lines of code:

```
MinDistYScript.Position.X=MinimumDistance.Position.X;  
MinDistYScript.Position.Y=0;
```

This causes MinDistYScript’s X position to change depending on the X position result of MinimumDistance, while the script sensor’s Y position remains constant. YPosition, shown in Figure 18, is a simple “Translational Positioning: Line” soft sensor that positions based on the result of MinDistYScript. The purpose of this translational sensor is to find the general vertical position of the baseline, depicted in Figure 2 on page 6. The sensors preceding YPosition are used to make certain that this sensor hits the baseline instead of

one of the parts. YPositionYScript, similar to MinDistYScript, sets its Y position off the vertical result of YPosition and leaves the X position constant.

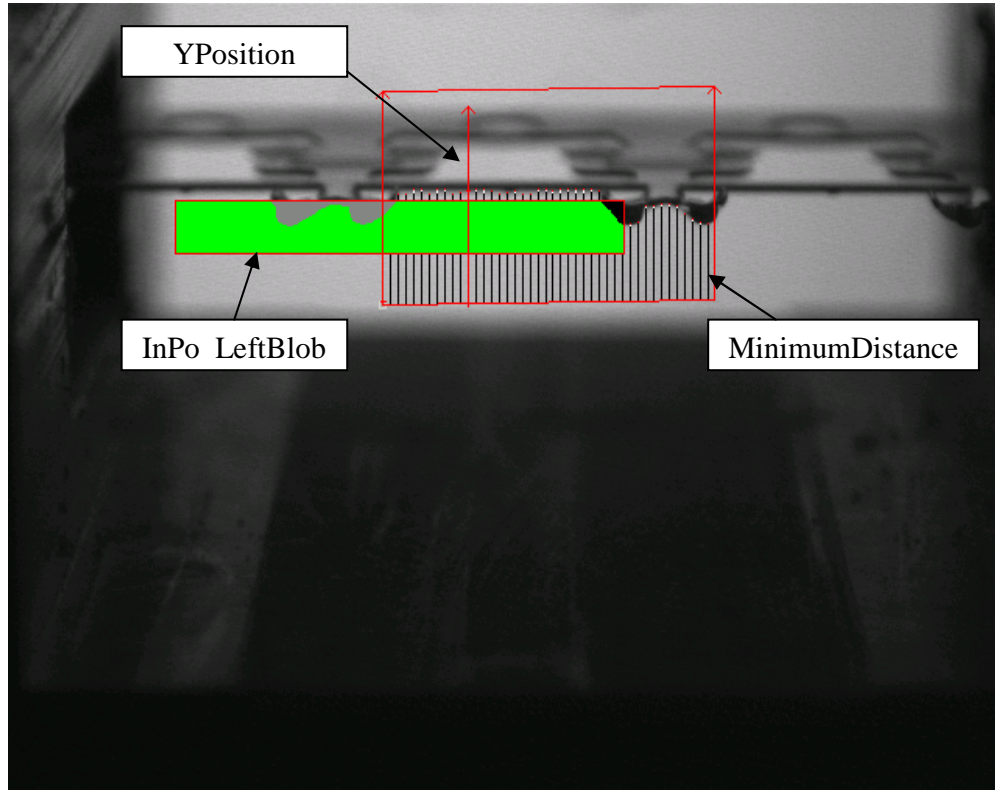


Figure 18. DVT soft sensors (initial chain links).

InPo_LeftBlob, shown in Figure 18, is positioned off YPositionYScript, so that it remains in the same horizontal position, but adjusts vertically to remain just barely below the baseline. InPo_LeftBlob is a “Blob Generator” soft sensor. A “Blob Generator” converts all of the pixels within its area into either light or dark pixels. All pixels with intensity values above a threshold (set to be 60 percent of the contrast of pixels within InPo_LeftBlob’s area) are set to light pixels and all pixels below the threshold intensity are set to dark pixels. The blob sensor is set to search for dark blobs, or groupings of dark

pixels that all touch each other. The objective of this sensor is to generate a blob wherever a part intersects its area. InPo_LBSelect is a “Blob Selector” soft sensor that references InPo_LeftBlob. It selects any number of blobs and filters out the rest based on parameters that are set. The only criterion that is set for blob selection is Bounding Box Width: Start=60, End=160. This means that the and blobs that have a length of less than 60 or greater than 160 will be filtered out and not selected by InPo_LBSelect. The script tool, InPo_BlobScript is a script tool that looks at all the blobs selected by InPo_LBSelect. This script chooses the left-most blob, and uses that blob’s left most pixel for its X position:

```

int i;
int LeftMost;

LeftMost=InPo_LBSelect.BlobBoundingBox.X0[1];
i=2;

while ((i<=InPo_LBSelect.NumBlobs))
{
    if (InPo_LBSelect.BlobBoundingBox.X0[i]<LeftMost)
    {
        LeftMost=InPo_LBSelect.BlobBoundingBox.X0[i];
    }
    i=i+1;
}
InPo_BlobScript.Position.X=LeftMost;

```

InPo_LBSelect.BlobBoundingBox.X0[i] is the blob’s left bounding box position (InPo_LBSelect.BlobBoundingBox.X1[i] would be used to return the right bounding box position). Just like the other two script sensors that have been examined so far, this script sensor changes only one of the positioning coordinates (X) and leaves the other (Y) constant.

The general idea behind the three related “blob” soft sensors is to find the X position of the left-most part that can be inspected. The part cannot be inspected if it is too far to the left, causing one of its pins to merge with the fixture on the side. The

position of the rectangle for InPo_LeftBlob ensures that if a blob is found too far to the left it will be cut off, and that the next closest blob to the right will be far enough into the rectangle. The length criteria of InPo_LBSelect assures that if a blob is too far to the left so that only a small portion of it is in the box, it will not be selected.

BaseLine, shown in Figure 19, is another “Precision Measurement: Area Edge Line” soft sensor, just like MinimumDistance, set to scan vertically from the bottom up. However, the measurement parameter of BaseLine is set to LineFit. This means that the positions found by each scan are used to interpolate a line. This line is considered to be the datum for the final pin distance measurements. BaseLine is set to have a scan density of 35 percent. It is positioned off InPo_BlobScript so that it will always fall directly between the two parts that are to be measured.

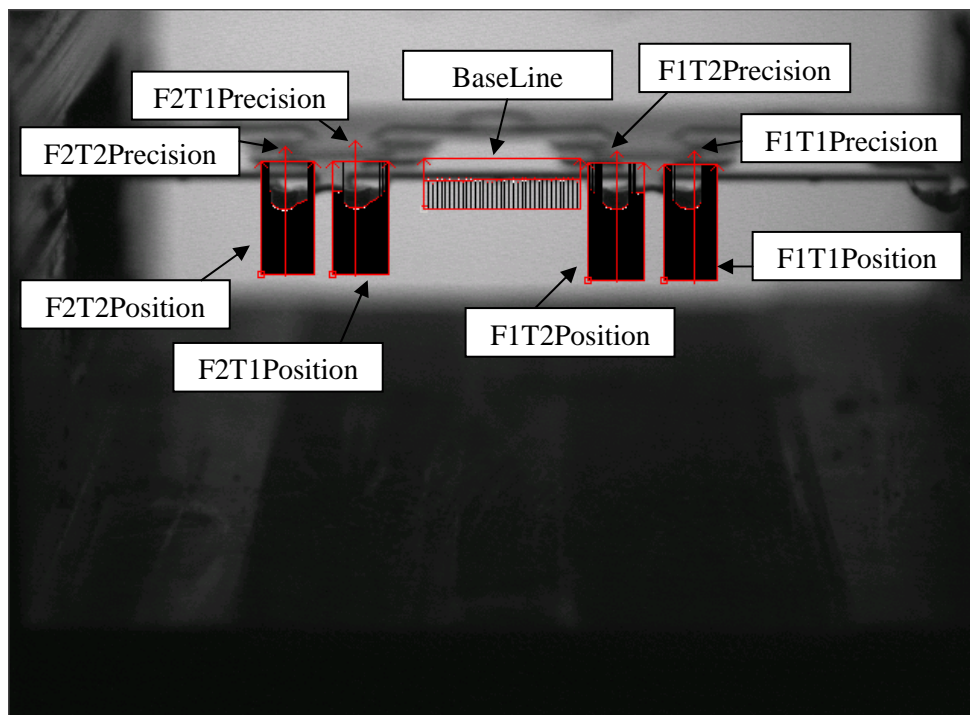


Figure 19. DVT soft sensors (final chain links).

F1T1Position, shown in Figure 19, is a “Precision Measurement: Area Edge Line” sensor set to find the minimum, or lowest Y position of the first part tip (fork 1 tip 1). It is set to 100 percent scan density so that no pixels are ignored in its search. F1T1Position references InPo_BlobScript for its position so that it always lies directly on the correct pin tip. Three other sensors, shown in Figure 19, are identical to F1T1Position (F1T2Position, F2T1Position, and F2T2Position). There is one sensor corresponding to each of the four pin tips.

Referencing off F1T1Position is F1T1Precision, shown in Figure 19. This “Precision Measurement: Line” soft sensor is always positioned to directly measure the lowest point of the pin. There are three identical sensors corresponding to the other fork tips. Fork1Tip1Distance is a “Math Sensor: Distance” tool that references both F1T1Precision and BaseLine. This sensor returns a perpendicular distance in pixels between the point that F1T1Precision finds and the line interpolated by BaseLine. The three other similar soft sensors, Fork1Tip2Distance, Fork2Tip1Distance, and Fork2Tip2Distance, measure a pixel distance for the other three fork tips. Each of the four soft sensors is to be set to have a minimum and maximum distance set. If the measurement does not fall within these values, the soft sensor’s result will be set to WARN and the result value will be set to a positive number. The result value and measurement value are both passed to the Inspection Program through Data Link for each of the four sensors. If the distance is not a passing value, the sensor’s result is set to WARN instead of FAIL, because when the sensor fails, the measurement value is not made available to send through Data Link. The minimum and maximum distance parameters remain to be set by the customer, depending on the tolerances required for the reel being

inspected. However, the passing criteria are likely to be close to the range of 25 to 40 pixels.

The actual measurements taken by the distance sensors are not direct measurements. They are indirect measurements taken in the plane of the camera lens. The part is not directly on-axis with the camera. The slight angle, which is necessary to create background contrast at the pin tips, has two important consequences. First, a perfect, unbent pin does not have a measured value of zero. Second, the measured values are not an exact measurement of the distance a part is bent. However, because the angle of the part with respect to the camera is only 10 degrees, the measured values are a close approximation to the actual distance. The measured value of a pin has the most meaning when it is compared relatively to other pins. For example, stating that a pin's measured distance is 25 thousandths greater than the measurement for a perfect, non-bent pin is probably a good indication that the part should fail.

The script soft sensor, `FinalResult`, references the results of the four distance sensors as variables. If any of the four sensors do not return a PASS value for their result, `FinalResult.Result` is set to FAIL. `FinalResult` is used to return an overall pass/fail result for the inspection for two reasons. The first is so that indicator outputs (lights and buzzers) can be set. The second is to ensure that the `DVTSID(1).PlayImages(Fail_Only)` command works appropriately, allowing all failed images to be displayed in the Inspection Program.

IO setup

The input and output for the DVT camera are configured in the in the I/O Parameters screen. On the Timing tab, shown in Figure 20, the outputs are set to be

immediately available after the trigger, and the output pulse width is set to 50 ms. This means that each time the camera triggers, the relevant outputs are asserted for 50 ms, starting immediately after the camera is done processing the inspection results.

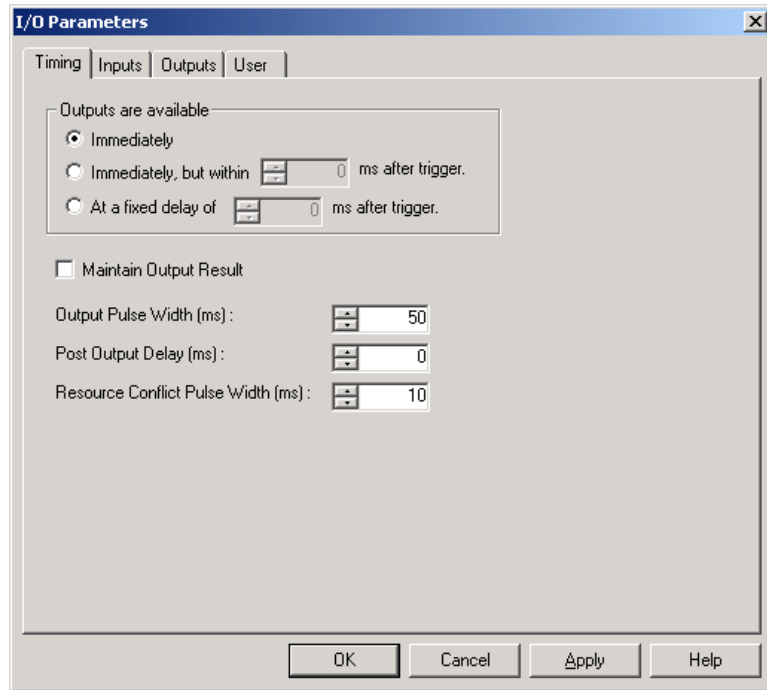


Figure 20. Framework I/O Parameters: Timing settings.

Under the Inputs tab, the only input, Pin 1, is set to Trigger. On the Outputs tab, shown in Figure 21, Pin 2 is set to User1, Pin 8 is set to User2 and Pin 12, is set to Strobe. All other output settings are irrelevant and can be left in their default state.

Under the User tab, shown in Figure 22, USER1 is set to the function value Any, and all soft sensors cleared below it, except for FinalResult, which is set to FAIL. This ensures that this output is asserted only when the FinalResult soft sensor fails. USER2 is set identically to USER1.

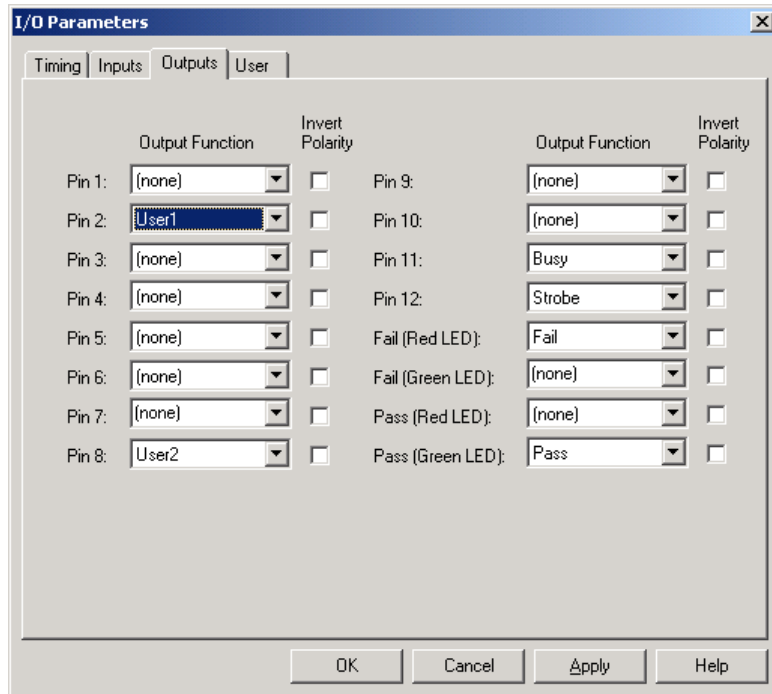


Figure 21. Framework I/O Parameters: Outputs settings.

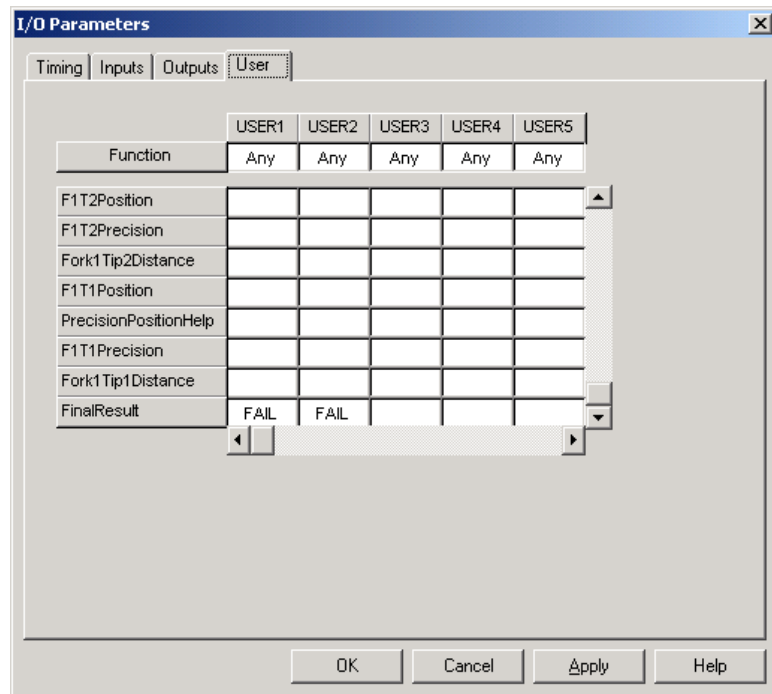


Figure 22. Framework I/O Parameters: User settings.

Inspection data is passed from the camera to the Inspection Program through the DVT camera's Data Link drivers. Data Link is configured through the Data Link Parameters screen. STRING 1 is configured to Always be sent in the Data Link Parameters: Sensors screen, shown in Figure 23. No other strings are used, and thus are all set to Never.

The actual data to be sent through Data Link is set up in the Data Link Parameters: Strings screen, shown in Figure 24. The results and measurements of the four distance soft sensors are sent out in an eight-value comma-separated string with no spaces, and preceded by a colon. The result values sent are signed integers. A positive value indicates that the soft sensor returned a warn value, and negative result represents a failure, and a zero indicates the passed state. The distance measurements are all passed as floating-point values.

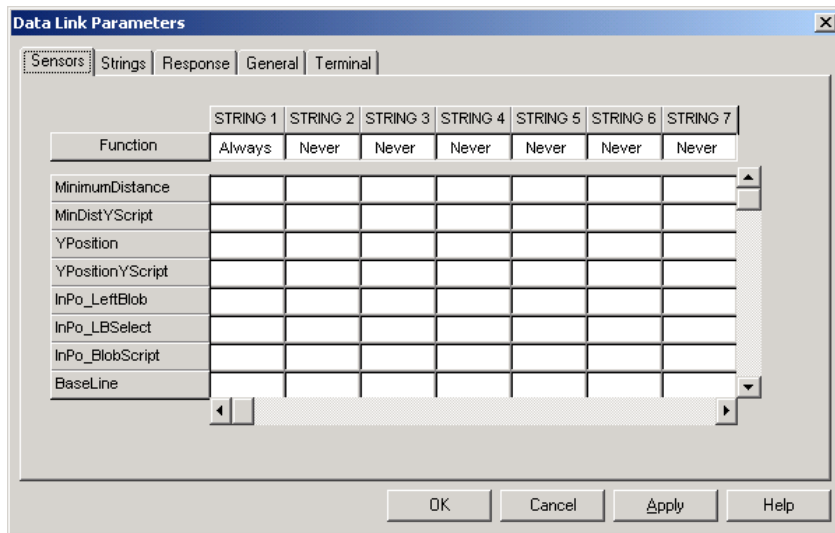


Figure 23. Data Link Parameters: Sensors screen.

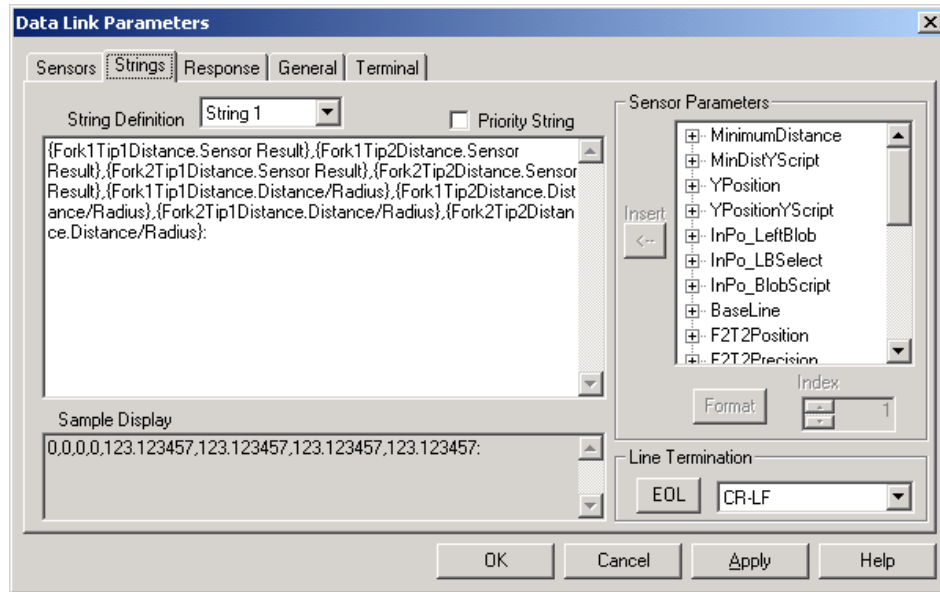


Figure 24. Data Link Parameters: Strings screen.

VI. CONCLUSIONS AND RECOMMENDATIONS

The system described in this thesis is a successful solution to the customer's problem. It was developed within budget and meets all design requirements. The system has not been extensively used yet for the customer's needs, and thus not heavily tested. However, it passed all verification tests.

The system is able to display live images on the PC. However, due to the speed of inspections, some images are never displayed on the PC, even though inspection data from all images are recorded. This is because a DVT 630 camera is not capable of transmitting that much data over Ethernet at a high enough rate to show all images. This is not a major issue as the images are already displayed at a rate faster than can be easily interpreted by an operator (two images per second).

The Inspection Program shows real-time statistics; changes in failed and total part counts are displayed on the screen in less than one second after each inspection when the line is running at full speed. The final system has an easy interface that allows quick entry of parameters and quick changeover to a new reel by the operator.

The system meets resolution requirements; it is capable of accurately detecting variations in image measurement less than .002 inches. The program running on the camera is able to inspect the line at maximum speed without missing parts.

Verification Methods and Results

The software was heavily tested for bugs before final deployment into the customer's site. The overall system was verified by running actual reels through the system. To test all data that the camera was viewing, the system was configured to record all measured pin information to the database rather than just failed parts. A reel with a known number of parts was run through the system to ensure that all pins were inspected. The same reel was sent through the system twice. Both runs gave similar results when the data was compared, proving system repeatability.

Future Recommendations

Ideally, much more rigorous testing and verification would be performed to determine exact levels of system performance. However, the costs of such testing to the customer outweigh the benefits. The system should be run online during actual production. In time additional system flaws may be discovered due to unforeseen conditions occurring, which may require additional modifications to the system.

Currently DMC is working with the customer on an additional project to modify the system to inspect and record data on a completely different part. The camera will be looking at the reel from a different perspective. Modifications to the triggering system, camera mounting, light mounting, part fixture, camera programming, and Inspection Program will permit the system to be changed over to this second inspection mode.

APPENDICES

APPENDIX A

CODE SAMPLES

```

Public Function Initialize() As Boolean

On Error GoTo Handler

    Initialize = False

    'Set the inifile
    iniFile1.FileName = App.Path & "\ " & IniFileName

'Set Reel Closed Parameters

    'Default states set as if there are no open reels.
    Paused = False
    ReelOpen = False

    'Blanks all Cumulative Analysis text boxes with "?" and both DVTSIDs with
    ' "WAITING FOR INSPECTION" labels.
    Call ClearPassFailImg

'Make all Connections

    'These commands close any existing connections. They are only executed
    'if the state is NOT zero (zero is the disconnected state). These are
    'needed for when this subroutine is run multiple times (due to any
    'initialization errors.)

    If DataLinkWinsock(0).State <> 0 Then DataLinkWinsock(0).Close
    If Winsock(0).State <> 0 Then Winsock(0).Close
    If DVTSID(0).State <> 0 Then DVTSID(0).Disconnect
    If DVTSID(1).State <> 0 Then DVTSID(1).Disconnect

    'Set IP addresses. Camera1IPAddress was set from the .ini file
    'during routines that run in the Setup Screen.

    DataLinkWinsock(0).RemoteHost = Camera1IPAddress
    Winsock(0).RemoteHost = Camera1IPAddress
    DVTSID(0).RemoteHost = Camera1IPAddress
    DVTSID(1).RemoteHost = Camera1IPAddress

    'Calls simple routines that connect each of the following devices to
    'the DVT Camera:
    'DVTSID(0): Current Image Display
    'DVTSID(1): Most Recent Failed Image Display
    'DataLinkWinsock(0): One way communication path through the DVT
    '  Data Link driven port. This port is used only to send inspection
    '  result strings from the camera to the Inspection Program
    'Winsock(0): Two way connection to the Command Terminal of the camera.
    '  Any of the built in DVT Comm. Term. commands can be sent to the
    '  camera and complex string parsing routines can be set to run to
    '  interpret the returned data. Not heavily used in this program.

    Call ConnectToWinsock(DVTSID(0))
    Call ConnectToWinsock(Winsock(0))
    Call ConnectToWinsock(DVTSID(1))
    Call ConnectToWinsock(DataLinkWinsock(0))

    'Makes certain that the DVTSIDs are not displaying live images

    DVTSID(0).StopImages
    DVTSID(1).StopImages

'Database Initialization

    'This calls the routine that first checks for the existence of the
    'database. Then either opens or creates the database. Next it checks
    'for the existence of both tables and then either opens or creates the
    'tables. It assigns all tables to an DAO Recordset object.

    Call OpenDatabase(DatabasePath)

'Check for Open Reels

    'Check to see if there is a reel that is open.
    'If an open reel is found, the routine will
    'change ReelOpen=True and Paused=True

    Call CheckForOpenReel

    Initialize = True      'No Initialize Errors
    Exit Function

Handler:
    Initialize = False    'Error Handler. Initialize error occurred

End Function

```

Figure A-1. MainMenu Initialize subroutine.

```

Public Sub CreateReelTable(ByRef CreateDb As Database)
    'This creates the RunData table. It is only called if the
    'table could not be opened.
    Dim NewTbl As TableDef, F1 As Object, F2 As Object, F3 As Object, F4 As Object

    Dim Idx1 As Index

    'Set the new table to the database and name it.
    Set NewTbl = CreateDb.CreateTableDef("ReelTable")

    'Create field objects
    Set F1 = NewTbl.CreateField("Index", dbLong)
    'F1 will be an Index, so this sets it to automatically increment
    'for each new record
    F1.Attributes = dbAutoIncrField
    Set F2 = NewTbl.CreateField("Order Number", dbText)
    Set F3 = NewTbl.CreateField("Reel Number", dbText)
    Set F4 = NewTbl.CreateField("Part Number", dbText)
    Set F5 = NewTbl.CreateField("Start Time", dbDate)
    Set F6 = NewTbl.CreateField("End Time", dbDate)
    Set F7 = NewTbl.CreateField("Total Part Count", dbLong)
    Set F8 = NewTbl.CreateField("Failed Part Count", dbLong)

    'Append field objects to the new table
    NewTbl.Fields.Append F1
    NewTbl.Fields.Append F2
    NewTbl.Fields.Append F3
    NewTbl.Fields.Append F4
    NewTbl.Fields.Append F5
    NewTbl.Fields.Append F6
    NewTbl.Fields.Append F7
    NewTbl.Fields.Append F8

    'Append the new table to the database
    CreateDb.TableDefs.Append NewTbl

    'Set up appropriate MS Access database field properties for the
    'Index field
    Set Idx1 = NewTbl.CreateIndex("Index")
    Idx1.Primary = True
    Idx1.Unique = True
    Set F1 = Idx1.CreateField("Index")
    Idx1.Fields.Append F1
    NewTbl.Indexes.Append Idx1
End Sub

```

Figure A-2. CreateReelTable subroutine.

```

Public Sub OpenDatabase(Filename As String)
' This Subroutine first looks for the database file. It sets it or creates it
' and sets it to DataBase1 depending on if it exists. Then each of the tables
' (Failures and Reel) are searched for in turn. If a table is not found, it
' is created by calling the CreateReelTable or CreateFailuresTable subroutine.
' The table is then set to the corresponding RecordSet.

    Dim DBTableExists As Boolean

    Dim i As Integer

    'DBEngine is a class member of DAO
    Set DataBaseWorkSpace = DBEngine.Workspaces(0)

    If Len(Dir(DatabasePath)) = 0 Then 'If Database does not exist
        'create the database
        Set DataBase1 = DataBaseWorkSpace.CreateDatabase(Filename, dbLangGeneral)
    Else
        'open and set preexisting database
        Set DataBase1 = DataBaseWorkSpace.OpenDatabase(Filename)
    End If

    'Check for reel table -- create if it does not exist else just set it
    DBTableExists = False
    For i = 0 To DataBase1.TableDefs.Count - 1
        DBTableExists = DBTableExists Or (DataBase1.TableDefs(i).Name = "ReelTable")
    Next i

    If Not DBTableExists Then
        Call CreateReelTable(DataBase1)
    End If

    Set ReelTable = DataBase1.OpenRecordset("ReelTable", dbOpenDynaset)

```

Figure A-3. OpenDatabase subroutine.

APPENDIX B

FLOWCHARTS

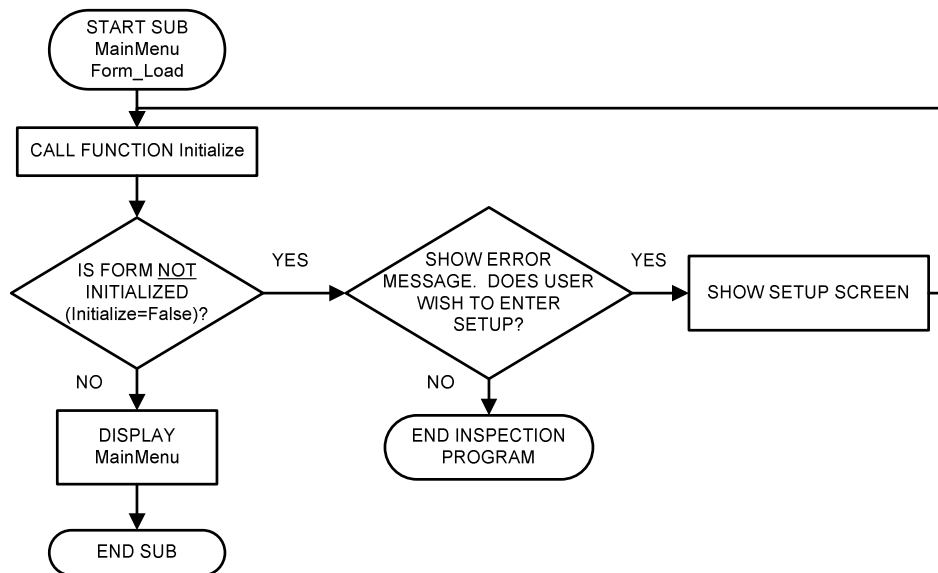


Figure B-1. Flowchart for MainMenu Form_Load subroutine.

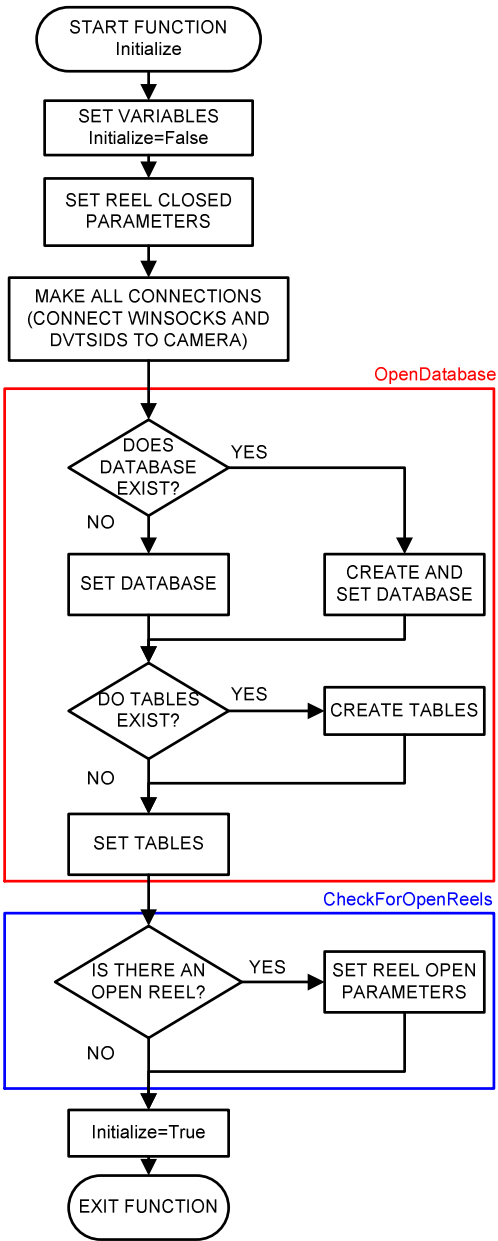


Figure B-2. Flowchart for MainMenu Initialize subroutine.

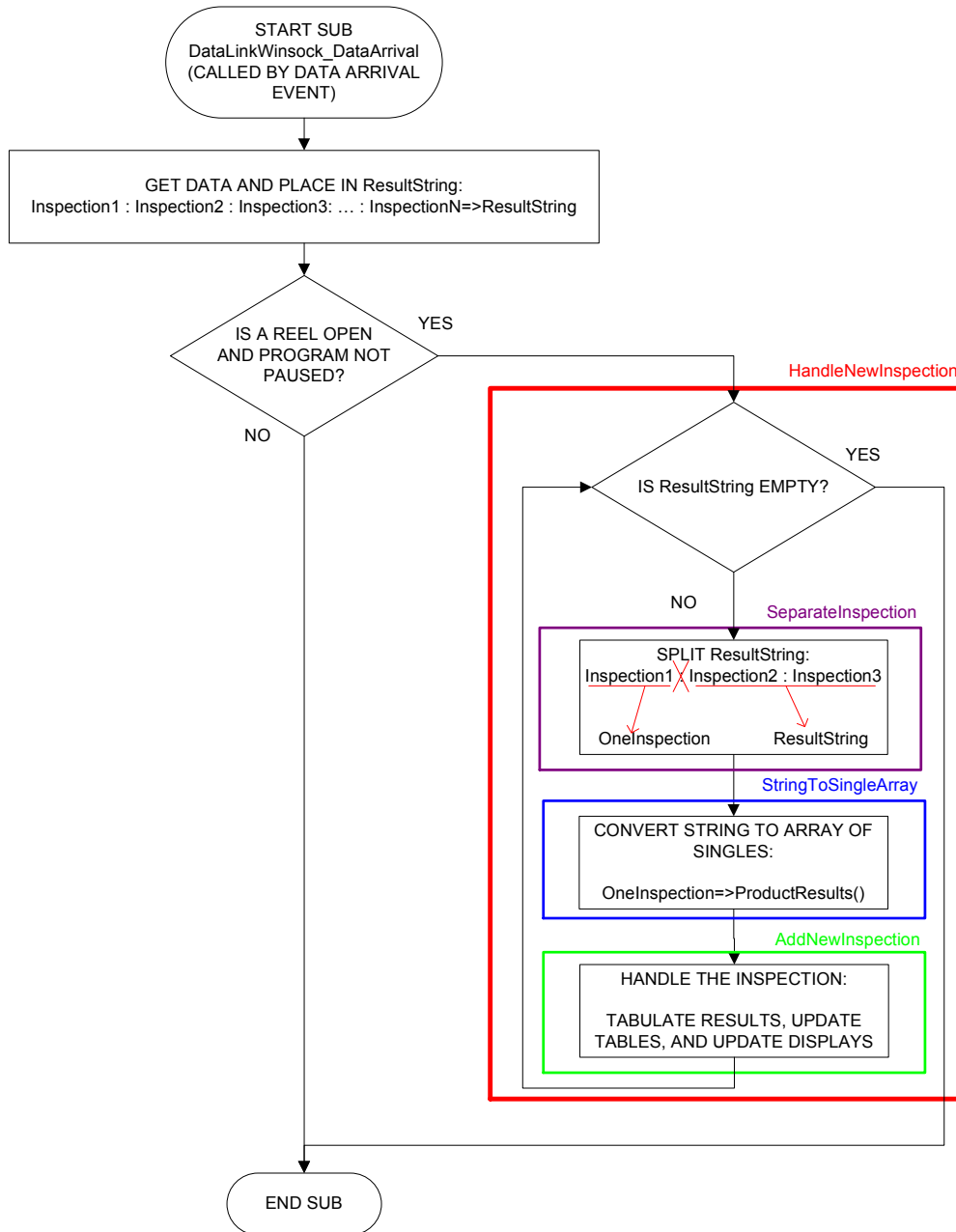


Figure B-3. Data handling flowchart.

APPENDIX C

SYSTEM WIRING DIAGRAM

Please note that not all fuses, switches, and circuit breakers are shown in this diagram. 120VAC power to the PC and the 24VDC power supply are not shown. Non-connected pins for the high speed counter are not shown.

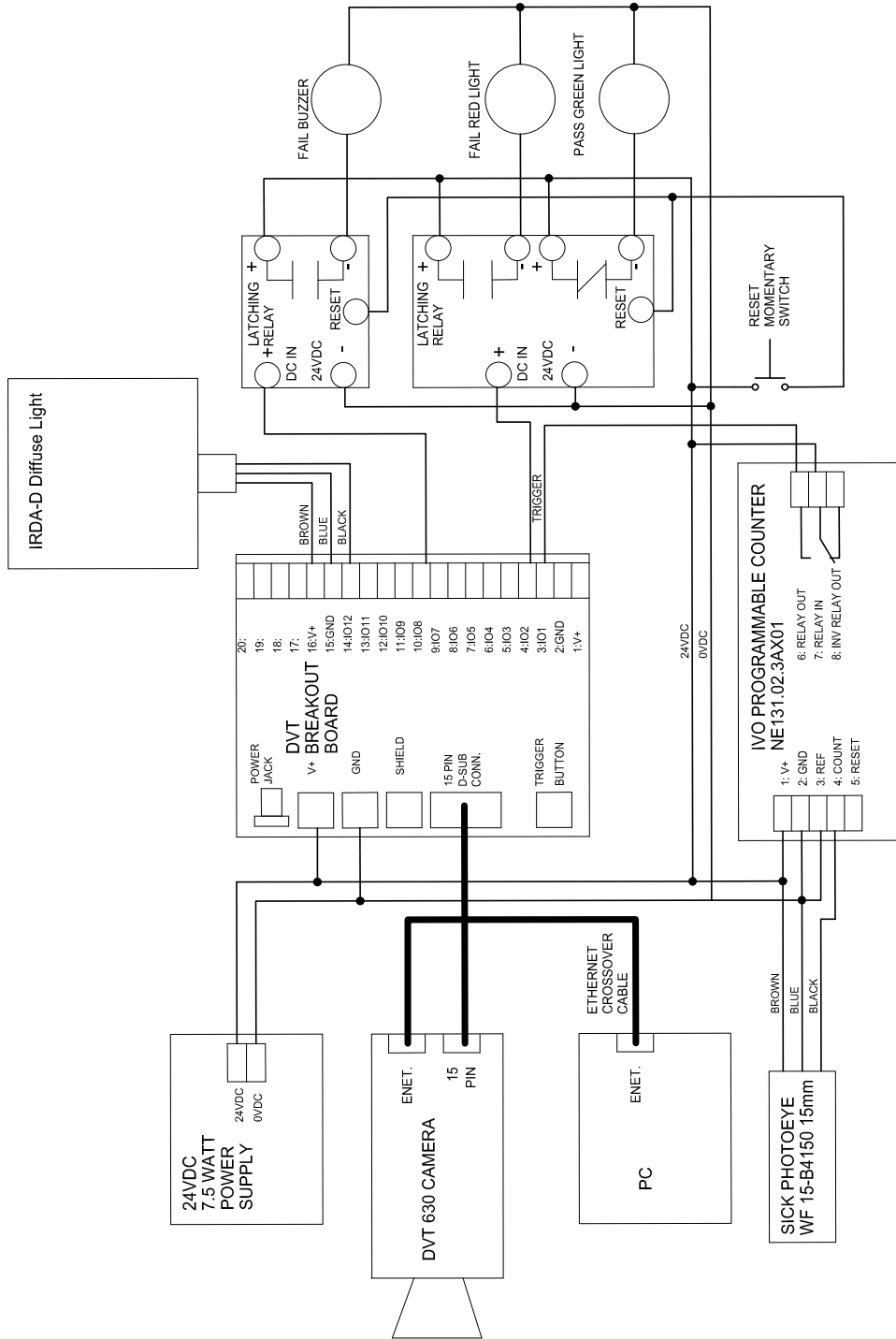


Figure C-1. System Wiring Diagram